Macro System Overview

Reference

Visual Cafe provides an extensive macro capability.   Macros are used to automate common, repetitive, or lengthy editing tasks.   Additionally macros can perform most Visual Cafe commands and can therefore be used to automate many Visual Cafe operations.

Macros are programmed in Symantec Basic.   This Basic language has been extended to include over 200 commands that can be used to control the Visual Cafe editor and environment.   Macros can edit text, display and manage dialogs, work with directories and files as well as control the Visual Cafe environment.

Once a macro has been created it can added to the macro menu and the user may assign keyboard shortcut to activate the macro directly.   Managing macros and assigning them to menus are done in the ScriptMaker dialog.

Macro support includes a macro recorder, a macro editor/debugger, and a macro dialog editor.   The macro recorder allows you to record an edit session or a sequence of commands and play them back later.   Macros created with the recorder can be edited, enhanced and debugged using the macro editor.   This macro editor includes a full set of macro debugging tools and provides access to the macro dialog editor.   Use the dialog editor to add an advanced user interface to your macros.

Macro Tools information
Macro Language information

Creating a New Macro

You can create a new macro by using the Macro Recorder or by creating a new macro from an existing one using the tools provided in the Symantec Basic Macro Editor.

Though it is generally easier to use the Recorder, as you become proficient with Symantec Basic macros, you will occasionally need to create a macro without using the Recorder.

See Also:

Symantec Basic Function Library

Symantec Basic Language Reference

Macro Editor Menu Commands

Macro Editor Toolbar

Creating a New Macro from an Existing One

Overview

Reference

In some cases, a macro you have written or recorded previously may be similar to the one you want to create. Use the Duplicate button in the ScriptMaker dialog box to create a copy of an existing script under a new name.

To create a new macro from an existing one:

1. From the main menu choose File, then select Macro, then ScriptMaker.

2. Select the existing macro you want to use as a basis for your new macro from the Macros list box. Click the Duplicate button. The Rename/Duplicate Macro dialog box appears.

3. Type a Macro name and a File name for the new macro. Click OK.

4. Click the Edit button. Make the modifications you want using the tools and commands in the Macro Editor.

Editing a Macro

If one of your Symantec Basic macros doesn't work properly in all circumstances, you can use the Macro Editor to modify its behavior. You can change how a macro works by editing the Symantec Basic statements that are executed when you run that macro.

You use the Macro Editor to edit Symantec Basic macros in much the same way as you edit text in a word processing program. You can do the following:

- Type new code.
- Select code and then delete, move, or copy it.
- Use the standard Windows keystrokes or mouse actions to edit Symantec Basic macros.
- Use the Windows Clipboard to move or copy code within your Symantec Basic macro.

To edit a Symantec Basic macro file:

1. From the main menu choose File, then select Macro, then ScriptMaker.

2. Select the macro to edit from the Macros list box, then click the Edit button.

3. Symantec Basic then starts the Macro Editor and displays your macro.

Use the tools and commands in the Macro Editor to edit, run, and test the macro.

See Also:

[Macro Editor Menu Commands](#)
[Macro Editor Toolbar](#)
[Testing a Macro](#)

Testing and Debugging a Macro

If you use Symantec Basic macros only to automate tasks, you won't need to worry about the quality of your macros. You can always replace a flawed macro with one that works correctly.

There are two scenarios for using Symantec Basic macros that can cause you to spend a little extra time testing to make sure that your macro code works properly under varying conditions:

- If you are attempting to automate a complex task whose successful completion is critical to your project.
- If you are creating macros that will be run by others, whose computers might be set up differently than your own PC.

To test whether your macro does what you intended, run it in the Editor. If you have a problem, single step through your program to track it down.

To run your script from the Macro Editor choose Start from the Run menu or presss the Run button on the toolbar.

After you or another user of your macro finds an error in your code, you can use Symantec Basic's debugging tools to single step through your macro and locate the source of the error. Then use the Symantec Macro Editor to modify your macro accordingly.

See Also

[Start (Run Menu)](#)
[Debugging Macro Errors](#)

Debugging Macro Errors

Overview

Reference

See Also

Three types of errors can occur in your Symantec Basic macros:

- Syntax errors occur when you enter Symantec Basic statements or functions incorrectly. Symantec Basic checks for syntax errors each time you attempt to run or close a macro file. If no errors are found, Symantec Basic allows you to complete your action. If it finds any syntax errors, Symantec Basic will display a dialog box that identifies the line number of the first error it encounters, along with a brief description.

- Run-time errors occur when Symantec Basic is running a saved macro file and encounters a statement or function it cannot execute. Symantec Basic will generate a run-time error. If you have added code to handle this error, your macro can recover from the error. If not, Symantec Basic will halt your macro before reaching the end of your code.

- Logic errors occur when Symantec Basic executes your macro without generating a run-time error, yet fails to accomplish the task your macro was designed to accomplish. This type of error is usually the most difficult to track down, because it is often not apparent where the problem lies within your macro code.

Use the debugging features of the Macro Editor to:

- Single step through existing macros one line at a time,
- Set and use breakpoints to execute up to a particular point in the macro,
- Add a Watch expression to show the current value of an expression.

If you are developing Symantec Basic macros that will be run by other users, write code that checks for run-time errors and responds gracefully to the errors that do occur.

The Symantec language provides the following statements and functions to help you respond to run-time errors that might occur when your Symantec Basic macros are running:

On Error Goto	This statement specifies the name of the error handling routine that will take control if a run-time error occurs.
Err	This function returns a number representing the type of error that occurred.
Error	This function returns an error message describing the error that occurred.

To implement run time error handling in a Symantec Basic macro:

- Specify the routine within a Symantec Basic procedure that will be called when a run-time error occurs.

- Specify an error handler with an On Error Goto statement, as in the following statement: On Error GoTo HandleError
  When a run-time error occurs, this statement transfers macro execution to the specified line label. (HandleError: is the line label in the statement.) The run-time handler routine must be placed in the same macro in which the error occurred, above any other procedure.

- Provide code that tracks down the source of the run-time error.
  Symantec Basic provides the Err function to accomplish this.
  The Err function returns a number corresponding to the most recent run-time error.

- Include code that exits the error handler and resumes execution of your Symantec Basix macro at an appropriate point in the procedure.

See Also:

Err
Error
OnError
Testing and Debugging a Macro

Single Step Through a Macro

Overview

Reference

See Also

Single step through a macro to learn how different statements and functions work by watching Symantec Basic execute them.

To step through a macro:

1. Choose the Step Into command from the Debug menu, or press F8, or click on the Step Into screen icon.

2. Repeat step # 1 until you are ready to resume or halt the macro.

3. To resume the macro at full speed, press the Run Macro button on the toolbar.

4. To halt the macro, choose the Stop command from the Run menu, or press the Stop Macro button on the toolbar.

See Also:

Set and Use Breakpoints in the Macro Editor

It is often tedious to single-step through a macro from the beginning, especially when you know that the source of your problem lies elsewhere in your macro code. You can use a breakpoint to tell Symantec Basic to run a macro from the beginning and then pause execution at a specific statement, giving you an opportunity to begin debugging your code at that point.

To set and use a breakpoint:

1. Move the insertion point to the line in a Symantec Basic macro where you want to set a breakpoint.

2. Choose the Toggle Breakpoint command from the Debug menu. Or click the Toggle Breakpoint icon. Or press the Toggle Breakpoint key (F9).
    Any line on which a breakpoint has been set is displayed in red.

3. Choose the Start command from the Run menu or click the Run icon.
    Symantec Basic begins to run your macro but stops at the first line with a breakpoint without executing the statement on that line.

4. After Symantec Basic suspends macro execution when it reaches a breakpoint, you can take several different steps to try to isolate the problem in your macro code. Since a breakpoint has the effect of suspending macro execution just before executing the line on which the breakpoint was set, you can step through the line with the Single Step command.

Add a Watch Expression in the Macro Editor

A common source of errors in a Symantec Basic macro is an incorrect assignment of a value to a variable. You can discover the value assigned to any variable or array element by adding a watch expression to the watch pane of the Macro Editor's main window.

To open the Watch Pane and add a Watch Expression:

1. Choose Watch Variable from the Debug menu or click the Add Watch screen icon. Symantec Basic displays the Add Watch dialog box.

2. Choose any of the variables or array elements listed in the Variable Name list.

3. Click OK.

Using a Dialog Box in a Macro

When you are creating Symantec Basic macros to distribute to other users, you can improve the usability and effectiveness by allowing the user to interact with your macro.

Your macros can display dialog boxes you fill out to specify options. To use a dialog box in a macro you have two issues to consider:

- You need to determine which options to present in your dialog box
- You need to write code to respond appropriately when the user makes different selections from your dialog box.

You have the option of using Symantec 's Predefined dialog boxes, or if your macro needs more information than can be gathered by one of Symantec Basic's predefined dialog boxes, you can create a Custom dialog box.

Overview

Creating a Custom Dialog Box for a Macro

Overview

Overview

Overview

To create a custom dialog box for a macro:

1. Use the Dialog Editor to create the visual layout for your dialog box.
    Do this by choosing Insert New Dialog from the Macro Editor's Edit menu to begin designing a new dialog box.
    To create a dialog box template, resize the Untitled dialog box, give it a new title, and add controls to it by dropping them within the dialog box's borders.

2. Choose Exit & Return from the File Menu. The Dialog Editor inserts the dialog box template into the macro and automatically generates a template declaration for that dialog box (the lines that start with Begin Dialog and end with End Dialog). You can declare more than one template per macro as long as they have different names.

3. Write the Symantec Basic code needed to customize the presentation of the dialog box and to retrieve the choices made in response to the dialog box.

See Also:

[Creating a New Dialog Box](#)
[Adding Buttons and Boxes](#)
[Setting Attributes for Buttons and Boxes](#)
[Setting Dialog Box Attributes](#)

Symantec Basic Predefined Dialog Boxes

You can gather a great deal of information from a user by using one of the Symantec Basic statements or functions that display predefined dialog boxes. What's more, it's easy to use these predefined dialog boxes in your macros.

Statement/Function

	Basic Contents	Returns...
MsgBox	Message only	Nothing
MsgBox()	Message and command button	Number of selected button
AnswerBox( )	Message or prompt, with up to three command buttons	Number of selected button
AskBox$( )	Message or prompt, with text box	Contents of text box, or ""
InputBox$( )	Message or prompt, with text box	Contents of text box, or ""
AskPassword$( )	Message or prompt, with text box	Contents of text box, or ""
PopupMenu( )	Menu made up of array elements	Index of selected element
SelectBox( )	Dialog box name, list box, and message	Index of selected element

All of Symantec Basic's predefined dialog boxes, except for the progress message dialog box, are *modal*, which means that Symantec Basic stops executing statements until the user clicks one of the dialog box's command buttons. With *modeless* dialog boxes, the macro continues to execute statements while the dialog box is displayed.

Unless otherwise stated, each statement or function displays a dialog box which is sized to fit the message and the command buttons, but its maximum size is five-eighths of the width and three-fourths of the height of the screen. The widest button determines the width of the other buttons. When the message is long, it is word wrapped. In most dialog boxes, you can use **Chr$**(13) + **Chr$**(10) to include a carriage return/linefeed in the message when you want to specify more that one line. The font in the dialog box is eight-point Helvetica.

Using the Macro Recorder

As you use Visual Cafe to create and edit program code, you will find yourself performing some tasks again and again. You can automate the task by turning on the Macro Recorder and having it create a macro for you.

To create a macro, choose File from the main menu, then select Macro, then Record Macro. Now perform your task. The Macro Recorder records your keystrokes and many mouse actions. After completing the task, choose File from the main menu, then select Macro, then Stop Recording.

The Recorder automatically translates the keys you pressed and the actions you performed with the mouse into Symantec Basic statements and inserts these statements into a macro that is saved for use later. After creating a macro using the Recorder, repeat the task you recorded by running the macro. From the main menu choose File, then select Macro, then Play.

To record a macro:

1. From the main menu choose File, then Macro, then Record Macro.
   A message box appears asking if you want to record over the existing default macro.

2. Click OK.
   You are now recording your macro.

3. Use the keystrokes and mouse actions you want to incorporate into your macro.

4. When you are done choose File, then Macro, then Stop Recording.

NOTE: Recording Mouse Activity is subject to some restrictions.

Although you could type in the Symantec Basic statements corresponding to a macro, it is much simpler to use the Recorder to record the macro and generate the corresponding statements. You can use the statements in the macro to make general adjustments to an application window, or you can use them as a skeleton into which you add other statements that control dialog boxes and their components.

See Also:

[Creating a New Macro from and Existing One](#)
[Recording Mouse Activity](#)

Recording Mouse Activity

The macro recorder will successfully record every mouse action that ultimately will result in one of the following:

- A menu selection
- Window activations
- Interaction with any kind of control in a dialog, such as edit controls, combo boxes, list boxes, radio buttons, check boxes

The Macro Recorder will not record moving, dragging, and resizing action.

So, in short, any activity that can be translated into a higher-order command will be successfully recorded, any other activity will not be recorded.

Creating a New Dialog Box for a Macro

You can use Macro Dialog Editor to create a dialog box from "scratch." (If a dialog box already exists that is similar to the one you want to create, you may find it easier to modify the existing dialog box and then save it under a new filename.)

To create a new dialog box:

1. Choose Insert New Dialog from the Macro Editor's Edit menu to open the Dialog Editor window.
   Before starting a new dialog box, the Dialog Editor should display "Untitled" to indicate that no template file is currently open.

2. Determine the size, position, title, and other attributes of your dialog box by Setting Dialog Box Attributes.

3. Add components to your dialog box by creating new ones, duplicating existing ones, or pasting them from another file.

4. Test your dialog box using the Test icon.

5. When you're satisfied with the way your dialog box looks, choose Save As from the File menu and assign it a filename or choose Exit & Return from the file menu to insert the new dialog box into the macro.

TIP: To avoid accidental loss of your work, you can save your template from time to time while you're working on it.

Adding Buttons and Boxes to Macro Dialogs

You can add controls to a dialog box by dropping them within the dialog box's borders. Use the commands from the Controls menu or click the corresponding icon in the toolbar to select a box or button type from among the following:

- OK and Cancel buttons, as well as any number of Push buttons that you label.
- Groups of option buttons allowing you to select one option from each group.
- Check boxes you can either check or uncheck.
- Group boxes to surround and label related controls (such as all the option buttons in a group or a series of check boxes).
- Edit controls (or text boxes) allowing you to type text.
- List boxes for you to select one item from a list.
- Combo boxes for you to either select an item from a list or to type text.
- Static text providing labels for edit controls, list boxes, and combo boxes or displaying free-standing text.
- Pictures to display images.
- Picture buttons that are like push buttons but contain an image.

As you add controls to the dialog box, the status bar at the bottom of the Dialog Editor main window provides information about the position and size of the control currently selected; if the dialog box itself is currently selected, the status bar shows you its position and size.

To create a component:

1. Click the component on the toolbar or else choose it from the Item menu. (See The Dialog Editor Toolbar for the toolbar button associated with each component.)
    The mouse pointer changes into cross hairs and a symbol representing the component.

2. Position the pointer inside the dialog box and click your primary mouse button.
    The component appears in the dialog box. The first two numbers displayed in the Dialog Editor status bar indicate the current position, in dialog units, of the upper-left corner of the component. The second two numbers indicate its size, also in dialog units.

3. If necessary, resize the component.

4. Specify the other attributes of this component as explained in Setting Attributes for Buttons and Boxes.

TIP: Try to add components in the order that you want used for the dialog box's tab sequence. Also, try to add a text component that is to serve as the label for a text box, list box, or combo box immediately before you add the component it labels. Otherwise, you'll need to change the order of your components.

TIP: If your dialog box is to include option buttons and you're using the default field name (OptionGroup*n*) that Dialog Editor supplies, create all option buttons that are to be in the same group consecutively; if you want a second group of option buttons in this dialog box, create at least one different component after the last option button in the first group and before the first option button in the second group. To ad an option button to a group later, duplicate one of the existing option buttons.

See Also:

Editing Buttons and Boxes in a Macro Dialog Box

Resizing Buttons and Boxes

Moving and Aligning Buttons and Boxes

Removing Buttons and Boxes

Setting Attributes for Buttons and Boxes

Overview

Resizing Buttons and Boxes in Macro Dialogs

Overview

Overview

Overview

You can increase or decrease the width of a component so that its label is displayed properly. In the case of a push button, group box, text component, list box, or combo box, you may also want to increase or decrease its height. Or you can resize a component so that it is the same size as another component.

To resize a component:

- Position the mouse pointer on the border of the component and drag the border until the component is the desired size.
  The mouse pointer becomes a two-headed arrow when it is on the border. You can change a component's width by dragging a side border, its height by dragging the top or bottom border, or both the width and height by dragging a corner.

The Dialog Editor status bar indicates the component's width and height, in dialog units. (If you prefer, you can resize a component by using the Width and Height text boxes in its information dialog box, as explained in Setting Attributes for Buttons and Boxes

 TIP: You can make a text component display on multiple lines by decreasing its width and increasing its height.

To make two components the same size:

1. Select the component that is the desired size and note its width and height in the status bar.

2. Resize the other component until it has the same width and height.

 TIP: Another way to create two components of the same size is to duplicate a component.

See Also:

[Moving and Aligning Buttons and Boxes](#)
[Duplicate](#)
[Removing Buttons and Boxes](#)

Moving, Aligning Buttons and Boxes in Macro Dialogs

As you refine your dialog box design, you'll want to adjust the position of components so that they are evenly spaced and aligned with one another.

To move a component:

- With the mouse pointer anywhere inside the component, drag the component to the desired location.

TIP: If you want to move the component *only* horizontally or *only* vertically, hold the Shift key down while dragging it.

The first two numbers displayed in the Dialog Editor status bar indicate the position, in dialog units, of the upper-left corner of the component.

To align two components vertically:

1. Select the component that is in the desired position and note its X number in the status bar.

2. Drag the other component until it has the same X number.

To align two components horizontally:

1. Select the component that is in the desired position and note its Y number in the status bar.

2. Drag the other component until it has the same Y number.

NOTE: If you prefer, you can move and align components by using the X and Y text boxes in the information dialog box, as explained in Setting Attributes for Buttons and Boxes.

Removing Buttons and Boxes from Macro Dialogs

If you don't want an existing component included in your dialog box template, you can either delete it entirely, or you can remove it while copying its declaration to the Windows Clipboard. This second method is convenient if you want to use the component in a different template. Both methods let you remove either a selected component or all components.

To delete a component entirely:

1. Select the component. To delete all components, select the dialog box itself.

2. Choose Clear from the Edit menu.

To remove a component but copy it to the Clipboard:

1. Select the component. To remove all components, select the dialog box itself.

2. Choose Cut from the Edit menu.

You can now paste the component or (if you removed all components) template declaration from the Clipboard into any script.

NOTE: When the dialog box is selected, the Clear and Cut commands remove all components but do not affect the size and location of the dialog box itself.

Setting Attributes for Macro Dialog Controls

Use the information dialog box to change the attributes of a button or box.

To set component attributes:

1. Double-click the component or, with the component selected, click the Info icon on the toolbar or else choose Info from the Edit menu.

2. Using the information dialog box that appears, change the component's attributes as necessary.

    *   The X, Y, Width, and Height text boxes indicate the current position and size of the component. If you want this component to be aligned with another component, make sure both components have the same X value (for vertical alignment) or Y value (for horizontal alignment); if you want this component to be the same size as another component, make sure both components have the same width and height values.

    *   If the information dialog box includes a Text$ text box, use it to assign the component a label (or, in the case of the Text Information dialog box, to create either a label or standalone text to be displayed). If you want to assign an accelerator key to a label, type an ampersand (&) in front of the desired character; for example, Sort &Order makes the O an accelerator key.

    *   If you want the label (or display text) to be a variable, check the Variable Name check box. (However, you cannot assign an accelerator key, except in the macro itself, if you check the Variable Name check box. To assign an accelerator key to a variable, use the ampersand when you assign a value to the variable in your macro, above the declaration for this template

3. Click OK.

The information dialog box closes, and the component appears as defined. If the component has a label or if you added a text component to the dialog box, it may be necessary to resize the component so that all of its text is visible.

Overview

Setting Macro Dialog Box Attributes

Overview

Overview

The dialog box size, position, text for the title bar, and template name are all attributes that you control.

To set the dialog box attributes:

1. If you want to resize the dialog box you are creating, position the mouse pointer on its border and drag the border until the dialog box is the desired size. The mouse pointer becomes a two-headed arrow when it is on the border. You can change a dialog box's width by dragging a side border, its height by dragging the top or bottom border, or both the width and height by dragging a corner. The Dialog Editor status bar indicates the dialog box's width and height, in pixels.

2. If you want to change the position of your dialog box, place the mouse pointer on the title bar (or on any empty spot within the dialog box) and drag the dialog box to the desired location. The first two numbers displayed in the Dialog Editor status bar indicate the position, in dialog units, of the upper-left corner of the dialog box. This will be the position of your dialog box when the user opens it.

TIP: If you want to move the dialog box only horizontally or only vertically, hold the Shift key down while dragging it.

3. Double-click on any empty spot inside the dialog box or, click the Info icon on the toolbar, or else choose Info from the Edit menu. (First make sure the dialog box itself is selected, by clicking once on any empty spot inside it.)
    The Dialog Box Information dialog box appears.

4. In the Text$ text box, type either the title to appear in the dialog box title bar or the variable name for the title.

5. If the dialog box title is a variable, check the Variable Name check box.

6. In the Name text box, type the name you want assigned to this template.

7. Click OK.

NOTE: Rather than using your mouse to position and size the dialog box, you can use the X, Y, Width, and Height text boxes in the Dialog Box Information dialog box. The information displayed in these text boxes is the same as the information appearing in the Dialog Editor status bar.

See Also

[Creating a New Dialog Box](#)

[Overview]

Incorporating a Dialog Box into a Macro

[Overview]

[Overview]

[Overview]

In addition to pasting the declaration of the dialog box template into your script, which the Dialog Editor automatically does when you choose Exit and Return, you must add lines to your script both above and below the template declaration. Above the declaration, you need Dim statements to declare each variable used in the template as well as statements assigning values to those variables. Below the template declaration, you need to insert the following lines:

- A Dim statement to declare an instance of the template.
- Statements assigning initial values (if desired) to check boxes, option button groups, and text boxes.
- Post-processing statements to retrieve the user's dialog box input and perform actions based on that input.

See Also:

[Adding Buttons and Boxes](#)
[Setting Attributes for Buttons and Boxes](#)
[Dim](#)

Testing Your Dialog Box

While you're designing a dialog box, you can switch to test mode to see how your dialog box will actually look and behave. The Dialog Editor test mode lets you check and uncheck check boxes, select and deselect option buttons, type text in text boxes and combo boxes, and select items from combo boxes and list boxes. You can also check the navigation within your dialog box.

To test the dialog box:

1.   Click the Test icon on the toolbar or choose Test Dialog from the File menu.

2.   Test each component of your dialog box.

3.   When you're ready to leave test mode, repeat step 1.

4.   Make any adjustments your dialog box requires.

While you're in test mode, you cannot change the placement, size, or any other attributes of the dialog box or its components.

Here is a checklist of tests you can perform on your dialog box:

- Do component labels fit properly? If not, you need to resize components or the entire dialog box.
- Is the tab sequence correct? Also, does each accelerator key make the appropriate component active? If not, you need to change the order of your components.
- Do text boxes display an appropriate number of characters at once? If not, you need to increase or decrease their width to approximate the number of characters users are likely to enter. (As users type characters into a text box, it automatically scrolls to the right for them. However, you should allow the users to see as many characters as possible at once.)
- Are list boxes and combo boxes wide enough to display their items? Is their height appropriate to the number of items they will contain? If not, you need to resize them.
- Are your components properly aligned with one another? If not, you need to move them. (If you have a text component beside another type of component, you can align the components visually. If instead you align them by assigning them the same Y position, the text component will appear to be higher than the other component.)
- Is your dialog box visually appealing? If not, you can experiment with different sizes and positions for the components.

NOTE: While you're in test mode, clicking a push button does not carry out any action, and list boxes and combo boxes merely display line numbers rather than actual items. You need to add lines to your script to define what is to happen when a push button is clicked and what items are to appear in list boxes and combo boxes.

See Also:

Symantec Basic Language Reference

[Macro Structure](#)

[Case Sensitivity](#)

[Comments](#)

[Error Handling](#)

[Identifiers](#)

[Scope](#)

[Statements](#)

[Subroutines and Functions](#)

[Visual Cafe Commands](#)

[User Interface and Dialog Boxes](#)

Overview

Statements

Overview

Overview

[Overview]

Subroutines and Functions

[Overview]

[Overview]

[Subroutine and Function Overview](#)

[Subroutines](#)

[Predefined Subroutines](#)

[User-Defined Functions](#)

[Predefined Functions](#)

Macro Structure

The macro is the basic programming unit. A macro is an ASCII text file containing subroutines and functions, each of which performs a particular task. Each subroutine or function has a name and is executed when its name is used in another subroutine or function. For an explanation of the differences between subroutines and functions, see the Subroutine and Function Overview.

Every macro has a subroutine named **Main**. **Main** controls the macro's execution. It is the first to be executed and it causes other subroutines and functions to be executed by "calling" their names. **Main** calls the other subroutines and functions or they call each other. **Main** can be the only subroutine in the macro, but when there are other subroutines and functions, it is the last one listed in the file.

Main starts with `sub Main`. **Main's** last line ends the macro: `end sub`.

NOTE: If you are not writing long or complicated macros, **Main** is probably the only subroutine in your macro; the first line of your macro is **Sub Main**.

See Also:

Sending Keystrokes to an Application

Use the SendKeys statement to send keys to an active application.   Depending on the keys you send, you can perform many of the actions done by other built-in routines.

Typically, you add SendKeys calls to a script manually.   In some instances you might find it easier to record keystrokes as a macro, thus generating SendKeys statements in the script.

Keystroke Specification Format

DoKeys, and SendKeys use the same string format for specifying keystrokes. Keystrokes are specified as follows:

To specify any printable character from the keyboard, use that key (for example, "h" for lowercase h, and "H" for uppercase h).

To specify a sequence of keystrokes, append keystrokes, one after the other, in the order desired (for example, "asdf" or "dir /p").

The plus sign (+), caret (^), tilde (~), percent sign (%), parentheses, square brackets, and curly braces are used to specify keystroke combinations. For example "^d" indicates Ctrl+D (see below). To specify one of these characters as itself, a single or shifted keystroke with no special meaning, enclose the character within curly braces.   For example, "{(}" specifies a left parenthesis;   "{%}" specifies the percent symbol.

To specify keys that are not displayable characters, enclose the name of the key within curly braces.   For example, {ENTER} is the Enter key and {UP} is the UpArrow key. The following table lists these keys:

{BACKSPACE}	{BS}	{BREAK}	{CAPSLOCK}
{CLEAR}	{DELETE}	{DEL}	{DOWN}
{END}	{ENTER}	{ESCAPE}	{ESC}
{HELP}	{HOME}	{INSERT}	{LEFT}
{NUMLOCK}	{NUMPAD0}	{NUMPAD1}	{NUMPAD2}
{NUMPAD3}	{NUMPAD4}	{NUMPAD5}	{NUMPAD6}
{NUMPAD7}	{NUMPAD8}	{NUMPAD9}	{NUMPAD/}
{NUMPAD*}	{NUMPAD-}	{NUMPAD+}	{NUMPAD.}
{PGDN}	{PGUP}	{PRTSC}	{RIGHT}
{TAB}	{UP}	{F1}	{SCROLLLOCK}
{F2}	{F3}	{F4}	{F5}
{F6}	{F7}	{F8}	{F9}
{F10}	{F11}	{F12}	{F13}
{F14}	{F15}	{F16}	

To specify keystrokes combined with a modifier key, such as Shift, Ctrl, or Alt, precede the keystroke specification with "+", "^", or "%" respectively. For example, "+{ENTER}" means Shift+Enter, "^c" means Ctrl+C, and "%{F2}" means Alt+F2.

To specify a modifier key combined with a sequence of consecutive keys, group the key sequence within parentheses and precede it with either "+", "^", or "%".   For example, "+{abc}" means the Shift key is held down while the a, b, and c keys are typed in consecutive order; "^({F1}{F2})" means the Ctrl key is held down while the F1 and then the F2 keystrokes are specified.

The "~" can be used as a shortcut for embedding the ENTER keystroke within a key sequence. For example, "ab~de" means the Enter key is pressed after "ab".

To embed quotes, use two quotes in a row, for example, "This is a ""test"" of the system".

To repeat a keystroke, enclose the keystroke and a repeat count within curly braces.   For example, "{a 10}" means "Produce 10 "a" keystrokes"; "{ENTER 2}" means "Produce two ENTER keystrokes".

Warning: Binding Symantec Basic Macros to Keystrokes

Binding a user-defined Symantec Basic macro to a CTRL+KEY sequence might play back incorrectly. Use another keystroke sequence, or use a two-step keystroke sequence; for example: CTRL+M N

See Also:

[Keyboard Functions](#)

Overview

Error Handling

Overview

Overview

Overview

Symantec Basic supports   handling of run-time errors in a manner that conforms to the Visual Basic error handling model. Symantec Basic errors fall into these categories:

- Errors your scripts can trap have numbers between 10 and 1000.
- Errors your scripts cannot trap have numbers between 0 and 10. Internal errors and out of memory errors cannot be trapped by scripts.
- Error numbers defined by Visual Cafe are greater than or equal to 1000.

You can save and restore error traps within a user-defined subroutine or function.   Error traps are valid only during the execution of the current subroutine or function.

A script starts with the value of the most recent error set to 0 (meaning that no error has occurred so far).   When an error occurs, the script's error value changes to the number for that error.   The script's error value is reset to 0 by each Resume statement, and as each function or subroutine ends.

If an error occurs within a user-defined error handling routine, error trapping is disabled and the script terminates with a run-time error.

Built-in Error Handling Statements

You use the following statements to handle trappable errors; see their descriptions in Function Reference for more information:

Statement	Purpose
On Error...Resume	Enables error trapping, and defines the action a script takes when an error is trapped. Resume 0 returns control to the statement that caused the error. Resume Next returns control to the statement after the that caused the error. Resume *label* returns control to *label*.
Erl	Returns a value of 0
Err()	Returns the error number of the most recently trapped error
Err	Sets the script's error number to the value returned by Err()
Error	Simulates the occurrence of the specified error
Error$	Returns the error message for an error

Overview

Statement Overview

Overview

Overview

Overview

A statement is an executable line of a macro. A carriage return/linefeed separates each statement from the one that follows it.

Often a statement is part of a construct, a sequence of statements that has a particular pattern or order. Constructs can control which statements are executed in which order. Constructs include such items as subroutines, functions and loops.

See Also:

[Constructs](#)
[Subroutine and Function Overview](#)
[Subroutines](#)
[User-defined Functions](#)
[Predefined Functions](#)

Statement Components

Statements are composed of reserved words and expressions.

A reserved word is a word that has a special meaning in the programming language and can be used only as its syntax allows. A reserved word cannot be used to name files, variables, and so forth.

An expression consists of one or more operands separated by operators and is evaluated to form a result. To use an example from algebra, $x + y$ is a numeric expression with two operands (x and y) and one operator (+).

See Also:

Operands

An operand used in a statement can be one of the following:

variable	A variable is the name of a location in memory that stores a value. The value of a variable usually changes during macro execution. The macro uses the name of the variable, such as *x*, to represent the value currently stored in *x*'s location. Every variable has a name, a value, and a data type. The *data type* tells what kind of value is stored in the variable. The data type determines how the value can be manipulated. For example, if a variable's data type is integer, its value can be added, subtracted, and so forth.
literal	A literal is a value of a particular type, rather than a representation of that value. An example of a *numeric literal* is the number 4. A numeric variable can store the value 4, but its name is not 4, the value itself. A *string literal* is the sequence of characters in the string. For example, "Hello, world." is a string literal. String literals are enclosed in delimiter characters. In Symantec Basic the only string delimiters used are the double quotation marks.
	In some user's guides, a literal is called a constant, because its value does not change. However, this guide uses the term *constant* only in the context of the predefined and user-defined constants explained briefly below and in depth later in More About Constants.
constant	A constant is like a variable in that it has a name and represents a value. Unlike the value of a variable, the value of a constant cannot change during the execution of the macro. Symantec Basic has both predefined and user-defined constants.
function	A function is a named sequence of statements that performs a task. The statements are executed when the name of the function appears in an expression. Symantec Basic has user-defined and predefined functions, both of which are explained in User-defined Functions and Predefined Functions.

See Also:

Operators

Operators

Operators indicate what operations, such as addition and subtraction, are to be performed on the operands in an expression. Some operators have different meanings depending on the data type being operated on. For example, the plus sign (+) indicates addition between numbers and concatenation between strings.

In general, an expression's operands and result must all be the same data type, and the operators must be valid for that data type.

The following are all examples of expressions.

```
'Numeric expression (result is number)
x + y

'String expression (result is string)
"Good " + "Day"

'logical expression (result is true or false)
'Abs is a function that finds absolute value
x > Abs(y)-5
```

The following outline of a macro contains one user-defined function and two subroutines. The syntax (rules for constructing) subroutines and functions is explained in the Subroutine and Function Overview.

```
Sub One ( )
    ...
End Sub
Function First ( ) As Integer
    ...
    First = ...
    ...
End Function
Sub Main
    ...
End Sub
```

See Also:

Operands

[Overview]

Data Types

[Overview]

[Overview]

[Overview]

Any (data type)

Boolean (data type)

Currency (data type)

Date (data type)

Double (data type)

Integer (data type)

Long (data type)

Single (data type)

String (data type)

Variant (data type)

See Also:

[Operands](#)
[Operators](#)
[Variables](#)
[Constants](#)

Variables

A variable is the name of a location in memory that stores a value. The value of a variable can change during macro execution. Every variable has a name, a data type, and a value.

To declare a variable is to provide its name and data type. Symantec Basic assumes the first appearance of a variable's name in a subroutine or function is its declaration. If that first use does not explicitly reveal the variable's data type, the compiler implicitly decides on a type.

The explicit declaration of a variable uses a type declarator or a **Dim** statement or both. For an implicit declaration, the first letter of the variable's name determines its data type. Any misspelling of a variable's name can become an implicit declaration of another variable.

Symantec Basic gives each variable an initial value at the time it is declared. A string variable is initialized to the empty string, a string with no characters (""). A numeric variable is initialized to zero. Symantec   has only local variables. In general, a local variable is a variable that is only known to and used by the subroutine or function where it is declared.

See Also:

Explicitly Declaring Variables

When the variable is a simple type (integer, long, single, double, or string), use a symbol called a *type declarator* after the variable's name the first time it appears in the macro. It does not have to appear in a particular kind of statement. The type declarator for an integer is %, for a long is &, for a single is !, for a double is #, and for a string is $. The compiler recognizes the first use of the variable as an indication of its existence and type. Subsequent uses of that variable do not need the type declarator.

For example, the following statements tell the compiler that the variables `Number_of_guests`, `Number_of_members`, and `Total_number` are of type long.

```
Number_of_guests& = 45
Number_of_members& = 100
Total_number& = Number_of_guests + Number_of_members
```

For any type of variable, you can use a **Dim** statement.

Syntax:

Dim *VarName* [**As** *type*] [**,** *VarName* [**As** *type*]]...

If you use a type declarator at the end of the variable's name, the `as type` clause is unnecessary, and vice versa. You can use both so long as they indicate the same type. All **Dim** statements appear inside user-defined functions or subroutines. You must use a **Dim** statement to declare an array. See Dim for details about array declarations.

Both of the **Dim** statements in the following example declare a string variable.

```
Sub Main
Dim first_name As String   'User's first name
Dim last_name$       'User's last name
...
End Sub
```

The **Dim** statement in the next example declares more than one variable.

```
Sub Main ()
Dim Total_number&, Number_of_guests&
...
End Sub
```

Using a separate **Dim** statement for each variable, along with an explanation of the variable's purpose at the beginning of each subroutine and function, makes the macro easier to debug and maintain.

Implicitly Declaring Variables

You can use a **Def** statement to specify a simple type and the initial letters for variables of that type.

Syntax:

Def_type letters_

where _type_ is replaced by **Int** for integer, **Lng** for long, **Dbl** for double, **Sng** for single, or **Str** for string, and _letters_ is replaced by a series of letters of the alphabet separated by commas. A range of letters can be specified by placing a hyphen between the first and last letters of the range. The syntax for _letters_ is:

letter [**-** _letter_] [**,**_letter_ [**-** _letter_]]...

The **Def** statements in the following example make any variables that are not explicitly declared into integers if their names start with I, M, or Q; into longs if their names start with A, B, C, or N; and into strings if their names start with T through Z.

```
DefInt I, M, Q
DefLng A-C, N
DefStr T-Z
Sub Main
...
End Sub
```

Def statements must appear outside of user-defined functions and subroutines, not within them. This makes them global type definitions that are valid for any subroutine or function that follows them. (It does not make the variables whose types are defined by the **Def** statement global variables.) Additional **Def** statements cannot contradict earlier ones. For example, you cannot define A-F as integers and later define C as a string. To use the same **Def** statements throughout the macro, make them the first statements in the macro.

NOTE: If a variable does not appear in a **Dim** statement, nor end in a type declarator, nor start with a letter listed in a **Def** statement, the compiler assumes it is an integer. Because of these implicit declarations, misspellings can result in new variables that you never intended. You can check your variable names if a macro compiles successfully but does not run correctly.

[Overview]

Constants

[Overview]

[Overview]

A constant is like a variable in that it has a name and represents a value. Unlike the value of a variable, the value of a constant cannot change during the execution of the macro. Symantec Basic has both predefined and user-defined constants.

See Also:

[User-defined Constants](#)
[Predefined Constants](#)

User-Defined Constants

User-defined constants are constants that you create outside of functions and subroutines. This makes them global constant declarations recognized by all the user-defined functions and subroutines that follow them. Each constant is valid only in the macro in which it appears.

If you use some string or numeric literal repeatedly, such as "Invalid input." or 459, define it as a user-defined constant. Using constants:

- Makes the code more readable. For example, 459 could become the constant Number_Of_Users.
- Saves memory because the literal is not repeated and, therefore, not stored in more than one place in memory.
- Allows you to change the literal by changing just one line rather than several lines throughout the macro. For example, you can change the message "Invalid input." to the more user-friendly message "You must enter a number." by changing only the line where the message is declared as a constant.

Syntax:

Const *constantName* **=** *expression* [**,** *constantName* **=** *expression*]...

The expression can use only string or numeric literals, the predefined constants TRUE or FALSE, or previously declared user-defined constants. Functions are not allowed. You do not have to use type declarators.

```
Const Message1 = "Are you sure?", Message2 = "Please wait..."
Sub Main
MsgBox Message2
...
End Sub
```

Predefined Constants

Predefined constants are reserved words in the language. They represent values needed in certain statements. On-line help explains each predefined constant as part of the statement that uses it. Each predefined constant has a numeric value. For example, ATTR_ARCHIVE, used in the FileList statement, has the value 32. However, you should use the constant name in your macros for readability and maintainability. The numeric values for predefined constants can change from version to version, but the constants' names do not.

Assignment Statements

The assignment statement is one of the most-often-used statements. It assigns the value of the expression on the right side of the assignment operator to the variable or element of an array on the left side of the operator. The assignment statement must do the following:

- Identify the variable or array element that receives the value.
- Use the assignment operator (=) to separate the variable or element from the expression.
- End with the expression that determines the variable's value.

Optionally, it can start with the reserved word **Let**. This word is left over from the earliest versions of BASIC.

NOTE: The assignment operator is the equal sign (=). The equal sign is also used as a relational operator that compares two quantities to see if they are equal. The difference is that the assignment operator gives a variable a value, and a comparison for equality returns a value of true (if equal) or false (if unequal).

The initial value of a numeric variable is zero. A string variable has the empty string as its initial value. An assignment statement changes that value.

The syntax for assigning a value to a variable is:

[**Let**] *varName = expression*

Both of the following examples assign the value 5 to x.

```
Let x = 5

x = 5
```

You can use a variable on both sides of the first assignment statement that uses it. For example, the following statement increases the value of the variable `Counter` by one.

```
Counter = Counter + 1
```

When this statement is executed, the value of the `Counter` on the right side is 0, its initial value, and the value of the `Counter` on the left is the sum of 0 + 1, which is 1.

See Also:

[Component Overview](#)
[Constructs](#)

Constructs

A construct is a sequence of statements that follow a pattern and serve a purpose within the macro. Failing to follow the pattern causes compiler errors. For example, control constructs control which statements in a macro are executed and which statements are not. They can choose a group of statements to execute from several such groups, repetitively execute a group of statements, and transfer control from one part of the macro to another. The control constructs are conditional constructs (such as **If** statements and **Select Case** statements), loops (such as **For**, **While**, and **Do** loops), goto statements (such as **GoSub** and **Goto**), and the **End**, **Stop**, and **Sleep** statements.

Subroutine and Function Overview

Subroutines and functions are very similar. Each is a sequence of statements that performs a task. Each has to be declared or defined, and each is executed when its name is used in another subroutine or function. Each can change the values of variables that are passed to it by reference. Passing parameters by reference is explained in Parameters in Calls.

The differences between subroutines and functions are:

- A subroutine's name never returns a value to the subroutine or function that calls it, and a function's name always does.

- The way a subroutine is called differs from the way a function is called. A subroutine's name appears in a **Call** statement. A function's name is part of an expression.

See Also:

[Calling a Function or Subroutine](#)

Subroutines

The parts of a subroutine declaration are:

- The statement that identifies it as a subroutine, tells the subroutine's name, and identifies its parameters.
- Executable statements.
- The statement that ends the subroutine declaration.

Syntax for subroutine:

Sub *subName* [**(** [*parameterList*] **)**]
   [*statements*]
End Sub

The syntax for calling a subroutine:

[**Call**] *subName* [**(** [*parameterList*] **)**]

Parameters are passed by reference unless explicitly passed by value. See <u>Parameters</u> for details about parameters and parameter passing.

The following example shows the declarations or definitions of the subroutines named **Square** and **Main**. **Main** is the first subroutine to be executed. The **Call** statement in **Main** calls the **Square** subroutine. **Square** squares the value of the variable *sum* that is passed to it as the parameter *x*. Since *sum* is passed by reference, changes made to its value by **Square** are known to **Main** as well. In this example, *sum* has the value 7 before the call to **Square** and the value 49 after the call.

```
'declaration of Square subroutine
Sub Square (x&)
    'The variable sum becomes known to Square as x
    x = x * x
End Sub
'declaration of Main subroutine
Sub Main ()
    ...
    x = 3
    y = 4
    'sum equals 7 here
    sum = x + y
    'Execution of Square occurs
    Call Square (sum)
    ...        'sum equals 49 here
End Sub
```

See Also:

Predefined Subroutines

A number of statements are really predefined subroutines. For example the FileList statement can be executed in either of the following forms:

```
FileList files, "c:\*.bat"
```

Or,

```
Call FileList (files, "c:\*.bat")
```

See Also:

[Calling a Function or Subroutine](#)
[User-defined Functions and Subroutines](#)
[Subroutines](#)

User-Defined Functions

You create a user-defined function to perform a task and return a value. Usually you create a function for a task that the macro needs to perform more than once.

A function declaration consists of:

- The statement that identifies it as a function, tells its name, identifies its parameters, and provides a simple type for the function's name as though it were a variable.
- Executable statements, one of which assigns a value of the correct type to the function's name. This value is returned by the user-defined function to the statement using the function.
- The statement that ends the function declaration.

Syntax:

Function *functionName* [**(** [*parameterList*] **)**] [**As** *type*]
    [*localDeclarations*]
    [*Statements*]
End Function

Each function has a simple type: string, integer, long, single, or double. Parameters are passed by reference unless explicitly passed by value. See Parameters for details about parameters and parameter passing.

The following example shows the declarations of the **Square** function and the **Main** subroutine. Each use of the function's name (**Square**) inside **Main** calls the function. A statement in **Main** uses **Square** twice in the same expression. **Square** is used as though it were a variable of type long because the function is type long, and the value assigned to **Square** inside the **Square** function is used to evaluate the expression inside **Main**. **Square** squares *a* (which is passed to it as a parameter the first time) and returns the value 9 (which is 3 squared) to the statement. Then **Square** squares *b* (which is passed to it the second time) and returns the value 16 (which is 4 squared) to the statement. The statement assigns the value 25 (9 + 16) to *c*.

```
'declaration of Square function
Function Square (x&) As Long
    'x takes the value of the a, then b
    Square = x*x
End Function
'declaration of Main subroutine
Sub Main ()
    ...
    a = 3
    b = 4
    'calls Square twice
    c = Square(a) + Square(b)
    ...
End Sub
```

See Also:

[User-defined Functions and Subroutines](#)

Predefined Functions

Symantec Basic has a large library of commonly used predefined functions.

A predefined function saves you time because you don't have to write it. To use one, you only need to know its purpose, syntax, and the type of result that it returns to your macro. You use the function as though it were its result. A function is not a statement and never appears alone on a line of your macro. Most often, you use it as part or all of an expression in an assignment statement.

In this example, the **Len** function counts the characters in a string and returns that length to the macro. This saves you the time it would take to write statements that count the characters in a string. Its syntax is **Len(***exprS***)**, where *exprS* is any string expression. To find the length of a string variable named VendorName, you would use something like:

```
Length = Len(VendorName)
```

If `VendorName` is "Ajax Corporation", the function would return the number 16 (15 letters and 1 space).

See Also:

Case Sensitivity

Symantec Basic is not case-sensitive. SUB MAIN is equivalent to Sub Main and sub main. You can type everything in one case or use capitalization to increase readability.

See Also:

Comments

Comments in a macro file are explanations of what the macro does. Because they are set off by special characters, the compiler ignores them. Good comments save debugging and maintenance time by explaining:

- The purpose of each macro at the beginning of the macro.
- The purpose and parameters for each subroutine and user-defined function before the subroutine or function starts.
- Every variable as it is introduced.
- The beginning and ending of each construct.

To comment a whole line or partial line:

- Start the line with **REM** followed by a space.

```
REM This macro performs...
```

Or,

- Start the comment with a single quotation mark (').

```
MsgBox "Hello, world!" 'Displays a string inside a message box

' This macro performs...
```

The compiler ignores all characters between the single quotation mark or **REM** and the end of the line.

To comment more than one line:

- Start the comment with /* and end it with */ as in the C programming language.

```
MsgBox "Hello, world!" /* This displays a string inside a message box. The macro
pauses until the user clicks the OK button. */
```

No statements can appear on the same line as the ending comment marker. The */ can be followed only by spaces and the carriage return.

See Also:

Identifiers

The names of variables, constants, user-defined functions, subroutines, and so forth, are called identifiers. An identifier starts with a letter of the alphabet, but subsequent characters can be alphabetic, the digits 0 to 9, or the underscore character (_). A variable identifier can have as many as 255 characters. Creating meaningful identifiers makes your macros easier to read. Using *x* and *y* as identifiers may be useful in a numeric expression, but they are rarely useful elsewhere. You cannot use characters (such as the period) or reserved words (such as **Main**) as identifiers.

No identifier can be duplicated within the scope of its owner. See Scope for more details. For example, a subroutine cannot have a variable name that is the same as the name of a user-defined function defined before the subroutine in the macro. However, two subroutines can have the same identifier for a local variable, even if the variables are not the same type.

The following example uses an identifier for each of three variables.

```
Total_number = Number_of_guests + Number_of_members
```

See Also:

Scope

A Symantec Basic macro uses static scoping and has no forward declarations. his means that functions and subroutines can call themselves and the components of the macro declared prior to them within the macro but cannot call a component that comes after them. The variables declared inside a subroutine or function are local. This means that they are used only by the subroutine or function in which they appear. A subroutine or a function never uses another's variables. However, the values of those variables or their memory addresses can be passed to the subroutine or function as parameters.

The following example of a macro shows where to put executable statements and declarations of various types.

```
'Def statements for entire macro
'constant declarations for entire macro
Sub One ( )
    'local variable declarations
    'executable statements
End Sub
Function First
    'local variable declarations
    'executable statements
    'assignment of a value to First
End Function
'constant declarations for Main subroutine
Sub Main ( )
    'local variable declarations
    'executable statements
End Sub
```

Main can call itself, **One** and **First** because the definitions of **One** and **First** precede **Main**. **First** can call itself and **One**, but it cannot call **Main**. **One** can only call itself.

The constants declared before **One** apply to the entire macro; those declared just before **Main** only apply to **Main** because only **Main** follows those declarations.

The local variables in **One** cannot be used by **First** or **Main** because no subroutine or function can see another's local variables.

See Also:

Overview

User Interface and Dialog Boxes

Overview

Overview

Overview

In Windows (and other graphical user interfaces), a dialog box is a special window displayed by Symantec Basic or some other application to communicate with a user. A dialog box displays messages for and requests data from a user. When that data is used to determine what statements to execute, the macro is said to be event-driven.

Symantec Basic has several simple predefined dialog box templates. You can display them from any macro by using the statements and functions provided for them. If you don't do a lot of programming in Symantec , the predefined dialog boxes may be all you need.

Symantec Basic also provides a Dialog Editor, a tool with which you can create your own templates for dialog boxes. Each dialog box template defines a dialog box's size, its components (such as push buttons and text boxes), the size and position of those components, and so forth. User-defined templates can be included in any macro and give you control over the look and feel of the dialog box and the amount of data that can be obtained from it. However, macros that use them are more complicated than those that use the predefined dialog box templates.

See Also:

Calling a Function

A function can be called by any subroutine or function that is declared after it in the script; a function can also call itself. When a function calls itself, it is recursive.

You call a function by using its name in an expression in the calling routine. As the expression is executed, control is transferred to the statements in the function's declaration. One of those statements is an assignment statement that assigns a return value to the function's name. After the function is executed, control is returned to the calling routine and the calculation of the expression is completed. Therefore, using the function's name in the expression is the same as using its return value in the expression.

Syntax:

... *functionName* [**(** [*parameterList*] **)**] ...

functionName	Name of function to be called.
parameterList	List of parameters to be passed to the function.

NOTE: If a 0 or empty string ("") is returned by a function, the function may be missing the assignment statement that gives the function's name a value.

Example

In the following example, the function's name is SquareRoot( ).

```
x = SquareRoot(y)
```

In the function declaration, the function's name is assigned a value.

```
Function SquareRoot (someNumber as double) as double
    ...
    SquareRoot = ... 'square root of someNumber
End Function
```

During the execution of the statements in the function declaration, the parameter y becomes someNumber, its square root is calculated, and the value of that square root is assigned to the name of the function.

See Also:

Calling a Subroutine

Parameters

Parameters in Calls

User-Defined Functions and Subroutines

User-defined Functions and Subroutines

User-defined functions and subroutines are control constructs that:

- Allow statements that are repeated in various parts of the script to be written only once.
- Make some variables invisible to other parts of the script, out of the scope of any other function or subroutine. These variables are not changed by any other function or subroutine unless they are passed to it for that purpose.
- Make the script is easier to understand, debug, and maintain.

With the exception of the Main subroutine that automatically begins execution when the script runs, all functions and subroutines are called by another function or subroutine. The one making the call is the calling routine and the one being called is the called routine. When a calling routine makes a call, its execution is put on hold until the called routine has been executed. Then execution of the calling routine continues with the statement after the call.

A script can have an unlimited number of user-defined functions and subroutines.

See Also:

Parameters

Parameters are values passed from one function or subroutine to another. The number and types of parameters in a function or subroutine call must match the number and types of those in the called routine's declaration. They must also be in the same order. The called routine identifies each parameter by its order in the call and uses its own name for the parameter no matter what the name of the parameter is in the calling routine.

A parameter can be passed either by value or by reference.

Parameters are passed by reference unless explicitly passed by value. There are two ways to pass a parameter by value. One way involves the syntax of the call; the other involves the declaration of the called routine. See Parameters in Calls and Using Parameters in Function and Subroutine Declarations.

Parameters that are passed to the called routine because their values are needed by the called routine to complete its task are input parameters. Parameters whose values are determined by the called routine for the benefit of the calling routine are output parameters.

Input parameters are often passed by value to protect the original. Output parameters are always passed by reference. When a parameter is both an input and output parameter, it is passed by reference as well. Arrays, because of possible size and memory requirements, are always passed by reference.

See Also:

[Calling a Function](#)
[Calling a Subroutine](#)
[Parameters in Calls](#)
[Function...End Function](#)
[Sub...End Sub](#)
[User-Defined Functions and Subroutines](#)

Parameters in Calls

The syntax for parameters used in a call is different than the syntax for parameters used in a function or subroutine declaration.

Syntax

A *parameter list* (see earlier syntax for functions and subroutines) is a series of zero or more parameters:

[*parameter* [**,** *parameter*]...]

The syntax for a parameter is:

[**(**] { *varName* | *expr* } [**)**]

You can force a parameter to be passed by value by putting parentheses around the parameter in the calling routine. This is in addition to the set of parentheses that may surround the entire parameter list. For example, Call Square(x)   does not force x to be passed by value, but   Call Square((x))   or its equivalent without the reserved word call   Square(x)   does force x to be passed by value.

When a parameter is passed by value, the parameter can be any expression. When it passed by reference, it should be the name of a variable. For example, the expression y + 7 can be passed by value, but not by reference, because it does not have a location in memory that can be accessed. The variable y can be passed either by value or by reference, because it is a variable name and, therefore, has a location in memory.

When a variable name is used as a parameter in the calling routine, it must already be declared (and therefore initialized) in the calling routine. Because it has already been declared, you never need use a type declarator or the As Type clause in a call.

To pass an entire array, you do not use empty parentheses after its name, as you would in the called routine's declaration. To pass an element of an array, you use the subscripts that identify that element. Subscripts are always in parentheses. For example, if Array1 is an array, you can pass it to the Report subroutine with:

Call Report(Array1)

or

Report Array1

To pass an element of that array Array1 (2, 3) to the Square subroutine, you would use:

Call Square(Array1(2, 3))

or

Square Array(2, 3).

Examples

In the following assignment statement, the expression y + 7 is passed by value to a function.

```
x = Test((y + 7), z)
```

In the following subroutine call, z is passed by value.

```
Call Sort (x, y, (z))
```

See Also:

[Calling a Function](#)
[Calling a Subroutine](#)
[Function...End Function](#)
[Sub...End Sub](#)
[Parameters](#)
[User-Defined Functions and Subroutines](#)

Macro Editor Menus

Menu	Item	Description
File	Exit and Return	Compiles the macro and returns to Visual Cafe. If the current macro has not been saved to a file, Macro Editor first gives you an opportunity to do so.
		If there is a syntax error in your macro, an error message will appear when the macro is compiled. The location in the macro script where the error was encountered will be marked and you will have an opportunity to correct the error.
Edit	Undo	This command undoes the last cut, paste, replace, or typed character. You can undo previous commands, up to the limit of the undo buffer, by repeatedly choosing this command.
	Cut	This command removes the selected text from the macro and places it on the Clipboard so you can move it to another point in the file. If there is no text selected the command is dimmed.
	Copy	This command places a copy of the selected text on the Clipboard. You can copy text within the same macro or from one macro to another. If there is no text selected the command is dimmed.
	Paste	This command inserts the contents of the Clipboard at the current cursor position. This command is always used in conjunction with the Copy or Cut commands. If there is nothing in the Clipboard the command is dimmed.
	Delete	Choose Delete from the Edit menu to delete the currently selected text. If no text is selected,the command deletes the character to the right of the cursor.
	Find...	This command displays the Find dialog box, where you specify the text you want to search for. The Macro Editor searches forward starting at the current cursor position and stops when it finds the string or reaches the end of the file.
		Click Match Case to find exact matches only.
		NOTE: Since the Macro Editor searches line by line, it will not find a character string that breaks onto more than one line.
	Find Next	Use this command to find the next occurrence of the character string you specified with the Find command.

	Replace...	Use this command to replace one character string with another in some or all of the places it occurs in your file. You specify both the text to search for and the text you want to replace it with. The Macro Editor starts from the cursor position and works forward through the file.
		After you have specified the search and replacement text, click the Find Next button to begin replacing text.
		The Macro Editor stops at each found occurrence to give you the option of replacing it. Click the Replace button to replace the selection. Click the Find Next button to move to the next occurrence of the text without replacing the current selection.
		Click Replace All to replace all occurrences of the text with the new string without any more prompting.
	Go to Line...	Use the Go to Line command to locate a line in the macro by number. The Goto Line dialog box appears. Enter the line number and click OK.
	Insert New Dialog...	This command starts the macro dialog editor with a new dialog template. Use the Macro Dialog Editor to create your own dialog boxes that can be incorporated into Macros.
	Edit Dialog...	This command opens the dialog editor and loads the selected dialog box template. (This command is only enabled if the code for a dialog template is currently selected.)
Run	Start	To run the macro that is currently displayed in the macro editor, choose Start from the Run menu or click the Run screen icon.
	Stop	To halt a macro in Run mode or Break mode, choose Stop from the Run menu or click the Stop screen icon.
	Check Syntax	Use this command to test the syntax of the macro. If an error is encountered the Error message box appears with the location and description of the error.
Debug	Watch Variable...	Choose Watch Variable from the debug menu or click the Watch icon to show the current value of an expression. Other information, such as type and length (for strings) is also displayed. If a variable is not in scope, its value is listed as `<Not in context>`.
	Delete Watch	This command deletes a variable from the watch window. Use this command once you have determined that the variable in question has the correct value.
	Modify...	Choose Modify from the Debug menu to modify the value of a variable. The Modify Variable dialog box is displayed. Select the variable you want to modify in the variable list. Enter the new value in the value box. Click OK.

	Step Into	This command steps through the macro line-by-line, tracing into user-defined functions and subroutines. If a call is made to another procedure, each line in the called procedure is executed individually before macro execution is suspended.
	Step Over	This command steps through the macro line-by-line without tracing into user-defined functions and subroutines. If a call is made to another procedure, the entire procedure is run before macro execution is suspended.
	Toggle Breakpoint	Choose Toggle Breakpoint from the Debug menu or click the Breakpoint icon to set or remove a breakpoint on the current line. When Symantec Basic encounters a breakpoint, it suspends the macro just before executing the line on which you set the breakpoint.
	Clear All Breakpoints	This command removes all breakpoints.
	Set Next Statement	This command moves execution to the line with the cursor, if the cursor is within the currently executing function or subroutine.
Help	Contents	Use the Contents command from the Help menu to open the Help Contents screen. The Contents screen contains a list of help topics. Click a topic in the list to get more information.
	Search for Help on...	Choose Search for Help On... from the Help menu to display the Help Index tab.
		Type the word you want to search for in the text box at the top. When you start typing, the words that most closely match the text you type are displayed in the keyword list box below.
		You can also enter a word to search for by choosing it from the index.
		Click Display when you selected the word for which you you want to search . All Help topics associated with the selected word are listed, and you can select one to view.

System Menu

Menu	Item	Description
System	Restore	Restores the window to its original size
	Move	Places the window into move mode, allowing the user to reposition the window
	Size	Places the window into size mode, allowing the user to resize the window
	Minimize	Select this option to iconize the window.
	Maximize	Causes window to resize to it's maximum possible size.
	Close	Closes the window.

Macro Dialog Editor Menus

Menu	Item	Description
File	New	Use the New command from the Dialog Editor File menu to start a new dialog box template.
	Open...	Use the Open command from the Dialog Editor File menu to open an existing dialog box template. This command displays the standard Windows Open File dialog box.
		The default extension for templates is .DLG. If you saved the file you want with a different extension, change the file extension in the File Types drop down list to *.* All Files.
		The Directory field indicates the current directory. If the file you want is not listed in the File Name list box, select a new directory from the Directories list or a drive from the Drives drop-down list.
		Select the file you want from the list in the File Name list box or type it directly into the File Name text box and Click OK.

	Update	Choose Update from the Dialog Editor file menu to update the dialog template. This adds the Symantec Basic statements that control the dialog to your macro.
	Save As...	Use the Save As command from the Dialog Editor File menu to save the currently displayed dialog box template to a new file. This command opens a dialog box for specifying the template's filename.
		The default extension for templates is .DLG; unless you assign a different extension, Dialog Editor automatically assigns .DLG as the file extension.
		The Directory field indicates the current directory. If you want to save to a different directory, select a new directory from the Directories list or a drive from the Drives drop-down list.
		Use this command when saving a newly created dialog box or to save an existing dialog box under a new file name.
	Test Dialog	Use the Test Dialog command from the File menu to see how the dialog box you're designing will actually look and behave. While in test mode, you can check and uncheck check boxes, select and deselect option buttons, select items from combo boxes and list boxes, type text in text boxes and combo boxes and check the tab sequence.
		A checkmark appears beside this command when you are in test mode, and all other commands are dimmed except for Exit on the File menu as well as the items on the Help menu. You leave test mode simply by choosing the Test Dialog command again.
		You cannot make any changes to the dialog box while in test mode. Only commands from the File menu and the Help menu are enable in the test mode.
	Capture Dialog	Use the Capture Dialog command to capture a dialog box from another application that you would like to use in your macro. When you choose the Capture Dialog command, the cursor changes to the capture cursor. Place the cursor over the dialog you would like to capture and click the left mouse button. A message box asks if you want to replace the dialog box in the Dialog Editor with the new dialog box. Click Yes.
	Exit and Return	Choose Exit and Return from the File menu to close the Dialog Editor application. If the current dialog box template has not been saved to a file, Dialog Editor first gives you an opportunity to do so.
Edit	Undo	Choose Undo from the Edit menu to undo the last cut, paste, replace, or typed character. You can

		undo previous commands, up to the limit of the undo buffer, by repeatedly choosing this command.
	Cut	Choose Cut from the Edit Menu to copy the currently selected component to the Windows Clipboard and then removes it from the dialog box template. If the dialog box itself is currently selected, all of the components are removed and copied to the Clipboard.
	Copy	Choose Copy from the Edit menu to copy the currently selected component to the Windows Clipboard; if the dialog box itself is currently selected, copies the entire template to the Clipboard.
	Paste	Choose Paste from the Edit menu to insert the contents of the Windows Clipboard into the Dialog Editor main window. If the Clipboard does not contain information that Dialog Editor recognizes as a template or component, an error message appears.
	Delete	Choose Delete from the Edit menu to delete the currently selected text. If no text is selected, the command deletes the character to the right of the cursor.
	Duplicate	Choose Duplicate from the Edit menu to copy the currently selected component, partially overlaying the original component with the duplicate. Dialog Editor assigns the same attributes to the duplicate as to the original.
		The Duplicate command is dimmed if the OK push button, Cancel push button, or the dialog box itself is currently selected.
	Size to Text	Choose Size to Text from the Edit menu to have the Dialog Editor automatically resize a box or button to fit the enclosed text string.
	Info	This command opens the information dialog box for the currently selected component; if the dialog box itself is currently selected, opens the Dialog Box Information dialog box. An information dialog box lets you review or change the attributes of the dialog box or the selected component.
	Grid	Opens the Grid Settings dialog.
Controls		Items on the Controls menu enable you to place controls in the dialog box. After you select an item Your mouse pointer turns into cross hairs and a control symbol until you create the control by clicking the mouse button. (The mouse button must be positioned within the borders of the dialog box,

but beneath its title bar.)

The Dialog Editor assigns the control a default label. You can change the label by using the Info command from the Dialog Editor Edit menu.

OK button	Use this command to add an OK push button to the dialog box template.
	The returned value for the OK button is -1.
	This command is dimmed once your template has an OK push button.
Cancel button	Use this command to add a Cancel push button to the dialog box template.
	The returned value for the Cancel button is 0.
	This command is dimmed once your template has a Cancel push button.
Push button	Use this command to add a push button to the dialog box template.
	The Dialog Editor assigns the push button a default label of "Push me." You can change the label by using the Info command from the Dialog Editor Edit menu.
	The pre-defined push buttons OK and Cancel are assigned return values of -1 and 0 respectively. Dialog Editor sequentially numbers all other push buttons starting from 1. The returned value for a push button is the number assigned.
Option button	Use this command to add an option button to the dialog box template.
	An option button (sometimes referred to as a radio button), consists of a round button and a label and represents one of a set of mutually exclusive choices. Only one option button within a group can be selected. Dialog Editor sequentially numbers each option button starting from 0. The returned value is the number assigned to the button selected by the user.
Check box	Use this command to add a check box to the dialog box template.
	A check box consists of a square and a label. If the user checks a check box, an X appears in the square and the returned value is 1. If the user unchecks the check box, the square is empty and the returned value is 0.
Group box	Use this command to add a group box to the dialog box template.
	A group box has a label and encloses related components. For example, each of the Dialog Editor Information dialog boxes has a group box

		labeled Position that contains the X and Y text boxes.
	Text	Use this command to add a text component to the dialog box template.
		A text component can be free-standing text in the dialog box template or a label for a text box, list box, or combo box.
	Text box	Use this command to add a text box to the dialog box template.
		A text box consists of a rectangle in which a user can type a line of text. The returned value is the string that the user enters.
	List box	Use this command to add a list box to the dialog box template.
		A list box displays an array of items. The returned value from a list box is the subscript for the array item that the user selects. A list box created through Dialog Editor lets the user select only one item.
	Combo box	Use this command to add a combo box to the dialog box template.
		A combo box is a combination of a list box and a text box. The user can either type information in the text box or click the prompt button to open a drop-down list and make a selection. The returned value from a combo box is the string that is either entered or selected by the user.
	Droplist box	Use this command to add a drop list box to the dialog box template.
	Picture	Use the Picture command or tool to add a picture to the dialog box template.
	Picture button	Use the Picture Button command or tool to add a picture button to the dialog box template.
Help	Contents	Choose Contents ... from the Help menu to display the Help Contents tab.
	Search for Help on...	Choose Search for Help On ... from the Help menu to display the Help Index tab.

Overview

Macro Editor Toolbar

Overview

Overview

Tool	Menu Item Activated
▶	Start Macro
■	Stop Macro
	Toggle a Breakpoint
	Add Watch
	Call Stack
	Step Into
	Step Over

Overview

Dialog Editor Toolbar

Overview

Overview

Tool	Menu Item Activated	Tool	Menu Item Activated
	Test		Group Box
	Control Info		Text
	Selection Tool		Text Edit Box
	OK Button		List Box
	Cancel Button		Combo Box
	Push Button		Drop List Box
	Option Button		Picture
	Check Box		Picture Button

Selection Tool

Use the Selection tool to select a box or button control on the dialog box and to drag, move, and size items on the dialog box template.

See Also:

Option Base
Dim
ReDim
LBound( )
UBound( )
ArrayDims( )
ArraySort

See Also:

[Exp( )](#)
[Log( )](#)
[Sqr( )](#)

See Also:

[Random( )](#)
[Randomize](#)
[Rnd( ))](#)

See Also:

Fix( ))
Int( )

See Also:

[Atn( )](#)
[Cos( )](#)
[Sin( )](#)
[Tan( )](#)

See Also:

ClipboardClear

See Also:

Chr$
Str$( )
Comment
Oct$( )
CStr( )
DateSerial( )
DateValue( )

See Also:

[Rem](#)
[' Comment](#)

See Also:

[DateSerial( )](#)
[DateValue( )](#)

See Also:

Date$
Now( )
Time$

See Also:

[Day( )](#)
[Month,Weekday( )](#)
[Year( )](#)

See Also:

[Hour( )](#)
[Minute( )](#)
[Second( )](#)

See Also:

Date$
Time$

See Also:

TimeSerial( )
TimeValue( )

See Also:

[ebLeftButton](#)
[ebRightButton](#)

See Also:

See Also:

See Also:

[% Multiplication](#)
[+ Addition](#)
[- Subtraction](#)
[/ Division](#)
[\ integer division](#)
[MOD](#)

See Also:

[< Less than](#)
[<= Less than or equal to](#)
[<> Not equal to](#)
[= Equal to](#)
[> Greater than](#)
[>= (greater than or equal to)](#)

See Also:

[AND](AND)
[NOT](NOT)
[OR](OR)
[XOR](XOR)

See Also:

See Also:

AnswerBox( )
AskBox$( )
InputBox$( )
MsgBox
OpenFileName$( )
PopupMenu( )
SaveFileName$( )
SelectBox( )

See Also:

[Print](#)
[PrinterGetOrientation()](#)
[PrinterSetOrientation](#)
[PrintFile( )](#)

See Also:

[EBHomeDir$( )](#)
[EBOS( )](#)

See Also:

[Function...End Function](#)

See Also:

[Exit Function](#)
[Exit Sub](#)

See Also:

ReadINI$( )
ReadINISection

See Also:

WriteINI

See Also:

LCase$( )
UCase$( )

See Also:

[Space$( )](#)
[String$( )](#)

See Also:

See Also:

[Item$( )](#)
[ItemCount( )](#)
[Word$( )](#)
[WordCount( )](#)

See Also:

Asc( )
Chr$

See Also:

Erl( )
Err
Err( )
Error
Error$( )
On Error Resume

See Also:

[Let](#)
[Const](#)
[Dim](#)
[Deftype](#)

See Also:

[Open](#)
[Close](#)
[Reset](#)

See Also:

[EOF](#)
[ebDOS](#)
[ebWindows](#)
[FileAttr( )](#)
[FileDateTime( )](#)
[FileExists( )](#)
[FileLen( )](#)
[FileList](#)
[FileType( )](#)
[FreeFile( )](#)
[Loc( )](#)
[LOF( )](#)
[Seek](#)

See Also:

ebArchive

ebDirectory

ebHidden

ebNormal

ebReadOnly

ebSystem

ebVolume

FileAttr( )

See Also:

ChDir
ChDrive
CurDir$( )
DiskDrives
DiskFree( )
MkDir
RmDir

See Also:

[Dir$( )](#)
[FileParse$( )](#)
[FileDirs](#)
[Kill](#)
[Name...As](#)

See Also:

[Input #](#)
[Input$( )](#)
[Line Input #](#)

See Also:

ebArchive
ebDirectory
ebHidden
ebNormal
ebReadOnly
ebSystem
ebVolume
GetAttr
SetAttr

See Also:

Print
[Write #](Write)

See Also:

[GoSub...Return](#)
[GoTo](#)
[On Error](#)

See Also:

[End](End)
Stop

See Also:

[Do...Loop](#)
[Exit Do](#)
[Exit For](#)
[For...Next](#)
[While...Wend](#)

See Also:

See Also:

[ebDOS](#)
[ebWindows](#)

See Also:

See Also:

[DoKeys](#)
[SendKeys](#)
[Sending keystrokes to an application](#)

See Also:

[FileList](#)
[GetAttr( )](#)
[SetAttr](#)

See Also:

ebDOS
FileType( )
ebWindows

Symantec Basic Macro Editor

[Macro Editor](#)

[Dialog Editor](#)

[Assigning Macros and Commands to Keys](#)

[Symantec Basic Command, Function, and Language Reference](#)

Scriptmaker

Choose Scriptmaker from the Macro menu to see the Scriptmaker Dialog Box which you use to copy, name, and edit macros.

 ScriptMaker Dialog Box

Use the Scriptmaker dialog box to copy, name, and edit macros.

Macros

The **Macros** list box lists all the macros associated with the current project.   ScriptMaker macro files have the extension .MAC.   The default macro is highlighted by default. Click a macro to select it; double-click to edit it

Macro Menu and File Name

You can rename, duplicate, edit, or delete a named macro.   You cannot rename the default script; you must duplicate it first.   If DEFAULT.MAC is deleted from disk, Symantec Basic creates a stub file when it starts up.

Menu Order

Allows you to change the order of the macros which appear in the Macro menu. Click the up arrow to move the selected macro up in the menu; click the down arrow to move it down in the menu. The default macro always stays at the top of the list.

Display In Menu Check Box

Check Put in Menu if you want the current (highlighted) macro to appear by name at the bottom of the editor's Macro menu.

Edit Button

Click **Edit** to edit the current macro.   This button runs the Symantec BASIC for Windows macro editor. Click the Edit button to open the Macro Editor.

Rename Button

Click **Rename** to rename the current macro.   The Rename/Duplicate Macros dialog box appears.   **Rename** is not active when the default macro is highlighted.

Duplicate Button

Click Duplicate to create a copy of the current macro. The Duplicate/Clone Macros dialog box appears.

Delete Button

Click **Delete** to delete the current macro.   **Delete** is not active when the default macro is highlighted.

Done Button

Click **Done** to save all changes to the project's macros and close the ScriptMaker dialog

Rename/Duplicate Macro Dialog Box

Use this dialog box when you duplicate or rename a macro.

Macro Name

Name under which this macro appears in the Macro menu.

File Name

Name of file where the macro is saved. When duplicating, this filename must be different from any other macro's filename.

Macro Editor

Click the Edit button in the ScriptMaker dialog box to open the Symantec Basic Macro Editor. Use the tools and commands in the Macro Editor to create and edit macros to automate repetitive tasks in your project.

Edit Pane

The edit pane contains the Symantec Basic for the macro you are currently editing.

Status Bar

The status bar displays the current location of the insertion point within your macro.

Watch Pane

The watch pane opens to display the watch variable list after you have added one or more variables to that list.

Creating a New Macro

Creating a New Macro from an Existing One

Editing a Macro

Testing and Debugging a Macro

Using a Dialog Box in a Macro

Exiting the Macro Editor

Macro Editor Menus

Macro Editor Toolbar

Dialog Editor for Symantec Basic Macros

Choose Insert New Dialog or Edit Dialog from the Macro editor's Edit menu to open the Symantec Basic Dialog Editor. Use the tools and commands in the Dialog Editor to create dialog boxes for your Symantec Basic macros.

Creating a New Dialog Box for a Macro
Adding Buttons and Boxes
Editing Buttons and Boxes
Setting Dialog Box Attributes
Incorporating a Dialog Box into a Macro
Testing Your Macro Dialog Box

Dialog Editor Menus

Dialog Editor Toolbar

Assigning Commands to Keys

Click on the Keyboar tab in the Environment Options dialog box. Use this page to create and edit Key Bindings files.

A Key Bindings file (.KEY) associates keystroke sequences with menu/editor commands and user-defined macros. Macros are listed under the name "macroxxxxx" where "xxxxx" is the name you have assigned to the macro.

Also use this page to choose the particular key binding set you wish to use in a session.

Part 2

Macro Command, Function, and Language Reference

This section lists all of Symantec Basic's commands, functions, operators, constants, and statements, along with , syntax and examples.   Also included are the Visual Cafe commands.   These commands are used to control the functionality of Visual Cafe   from a Macro script.

Command Alphabetic Index   Locate a specific Visual Cafe command by name.

Function Alphabetic Index   Locate a specific function by name.

Command Reference Categories   Task-oriented overview of commands.

Function Reference Categories A task-oriented overview of functions and statements.

Language Reference Conventions, procedures, and components of Symantec Basic.

Symantec Basic Function Reference Alphabetic Index

& (operator)
' (comment)
() (keyword)
* (multiplication)
 + (addition)
. (keyword)
+ (concatenation)
- (unary minus)
- (subtraction)
/ (division)
< (less than)
<= (less than or equal to)
<> (not equal to)
= (equal to)
= (statement)
> (greater than)
>= (greater than or equal to)
\ (integer division)
^ (exponentiation)
_ (keyword)

A

Abs( )
AND
AnswerBox( )
Any (data type)
ArrayDims( )
ArraySort
Asc( )
AskBox$( )
AskPassword$( )
Atn( )

B

Basic.Capability

Basic.EoIn$

Basic.FreeMemory

Basic.HomeDir$

Basic.OS

Basic.PathSeparator$

Basic.Version$

Beep

Begin Dialog...End Dialog

Boolean (data type)

ByRef (keyword)

ByVal (keyword)

C

Call

CancelButton

CBool (function)

CCur (function)

CDate, CVDate (functions)

CDbl( )

ChDir

ChDrive

CheckBox

Choose (function)

Chr$( )

CInt( )

Clipboard$( )

Clipboard$

Clipboard.Clear

Clipboard.GetFormat

Clipboard.GetText

Clipboard.SetText

CLng( )

Close

ComboBox

Command$( )

Const

Cos( )

CSng( )

CStr( )

CurDir$( )

Currency (data type)

E

F

L

M

N

O

Symantec Basic Function Reference Categories

Each of the categories listed below contains a set of Symantec Basic functions that are related by task. Click a category to see the list of related functions.

Arrays	Keyboard
Clipboard Manipulation	Math
Conversion	Miscellaneous Comment
Date/time	Operators
Dialogs	Predefined Dialog Boxes
EB Environment	Printer
Environment	Procedures
Error Trapping	Strings
File I/O	Variables and Constants
Flow Control	

Array Functions

Change Default Lower Limit	Option Base
Declare and Initialize	Dim, ReDim
Find the Limits	LBound( ), UBound( )
Manipulate an array	ArrayDims( ), ArraySort

Math Functions

General calculations	Exp( ), Log( ), Sqr( )
Generate random numbers	Random( ), Randomize, Rnd( )
Get absolute value	Abs( )
Get the sign of an expression	Sgn( )
Numeric conversion	Fix( )), Int( )
Trigonometry	Atn( ), Cos( ), Sin( ), Tan( )

Clipboard Manipulation Functions

Clear the clipboard object

Clipboard.Clear

Conversion Functions

ANSI value to string	Chr$
Number to string	Str$( ), ' Comment, Oct$( ), CStr( )
Date to serial number	DateSerial( ), DateValue( )

Miscellaneous Comment

Comment	Rem, ' Comment
Get command-line arguments	Command$( )
Process pending events to occur before continue	DoEvents
Run another program	Shell( )
Sound a beep	BeepBeep

Date/time Functions

Date to serial number	DateSerial( ), DateValue( )
Get the current date or time	Date$, Now( ), Time$
Serial number to date	Day( ), Month,Weekday( ), Year( )
Serial number to time	Hour( ), Minute( ), Second( )
Set the date or time	Date$, Time$
Time a process	Timer( )
Time to serial number	TimeSerial( ), TimeValue( )

Operators

Arithmetic	* Multiplication
	+ Addition
	- Subtraction
	,/ Division
	\ integer division
	,^ Exponentiation
	MOD
Comparison	< Less than
	<= Less than or equal to
	<> Not equal to
	= Equal to
	> Greater than
	>= (greater than or equal to)
Logical	AND
	NOT
	OR
	XOR

Dialog Functions

Create user-defined dialogs	Begin Dialog...End Dialog CancelButton Checkbox Dialog Groupbox ListBox, OKButton OptionButton OptionGroup PushButton Text, TextBox
Display dialog boxes	AnswerBox( ), AskBox$( ), InputBox$( ), MsgBox, OpenFileName$( ), PopupMenu( ), SaveFileName$( ), SelectBox( )

Printer Functions

Manipulate the printer	Print
	PrinterGetOrientation( )
	PrinterSetOrientation
	PrintFile( )

EB Environment

| Get EB environment information | Basic.HomeDir$, Basic.OS, Basic.Version$ |

Procedures

Call a subroutine	Call
Declare a reference to an external routine	Declare
Define a procedure	Function...End Function
Exit from a procedure	Exit Function, Exit Sub

Environment Functions

Find environment variables	Environ$( )
Get information about the system	ReadINI$( ) ReadINISection
Modify the windows environment	WriteINI

String Functions

Convert to lowercase or uppercase letters	LCase$( ), UCase$( )
Create strings of repeating characters	Space$( ), String$( )
Find the length of a string	Len( )
Manipulate strings	Instr, Left$( ), LTrim$( ), Mid$, Right$( ), RTrim$( ), Str$( )
Parsing	Item$( ), ItemCount( ), Word$( ), WordCount( )
Work with the ASCII and ANSI values	Asc( ), Chr$

Error Trapping

Get error messages	Error$( )
Get error-status data	Erl( )
Get or set error-status data	Err
Simulate run-time errors	Error
Trap errors while a program is running	On Error Resume

Variables and Constants

Assign value	Let
Declare variables or constants	Const, Dim
Set default data type	Deftype

File I/O

Access or create a file	Open
Close files	Close, Reset
Get information about a file	EOF, FileAttr( ), FileDateTime( ), FileExists( ), FileLen( ), FileList, FileType( ), FreeFile( ), Loc( ), LOF( ), Seek
Manage disk drives or directories	ChDir, ChDrive, CurDir$( ), DiskDrives, DiskFree( ), MkDir, RmDir
Manage files	Dir$( ), FileParse$( ), FileDirs, Kill, Name...As
Read from a file	Input #, Input$( ), Line Input #

Set or get file attributes	GetAttr, SetAttr
Set read-write position in a file	Seek
Write to a file	Print, Write #

Flow Control

Branch	GoSub...Return, GoTo, On Error
Exit or pause the program	End, Stop
Loop	Do...Loop, Exit Do, Exit For, For...Next, While...Wend
Make decisions	If...End If, Select Case
Pause script	Sleep
Prevent applications of other applications	Exclusive

Keyboard Functions

Manipulate the keyboard	DoKeys SendKeys

& Operator

Description: Returns the concatenation of *expression1* and *expression2*.

Syntax: *expression1* & *expression2*

Comments: If both expressions are strings, then the type of the result is String. Otherwise, the type of the result is a String variant.

When nonstring expressions are encountered, each expression is converted to a String variant. If both expressions are Null, then a Null variant is returned. If only one expression is Null, then it is treated as a zero-length string. Empty variants are also treated as zero-length strings.

In many instances, the plus (+) operator can be used in place of &. The difference is that + attempts addition when used with at least one numeric expression, whereas & always concatenates.

Example: This example assigns a concatenated string to variable s$ and a string to s2$, then concatenates the two variables and displays the result in a dialog box.

```
Sub Main()
    s$ = "This string" & " is concatenated"
    s2$ = " with the & operator."
    MsgBox s$ & s2$
End Sub
```

See Also

[+ (addition)](#)

' (comment) Command

Description: The compiler ignores comments in a script file. A comment consists of one of the following:

- All the characters between a single quotation mark and the end of the line.

- A line that starts with Rem followed by a space.

- All the characters between /* and */, even if they appear on more than one line. (The */ can be followed only by spaces and the carriage return.)

Syntax: '
Rem
/* ... */

Example: The following are examples of the syntax for comments:

```
MsgBox "Hello, world!" 'Positive message
REM This script performs...
MsgBox "Hello, world!" /* This displays a string
inside a message box. The script pauses until the user clicks the OK
button. */
```

See Also

[REM](#)

()   **Keyword**

Syntax 1:	...(expression)...
Syntax 2:	...,(parameter),...
Description:	Forces parts of an expression to be evaluated before others or forces a parameter to be passed by value.
Comments:	**Parentheses within Expressions**

Parentheses override the normal precedence order of BasicScript operators, forcing a subexpression to be evaluated before other parts of the expression. For example, the use of parentheses in the following expressions causes different results:

```
i = 1 + 2 * 3          'Assigns 7.
i = (1 + 2) * 3 'Assigns 9.
```

Use of parentheses can make your code easier to read, removing any ambiguity in complicated expressions.

Parentheses Used in Parameter Passing

Parentheses can also be used when passing parameters to functions or subroutines to force a given parameter to be passed by value, as shown below:

```
ShowForm i              'Pass i by reference.
ShowForm (i)            'Pass i by value.
```

Enclosing parameters within parentheses can be misleading. For example, the following statement appears to be calling a function called ShowForm without assigning the result:

```
ShowForm(i)
```

The above statement actually calls a subroutine called ShowForm, passing it the variable i by value. It may be clearer to use the ByVal keyword in this case, which accomplishes the same thing:

```
ShowForm ByVal i
```

Note: The result of an expression is always passed by value.

Example:

```
'This example uses parentheses to clarify an expression.

Sub Main()
    bill = False
    dave = True
    jim = True
```

```
        If (dave And bill) Or (jim And bill) Then
           Msgbox "The required parties for the meeting are here."
        Else
           MsgBox "Someone is late for the meeting!"
        End If
     End Sub
```

Platform(s):   All.

See Also

ByVal (keyword)

* (multiplication)   **Numeric Operator**

Description: The multiplication operator (*) indicates that two numbers are to be multiplied. The result is the product of the two.

Syntax: operand1 * operand2

    operand1   A numeric expression for the first factor.

    operand2   A numeric expression for the second factor.

Example: To assign start the product of 4 and 5, and end the product of 100 and start:

```
start% = 4 * 5
end% = 100 * start
```

+ (addition)   **Numeric Operator**

Description:	The addition operator (+) indicates that two numbers are to be added. The result is the sum of the two.
Syntax:	operand1 + operand2
	operand1, The numeric expressions for the addends.
	operand2
Example:	In the following example, z stores the sum of x and y.

```
x& = 45113
y% = 25
z& = x + y
```

See Also

[&(operator)](#)

+ (concatenation)　　　String Operator

Description: The concatenation operator (+) indicates that two string expressions are to be joined into one string expression. The resulting string starts with operand1 and ends with operand2.

Syntax: operand1 + operand2

operand1,　　　The string expressions to be concatenated.

operand2

Example: In the following example, String3 becomes the concatenation of String1 and String2: "Good Morning, how are you?"

```
String1$ = "Good Morning"'
String2$ = ", how are you?"
String3$ = String1 + String2
```

See Also

[& (operator)](#)

- (unary minus)   **Numeric Operator**

Description: The unary minus operator (-) indicates that the sign of the specified number is to be changed. If the number is negative, it becomes positive. If it is positive, it becomes negative.

Syntax: -operand

operand A numeric expression to change the sign of.

Example: The following example takes the unary minus of 32 and assigns it to the variable n.

```
n% = -32
```

.   **Keyword**

Syntax 1:   object.property

Syntax 2:   structure.member

Description: Separates an object from a property or a structure from a structure member.

Examples: This example uses the period to separate an object from a property.

```
Sub Main()
    MsgBox "The clipboard text is: " & Clipboard.GetText()
End Sub

    'This example uses the period to separate a structure from a member.

Type Rect
    left As Integer
    top As Integer
    right As Integer
    bottom As Integer
End Type

Sub Main()
    Dim r As Rect
    r.left = 10
    r.right = 12
    Msgbox "r.left = " & r.left & ", r.right = " & r.right
End Sub
```

Platform(s):   All.

- (subtraction)   **Numeric Operator**

Description:	The subtraction operator (-) indicates that one number is to be subtracted from another. The result is the difference between the two.
Syntax:	operand1 - operand2
	operand1 A numeric expression for the minuend.
	operand2 A numeric expression to subtract for the subtrahend.
Example:	The following example subtracts n from -32.

```
s% = -32 - n
```

/ (division)   **Numeric Operator**

Description: The division operator (/) indicates that one number is to be divided by another. The result is the quotient of the two.

Syntax: operand1 / operand2

operand1   A numeric expression for the dividend.

operand2   A numeric expression for the divisor.

Example: In the following example, the value of z becomes 4.

```
z = 12/3
```

See Also

[\ (Integer division)](#)

< (less than)   Relational Operator

Description: The < relational operator stands for "less than." The result is true if the first expression is less than the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax: expr1 < expr2

expr1, expr2

The numeric or string expressions being compared.

Examples: The following examples show comparisons of either numeric or string expressions and their results.

```
1 < 2
'TRUE because 1 is less than 2

"alpha" < "beta
'TRUE because a is less than b in ASCII

"a " < "a"
'FALSE because the longer string is greater
'than the shorter
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

<= (less than or equal to)　　Relational Operator

Description:　The <= relational operator stands for "less than or equal to." The result is true if the first expression is less than or equal to the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax:　expr1 <= expr2

expr1, expr2

The numeric or string expressions being compared.

Examples:　The following examples show comparisons of either numeric or string expressions and their results.

```
1 <= 2
'TRUE because 1 is less than 2

"alpha" <= "beta"
'TRUE because a is less than b in ASCII
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

<> (not equal to)　　　**Relational Operator**

Description:　The <> relational operator stands for "not equal to." The result is true if the first expression is not equal to the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax:　expr1 <> expr2

　　　expr1, expr2

　　　The numeric or string expressions being compared.

Examples:　The following examples show comparisons of either numeric or string expressions and their results.

```
"alpha" <> "beta"
'TRUE because a is less than b in ASCII

123 <> 123
'FALSE because they are equal
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

=   **Statement**

Syntax: variable = expression

Description: Assigns the result of an expression to a variable.

Comments: When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This occurs when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```
Dim amount As Long
Dim quantity As Integer

amount = 400123 'Assign a value out of range for int.
quantity = amount      'Attempt to assign to Integer.
```

When performing an automatic data conversion, underflow is not an error.

The assignment operator (=) cannot be used to assign objects. Use the Set statement instead.

Example:

```
Sub Main()
    a$ = "This is a string"
    b% = 100
    c# = 1213.3443
    MsgBox a$ & "," & b% & "," & c#
End Sub
```

Platform(s):   All.

See Also

[Let](#)
[Set](#)

= (equal to)   **Relational Operator**

Description: The = relational operator stands for "equal to." The result is true if the first expression is equal to the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax: expr1 = expr2

expr1, expr2

The numeric or string expressions being compared.

Examples: The following examples show comparisons of either numeric or string expressions and their results.

```
"alpha" = "beta"
'FALSE because a is less than b in ASCII

123 = 123
'TRUE because they are equal
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

> (greater than)   **Relational Operator**

Description: The > relational operator stands for "greater than." The result is true if the first expression is greater than the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax: expr1 > expr2

expr1, expr2

The numeric or string expressions being compared.

Examples: The following examples show comparisons of either numeric or string expressions and their results.

```
1 > 2
'FALSE because 1 is less than 2

"alpha" > "beta"
'FALSE because a is less than b in ASCII
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

>= (greater than or equal to)   Relational Operator

Description: The >= relational operator stands for "greater than or equal to." The result is true if the first expression is greater than or equal to the second expression. Otherwise, the result is false. The comparison can be performed between two numbers or between two strings, but not between a number and a string.

Syntax: expr1 >= expr2

expr1, expr2

The numeric or string expressions being compared.

Examples: The following examples show comparisons of either numeric or string expressions and their results.

```
"alpha" >= "beta"
'FALSE because a is less than b in ASCII

"a " >= "a"
'TRUE because the longer string is greater than
'the shorter
```

See Also

[Is](#)
[Like](#)
[Option Compare](#)

\ (integer division)   Numeric Operator

Description: The integer division operator (\\) indicates that the integer division of one number by another number is to be performed. Each operand is rounded to an integer prior to the division. The result is the integer part of the unrounded quotient.

Syntax: operand1 \\ operand2

operand1   A numeric expression for dividend.

operand2   A numeric expression for divisor.

Examples: The following are examples of integer division.

```
z = 3\1.6
'Equivalent to 3\2. The result is 1.
z = 3\1.5
'Also equivalent to 3\2. The result is 1.
z = 3\1.4
'Equivalent to 3\1. The result is 3.
```

See Also

[/ (division)](/)

^ (exponentiation)　　　　**Numeric Operator**

Description:　The exponentiation operator (^) indicates that a specified base number is to be raised to a specified power.

Syntax:　base ^ exponent

　base　　　　A numeric expression for the base number.

　exponent　　A numeric expression for the power or exponent.

Example:　The following example finds 2 cubed.

```
result% = 2^3
```

The next example also finds 2 cubed. Parentheses are used because exponentiation has a higher precedence than addition.

```
result = (1+1)^(1+2)
```

_   **Keyword**

Description: Line-continuation character, which allows you to split a single Symantec Basic statement onto more than one line.

Syntax: s$ = "This is a very long line that I want to split " + _
"onto two lines"

Comments: The line-continuation character cannot be used within strings and must be preceded by white space (either a space or a tab).

The line-continuation character can be followed by a comment, as shown below:

```
i = 5 + 6 & _            'Continue on the next line.
    "Hello"
```

Example: Const crlf = Chr$(13) + Chr$(10)

```
Sub Main()
    'The line-continuation operator is useful when concatenating
    'long strings.

    Msg$ = "This line is a line of text that" + crlf + "extends beyond "
_
            + "the borders of the editor" + crlf + "so it is split into
" _
            + "multiple lines"

    'It is also useful for separating and continuing long calculation
lines.

    b# = .124
    a# = .223
    S# = ( (((Sin(b) ^ 2) + (Cos(a) ^ 2)) ^ .5) / _
        (((Sin(a) ^ 2) + (Cos(b) ^ 2)) ^ .5) ) * 2.00
    MsgBox Msg + crlf + "The value of S is: " + Str$(S)
End Sub
```

Abs( )   **Function**

Description:	This function returns a number that is the absolute value of the numeric expression. An absolute value is always positive.
Syntax:	Abs(exprN)
Parameter:	exprN
	A numeric expression.
Example:	The following example determines the absolute value of a numeric expression.

```
absoluteValue = Abs(x+y+z)
```

See Also

[Sgn()](Sgn())

AND Logical Operator

Description: This logical operator usually joins two logical or relational expressions into another logical expression. The result is true if both expressions are true; otherwise the result is false.

If the expressions are numeric (as in the FileList statement), the result is a bitwise AND of the two numbers. If either of the expressions is a floating-point number, the expressions are converted to longs before the bitwise AND.

Syntax: expr1 AND expr2

expr1, expr2

Numeric, relational, or logical expressions.

Examples: The following example determines whether a specified number is between 1 and 10, inclusive.

```
If theNumber >= 1 AND theNumber <= 10 Then
    validNumber = TRUE
End If
```

See Also

[Or](#)
[XOR](#)
[Eqv (operator)](#)
[Imp](#)

AnswerBox( )   Function

Description: This function displays a dialog box containing a message and a maximum of three command buttons. It returns the integer indicating which button was selected: 1, 2, or 3. If the user cancels the answer box by double-clicking the close box or pressing the Esc key, the function returns 0.

The width and height of the dialog box are sized to hold the entire contents of the message that is in 8-point Helvetica font. The maximum size of the dialog box is 5/8 of the width and 3/4 of the height of the screen. If a line is too long, it wraps from one line to the next. The widest button label determines the width of the buttons.

Syntax: AnswerBox(message [, button [, button [, button]]])

Parameters: message

A string expression for the user to respond to. The message can contain Chr$(13)+Chr$(10) (carriage return/linefeed) to separate lines.

button

A string expression containing the label of a button. If no buttons are specified, the default labels are OK and Cancel which have the values 1 and 2, respectively.

Example: In the following example, AnswerBox( ) displays a message and three buttons.

```
...
message = "What do you want to do with this record?"
buttonChoice = AnswerBox(message, "Add", "Modify", "Delete")
```

See Also

[MsgBox](#)
[AskBox$()](#)
[AskPassword$()](#)
[InputBox$()](#)
[OpenFileName$()](#)
[SaveFileName$()](#)
[SelectBox()](#)

Any   **Data type**

Description: Used with the Declare statement to indicate that type checking is not to be performed with a given argument.

Comments: Given the following declaration:

```
Declare Sub Foo Lib "FOO.DLL" (a As Any)
```

the following calls are valid:

```
Foo 10
Foo "Hello, world."
```

Example: The following example calls FindWindow to determine if the Program Manager is running. This example will only run under Windows and Win32 platforms. This example uses the Any keyword to pass a NULL pointer, which is accepted by the FindWindow function.

```
Declare Function FindWindow16 Lib "user" Alias "FindWindow" (ByVal Class _
    As Any,ByVal Title As Any) As Integer

Declare Function FindWindow32 Lib "user32" Alias "FindWindowA" (ByVal Class _
    As Any,ByVal Title As Any) As Long

Sub Main()
    Dim hWnd As Variant

    If Basic.Os = ebWin16 Then
       hWnd = FindWindow16("PROGMAN",0&)
    ElseIf Basic.Os = ebWin32 Then
       hWnd = FindWindow32("PROGMAN",0&)
    Else
       hWnd = 0
    End If

    If hWnd <> 0 Then
       MsgBox "Program manager is running, window handle is " & hWnd
    End If
End Sub
```

Platform(s):   All.

See Also

[Declare](#)

ArrayDims( )   Function

Description: This function returns an integer (from 0 to 60) representing the number of dimensions in an array. A return value of 0 means that the array has no dimensions and is, therefore, empty.

Syntax: ArrayDims(arrayName)

Parameters: arrayName

Array variable whose dimensions you want to determine.

Example: In the following example, the ArrayDims( ) function checks an array for emptiness. This function determines emptiness in this case because the FileList statement redimensions the array.

```
'allocate empty array
Dim files$(1 to 10)
'The FileList statement searches for files with
'the specified extension and redimensions the array

'fill the array
FileList files, "C:\*.BAT"
'If the array has no dimensions,
'no files were found
If ArrayDims(files$) = 0 Then
    Exit Sub 'exit if no elements
End If
```

See Also

[LBound()](#)
[UBound()](#)

ArraySort

Statement

Description:	This statement sorts a one-dimensional array in ascending order. If a string array is specified, then the routine sorts alphabetically (using case-sensitive string comparisons). A run-time error results if the number of dimensions is more than one.
Syntax:	ArraySort arrayName
Parameter:	arrayName
	One-dimensional array variable.
Example:	The following example sorts an array of any type.

```
'dayArray is a one-dimensional array
ArraySort dayArray
```

See Also

[ArrayDims()](#)
[LBound()](#)
[UBound()](#)

Asc( )　　　　　　　　**Function**

Description:　This function returns an integer between 0 and 255 corresponding to the ANSI code for the first character of the text string.

Syntax:　Asc(char)

Parameter:　char

　A string expression whose initial character is evaluated.

Example:　To determine the ASCII value for the letter "A," you could use the following:

```
asciiA% = Asc("A")
```

See Also

[Chr$()](Chr$())

AskBox$( )　　　　　　　Function

Description:　This function displays a dialog box with a message, a text box, and OK and Cancel command buttons. The name of the dialog box is always "BasicScript." The dialog box is sized to the width of the message, using 8-point Helvetica font. The text box is active. The function returns the string in the text box, or an empty string if the user cancels the dialog box.

Syntax:　AskBox$(message [,contents])

Parameters:　message

A string expression for the user to respond to.

contents

A string expression (with a maximum of 255 characters) used as the initial contents of the text box. The user can accept this or type in a new string. The default is the empty string.

Examples:　The following example displays an empty text box.

```
Filename = AskBox$("File Name:")
```

The next example displays a default filename in the text box.

```
Filename = AskBox$("File Name:", "FOO.TXT")
```

See Also

[MsgBox](#)
[AskPassword$()](#)
[InputBox$()](#)
[OpenFileName$()](#)
[SaveFileName$()](#)
[SelectBox()](#)

AskPassword$( )   Function

Description: This function displays a dialog box, a message, a password box, and OK and Cancel command buttons. The dialog box's name is always "BasicScript." A password box is a text box that displays an asterisk for every character the user types. The dialog box is sized to the width of the message using 8-point Helvetica font. The password box is active. This function returns the string (up to 255 characters) that the user types, or an empty string if the user cancels the dialog box.

Syntax: AskPassword$(message)

Parameter: message

A string expression requesting a password or other sensitive information.

Example: The following example asks the user for a password.

```
s$ = AskPassword$("Enter Password:")
```

See Also

[MsgBox](#)
[AskBox$()](#)
[InputBox$()](#)
[OpenFileName$()](#)
[SaveFileName$()](#)
[SelectBox()](#)
[AnswerBox()](#)

Atn( )   **Function**

Description:	This function returns the arctangent of the specified number. The value returned is a number of type double.
Syntax:	Atn(exprN)
Parameters:	exprN
	A numeric expression.
Example:	The arctangent of a number is equivalent to the inverse tangent as the following example shows.

```
'Find the tangent of PI/2
tanPI_2 = Tan(PI/2)
'Find the arctangent of the tangent of PI/2
angle = Atn(tanPI_2)'angle is PI/2
```

See Also

[Tan()](#)
[Sin()](#)
[Cos()](#)

Basic.Capability   Method

Description: Returns True if the specified capability exists on the current platform; returns False otherwise.

Comments: The *which* parameter is an Integer specifying the capability for which to test. It can be any of the following values:

Value	Returns **True** If the Platform Supports
1	Disk drives
2	System file attribute (ebSystem)
3	Hidden file attribute (ebHidden)
4	Volume label file attribute (ebVolume)
5	Archive file attribute (ebArchive)
6	Denormalized floating-point math

Example: This example tests to see whether your current platform supports disk drives and hidden file attributes and displays the result.

```
Sub Main()
   Msg$ = "This OS "

   If Basic.Capability(1) Then
      Msg = Msg + "supports disk drives."
   Else
      Msg = Msg + "does not support disk drives."
   End If

   MsgBox Msg
End Sub
```

See Also

Basic.OS

Basic.HomeDir$ Property

Description: This property returns a string containing the directory name for Symantec Basic.

Syntax: Basic.HomeDir$

Example: The following example stores Symantec Basic's home directory in the string variable homeDir.

```
homeDir$ = Basic.HomeDir$
```

Basic.Eoln$ **Property**

Description: Returns a String containing the end-of-line character sequence appropriate to the current platform.

Syntax: Basic.Eoln$

Comments: This string will be either a carriage return, a carriage return/line feed, or a line feed.

Example: This example writes two lines of text in a message box.

```
Sub Main()
    MsgBox "This is the first line of text " + Basic.Eoln$ + "The second
line"
End Sub
```

See Also

[Basic.PathSeparator$](#)

Basic.FreeMemory   Property

Description: Returns a Long representing the number of bytes of free memory in Symantec Basic's data space.

Syntax: Basic.FreeMemory

Comments: This function returns the size of the largest free block in Symantec Basic's data space. Before this number is returned, the data space is compacted, consolidating free space into a single contiguous free block.

Symantec Basic's data space contains strings and dynamic arrays.

Example: This example displays free memory in a dialog box.

```
Sub Main()
    MsgBox "The free memory space is: " + Str$(Basic.FreeMemory)
End Sub
```

Basic.OS   Property

Description: This property returns a numeric expression representing the operating environment for Symantec Basic: 0 for Windows, 1 for DOS, or 2 for Chicago, NT, or Win32s.

Syntax: Basic.OS

Example: The following example stores a 0 in the variable opSys for the Windows operating system.

```
opSys% = Basic.OS
```

Basic.PathSeparator$ Property

Description: Returns a String containing the path separator appropriate for the current platform.

Syntax: Basic.PathSeparator$

Comments: The returned string is any one of the following characters: / (slash), \ (back slash), : (colon)

Example:
```
Sub Main()
    MsgBox "The path separator for this platform is: " +
Basic.PathSeparator$
End Sub
```

See Also

Basic.Eoln$

Basic.Version$ **Property**

Description: Returns a String containing the version of Symantec Basic.

Syntax: Basic.Version$

Comments: This function returns the major and minor version numbers in the format *major.minor.BuildNumber*, as in "2.00.30."

Example: This example displays the current version of Symantec Basic.

```
Sub Main()
    MsgBox "The current version is: " + Basic.Version$
End Sub
```

Beep   **Statement**

Description: This statement sounds a single tone through the computer's speaker.

Syntax: Beep

Example: The following loop causes the system to generate 10 beeps in rapid succession.

```
For i = 1 To 10
    Beep
Next i
```

Begin Dialog...End Dialog   Construct

Description:	This construct declares and defines a dialog box template created in the Dialog Editor. Declarations of controls go between the Begin Dialog and End Dialog statements.
Syntax:	Begin Dialog dialogName, x, y, width, height
	[, name]
	End Dialog
Parameters:	dialogName
	The name of the dialog box template that is being defined.
	x, y
	The integers indicating the horizontal and vertical distances from the upper-left corner of the window to the upper-left corner of the dialog box in dialog units.
	width, height
	The integers indicating the width and height of the dialog box in dialog units.
	name
	A string variable or literal which specifies the name of the dialog box. The default is "Untitled."
Example:	The following example defines a dialog template named locateDialog. It has a text component that displays a message for the user and an OK button. The Dim statement declares myDlg as an instance of the template. Then the Dialog( ) function displays that instance.

```
Begin Dialog locateDialog 10,10,100,100, "Text Box and Button"
    Text 40,14,48,8 "Do you want to continue?"
    OkButton 64,50,45,14
End Dialog
'Declare an instance of the dialog box
Dim myDlg As locateDialog
'Display the dialog box with the Dialog function
i = Dialog(myDlg)
```

See Also

[CancelButton](#)
[CheckBox](#)
[ComboBox](#)
[Dialog()](#)
[Dialog](#)
[DropListBox](#)
[GroupBox](#)
[ListBox](#)
[OKButton](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PushButton](#)
[Text](#)
[TextBox](#)
[DlgProc](#)

Boolean   Data type

Syntax: Boolean

Description: A data type capable of representing the logical values True and False.

Comments: Boolean variables are used to hold a binary value, either True or False. Variables can be declared as Boolean using the Dim, Public, or Private statement.

Variants can hold Boolean values when assigned the results of comparisons or the constants True or False.

Internally, a Boolean variable is a 2-byte value holding –1 (for True) or 0 (for False).

Any type of data can be assigned to Boolean variables. When assigning, non-0 values are converted to True, and 0 values are converted to False.

When appearing as a structure member, Boolean members require 2 bytes of storage.

When used within binary or random files, 2 bytes of storage are required.

When passed to external routines, Boolean values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.

There is no type-declaration character for Boolean variables.

Boolean variables that have not yet been assigned are given an initial value of False.

Platform(s):   All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Deftype](#)
[CBool (function)](#)
[TRUE](#)
[False](#)

ByRef   **Keyword**

Syntax:	...,ByRef parameter,...
Description:	Used within the Sub...End Sub, Function...End Function, or Declare statement to specify that a given parameter can be modified by the called routine.
Comments:	Passing a parameter by reference means that the caller can modify that variable's value.
	Unlike the ByVal keyword, the ByRef keyword cannot be used when passing a parameter. The absence of the ByVal keyword is sufficient to force a parameter to be passed by reference:

```
MySub ByVal i    'Pass i by value.
MySub ByRef i    'Illegal (will not compile).
MySub i          'Pass i by reference.
```

Example:

```
Sub Test(ByRef a As Variant)
    a = 14
End Sub

Sub Main()
    b = 12
    Test b
    MsgBox "The ByRef value is: " & b    'Displays 14.
End Sub
```

Platform(s):   All.

See Also

[() (keyword)](#)
[ByVal (keyword)](#)

ByVal   **Keyword**

Syntax: ...ByVal parameter...

Description: Forces a parameter to be passed by value rather than by reference.

Comments: The ByVal keyword can appear before any parameter passed to any function, statement, or method to force that parameter to be passed by value. Passing a parameter by value means that the caller cannot modify that variable's value.

Enclosing a variable within parentheses has the same effect as the ByVal keyword:

```
Foo ByVal i         'Forces i to be passed by value.
Foo(i)              'Forces i to be passed by value.
```

When calling external statements and functions (i.e., routines defined using the Declare statement), the ByVal keyword forces the parameter to be passed by value regardless of the declaration of that parameter in the Declare statement. The following example shows the effect of the ByVal keyword used to passed an Integer to an external routine:

```
Declare Sub Foo Lib "MyLib" (ByRef i As Integer)

i% = 6
Foo ByVal i%              'Pass a 2-byte Integer.
Foo i%                   'Pass a 4-byte pointer to an Integer.
```

Since the Foo routine expects to receive a pointer to an Integer, the first call to Foo will have unpredictable results.

Example:

```
'This example demonstrates the use of the ByVal keyword.

Sub Foo(a As Integer)
    a = a + 1
End Sub

Sub Main()
    Dim i As Integer
    i = 10
    Foo i
    MsgBox "The ByVal value is: " & i          'Displays 11 (Foo changed
the value).
    Foo ByVal i
    MsgBox "The ByVal value is still: " & i          'Displays 11 (Foo
did not change the value).
End Sub
```

Platform(s): All.

See Also

[() (keyword)](#)
[ByRef (keyword)](#)

Call   **Statement**

Description: This statement makes a subroutine call. It transfers control and passes parameters to the specified subroutine. The reserved word Call is optional. The parentheses around the list of parameters are used only when the word Call is used.

Syntax: [Call] subName [([parameterList])]

Parameters: subName

The name of the subroutine being called.

parameterList

List of parameters for the subroutine separated by commas. The syntax is:

parameter [, parameter]...

and the syntax for each parameter is:

[( ] parameterName [ )] | expr

Putting the parameter name in parentheses forces it to be passed by value.

Example: Both of these examples call a subroutine that has three parameters.

```
The parameter hours is being passed by value.
Call Task1 (day, (hours), user)
Task1 day, (hours), user
    Both of the following examples call a subroutine that has no
parameters.
Task2

Call Task2 ( )
```

See Also

[Goto](#)
[GoSub](#)
[Declare](#)

CancelButton   Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a Cancel button for a dialog box template.

Syntax: CancelButton x, y, width, height

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the Cancel button in dialog units.

width, height

The integers indicating the width and height of the Cancel button in dialog units.

Example: The following example displays an instance of a dialog template with OK and Cancel command buttons. Selecting either button causes the dialog function that displays the template to end. If OK is selected, the Dialog( ) function returns TRUE. If Cancel is selected, the function returns FALSE. The result is displayed in a message box.

```
'Define the dialog box template
Begin Dialog userDialog 15, 28, 100, 100, "OK and Cancel"
    Text 40,14,48,8, "Do you want to continue?"
    OKButton 55, 64, 41, 14
    CancelButton 55, 82, 41, 14
End Dialog
'Declare the name of the instance of
'the template
Dim OKCancelDialog As userDialog
'Display the instance of the template
result = Dialog(OKCancelDialog)
'What was the result?
If result = TRUE Then
    MsgBox "OK"
Else
    MsgBox "Cancel"
End If
```

See Also

[CheckBox](#)
[ComboBox](#)
[Dialog()](#)
[Dialog](#)
[DropListBox](#)
[GroupBox](#)
[ListBox](#)
[OKButton](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PushButton](#)
[Text](#)
[TextBox](#)
[Begin Dialog . . .End Dialog](#)
[PictureButton](#)

CBool   **Function**

Syntax:	CBool(expression)
Description:	Converts expression to True or False, returning a Boolean value.
Comments:	The expression parameter is any expression that can be converted to a Boolean. A runtime error is generated if expression is Null.
	All numeric data types are convertible to Boolean. If expression is zero, then the CBool returns False; otherwise, CBool returns True. Empty is treated as False.
	If expression is a String, then CBool first attempts to convert it to a number, then converts the number to a Boolean. A runtime error is generated if expression cannot be converted to a number.
	A runtime error is generated if expression cannot be converted to a Boolean.
Example:	

```
'This example uses CBool to determine whether a string is numeric
'or just plain text.

Sub Main()
    Dim IsNumericOrDate As Boolean
    s$ = "34224.54"
    IsNumeric = CBool(IsNumeric(s$))
    If IsNumeric = True Then
       MsgBox s$ & " is either a valid number!"
    Else
       MsgBox s$ & " is not a valid number!"
    End If
End Sub
```

Platform(s):	All.

See Also

[CCur (function)](#)
[CDate, CVDate (functions)](#)
[CDbl( )](#)
[CInt( )](#)
[CLng( )](#)
[CSng( )](#)
[CStr( )](#)
[CVar (function)](#)
[CVErr (function)](#)
[Boolean (data type)](#)

CCur   **Function**

Syntax:	CCur(expression)
Description:	Converts any expression to a Currency.
Comments:	This function accepts any expression convertible to a Currency, including strings. A runtime error is generated if expression is Null or a String not convertible to a number. Empty is treated as 0.
	When passed a numeric expression, this function has the same effect as assigning the numeric expression number to a Currency.
	When used with variants, this function guarantees that the variant will be assigned a Currency (VarType 6).

Example:

```
'This example displays the value of a String converted into a Currency
value.

Sub Main()
    i$ = "100.44"
    MsgBox "The currency value is: " & CCur(i$)
End Sub
```

Platform(s):	All.

See Also

[CBool (function)](#)
[CDate, CVDate (functions)](#)
[CDbl( )](#)
[CInt( )](#)
[CLng( )](#)
[CSng( )](#)
[CStr( )](#)
[CVar (function)](#)
[CVErr (function)](#)
[Currency (data type)](#)

CDate, CVDate Functions

Syntax: CDate(expression)

CVDate(expression)

Description: Converts expression to a date, returning a Date value.

Comments: The expression parameter is any expression that can be converted to a Date. A runtime error is generated if expression is Null.

If expression is a String, an attempt is made to convert it to a Date using the current country settings. If expression does not represent a valid date, then an attempt is made to convert expression to a number. A runtime error is generated if expression cannot be represented as a date.

These functions are sensitive to the date and time formats of your computer.

The CDate and CVDate functions are identical.

Example:

```
'This example takes two dates and computes the difference between them.

Sub Main()
    Dim date1 As Date
    Dim date2 As Date
    Dim diff As Date

    date1 = CDate(#1/1/1994#)
    date2 = CDate("February 1, 1994")
    diff = DateDiff("d",date1,date2)

    MsgBox "The date difference is " & CInt(diff) & " days."
End Sub
```

Platform(s): All.

See Also

[CCur (function)](#)
[CBool (function)](#)
[CDbl( )](#)
[CInt( )](#)
[CLng( )](#)
[CSng( )](#)
[CStr( )](#)
[CVar (function)](#)
[CVErr (function)](#)
[Date (data type)](#)

CDbl( )   **Function**

Description: This function converts the specified numeric expression to a double-precision number and returns
that number. A run-time error occurs if the specified expression is not within the correct range. This
function is equivalent to assigning the numeric expression to a variable of type double.

Syntax: CDbl(exprN)

Parameter: exprN

A numeric expression within the range for numbers of type double: approximately +/-1.7E+/-308.

Example: The following two assignments are equivalent.

```
x# = Cdbl(4)   'Explicit conversion
x# = 4         'Implicit conversion
```

See Also

CCur (function)
CurDir$( )
Dir$( )
MkDir
RmDir

ChDir   **Statement**

Description: This statement changes the directory on the current drive.

Syntax: ChDir newDir

Parameter: newDir

                   A string expression containing the complete or relative pathname to a directory.

Examples: Assuming that the current drive is C and the directory \LEVEL1\SUB1 exists on the D drive, the
                   following statement makes that directory the current directory on the D drive. The C drive remains
                   the current drive.

```
ChDir "D:\LEVEL1\SUB1"
```

                   To change to the directory one level above the current directory on the current drive, you could use
                   the following:

```
ChDir ".."
```

See Also

[ChDrive](#)
[CurDir$( )](#)
[Dir$( )](#)
[MkDir](#)
[RmDir](#)

ChDrive   Statement

Description: This statement makes the specified drive the current drive.

Syntax: ChDrive driveLetter

Parameter: driveLetter

A string expression whose first letter is the drive you want to change to.

Examples: The following example makes the C drive the current drive.

```
ChDrive "c"
```

The next example makes the E drive the current drive. Only the first character is used.

```
ChDrive "Extended"
```

See Also

[ChDir](#)
[CurDir$()](#)
[Dir$()](#)
[MkDir](#)
[RmDir](#)
[DiskDrives](#)

CheckBox

Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a check box for a dialog box template.

Syntax: CheckBox x, y, width, height, name, .field

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the check box in dialog units.

width, height

The integers indicating the width and height of the check box in dialog units.

name

A string variable or literal that specifies the name of the check box.

.field

An integer variable used to set and/or retrieve the state of the check box. (The state is 0 for unchecked or 1 for checked.)

Example: The following example displays a dialog box with two check boxes within a group box.

```
Dim checkMsg2$, chkMsg$(1)

chkMsg(0) = "unchecked!"
chkMsg(1) = "checked!"
checkMsg2 = "No, check me!"

'Declare the dialog
Begin Dialog userDialog 15,28,100,100, "Untitled"
    GroupBox 4,4,84,51, "Check Boxes"
    CheckBox 10,15,48,14, "Check me!", .CheckBox1
    CheckBox 10,35,68,14, checkMsg2, .CheckBox2
    OKButton 55,64,41,14
End Dialog

'Declare the name for the instance
'of the template
Dim myDialog As userDialog
'Make the first check box initially checked
myDialog.CheckBox1 = 1

'Display the instance of the template
Dialog myDialog
```

```
'What was the result?
MsgBox "Check Box 1 was " + chkMsg(myDialog.CheckBox1)
MsgBox "Check Box 2 was " + chkMsg(myDialog.CheckBox2)
```

See Also

[CancelButton](#)
[Dialog](#)
[Dialog()](#)
[DropListBox](#)
[GroupBox](#)
[ListBox](#)
[OKButton](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PushButton](#)
[Text](#)
[TextBox](#)
[Begin Dialog . . .End Dialog](#)
[PictureButton](#)

Choose   Function

Syntax:	Choose(index,expression1,expression2,...,expression13)
Description:	Returns the expression at the specified index position.
Comments:	The index parameter specifies which expression is to be returned. If index is 1, then expression1 is returned; if index is 2, then expression2 is returned, and so on. If index is less than 1 or greater than the number of supplied expressions, then Null is returned.
	The Choose function returns the expression without converting its type. Each expression is evaluated before returning the selected one.
Example:	

```
'This example assigns a variable of indeterminate type to a.

Sub Main()
    Dim a As Variant
    Dim c As Integer
    c% = 2
    a = Choose(c%,"Hello, world",#1/1/94#,5.5,False)
    MsgBox "Item " & c% & " is '" & a & "'"            'Displays the date
passed as parameter 2.
End Sub
```

Platform(s):	All.

See Also

Switch (function)
IIf (function)
If...Then...Else (statement)
Select Case...End Select

Chr$( )   **Function**

Description:	This function returns the character that corresponds to the specified ANSI code.
Syntax:	Chr$(ANSICode)
Parameter:	ANSICode
	An integer between 0 and 255.
Description:	This function returns the character that corresponds to the specified ANSI code.
Examples:	The following example converts the ASCII value 65 to the string "A."

```
string65$ = Chr$(65)
```

The ASCII value for a carriage-return is 13 and the value for a linefeed is 10. The next example converts the carriage-return/linefeed characters into a string.

```
crlf$ = Chr$(13) + Chr$(10)
```

See Also

[Asc()](Asc())
[Str$()](Str$())

CInt( )   **Function**

Description: This function converts the specified numeric expression to an integer and returns that integer. A run-time error occurs if the specified expression is not within the correct range. The function is equivalent to assigning a numeric expression to a variable of type integer.

Syntax: CInt(exprN)

Parameter: exprN

A numeric expression in the range from -32768 to 32767.

Example: The following two assignments are equivalent.

```
x% = Cint(4.5)   'Explicit conversion
x% = 4.5         'Implicit conversion
```

See Also

Clipboard$ Statement

Description: This statement replaces anything currently in the Clipboard with the specified text string.

Syntax: Clipboard$ contents

Parameter: contents

A string expression.

Example: The following example puts a message in the Clipboard. A message box displays the contents of the Clipboard for verification.

```
'Put the message in the Clipboard
Clipboard$ "This is the message placed in the Clipboard."
MsgBox Clipboard$( )'Verify the placement
```

See Also

[Clipboard$()](#)
[Clipboard.GetText](#)
[Clipboard.SetText](#)

Clipboard$( )   Function

Description: This function returns a string expression containing the contents of the Clipboard. If the Clipboard is empty or does not contain text, an empty string is returned.

Syntax: Clipboard$( )

Example: The following example assigns the contents of the Clipboard to a string variable. If the Clipboard is empty, a message appears. Otherwise, the contents are displayed.

```
contents$ = Clipboard$( )

'Is the Clipboard empty?
If contents = "" Then
    'Empty Clipboard information message
    MsgBox "The Clipboard is empty.", 64
Else
    'Show the contents
    MsgBox contents
End If
```

See Also

Clipboard$
Clipboard.GetText
Clipboard.SetText

Clipboard.Clear   Method

Description: This method clears the contents of the Clipboard.

Syntax: Clipboard.Clear

Example: The following example clears the Clipboard. A message verifies that the Clipboard has been
                     cleared.

```
'Clear the Clipboard and verify clearance
Clipboard.Clear
If Clipboard$( ) = "" Then
    MsgBox "The Clipboard has been cleared."
Else
    MsgBox "The Clipboard has NOT been cleared."
End If
```

Clipboard.GetFormat   Method

Description: Returns True if data of the specified format is available in the Clipboard; returns False otherwise.

Syntax: Clipboard.GetFormat(*format*)

Comments: This method is used to determine whether the data in the Clipboard is of a particular format. The *format* parameter is an Integer representing the format to be queried:

Format	Description
1	Text
2	Bitmap
3	Metafile
8	Device-independent bitmap (DIB)
9	Color palette

Example: This example puts text on the Clipboard, checks whether there is text on the Clipboard, and if there is, displays it.

```
Sub Main()
    Clipboard$ "This is text on the Clipboard."
    If Clipboard.GetFormat(1) Then
        MsgBox Clipboard$
    Else
        MsgBox "No text on the Clipboard."
    End If
End Sub
```

See Also

[Clipboard$](#)
[Clipboard$()](#)

Clipboard.GetText   Method

Description: Returns the text contained in the Clipboard.

Syntax: Clipboard.GetText([*format*])

Comments: The *format* parameter, if specified, must be 1.

Example: This example retrieves the text from the Clipboard and checks to make sure that it contains the word "dog."

```
Sub Main()
    If Clipboard.GetFormat(1) Then
        s$ = Clipboard.GetText(1)
        If instr(0,s$,"dog",1) = 0 Then
                MsgBox "The Clipboard does not contain the word ""dog."""
        Else
                MsgBox "The Clipboard contains the correct word."
        End If
    Else
        MsgBox "The Clipboard does not contain text."
    End If
End Sub
```

See Also

[Clipboard$](#)
[Clipboard$()](#)
[Clipboard.SetText](#)

Clipboard.SetText   Method

Description: Copies the specified text string to the Clipboard.

Syntax: Clipboard.SetText *data$* [,*format*]

Comments: The *data$* parameter specifies the text to be copied to the Clipboard. The *format* parameter, if specified, must be 1.

Example: This example gets the contents of the Clipboard and uppercases it.

```
Sub Main()
    If Not Clipboard.GetFormat(1) Then Exit Sub
    Clipboard.SetText UCase$(Clipboard.GetText(1)),1
End Sub
```

See Also

[Clipboard$](#)
[Clipboard.GetText](#)
[Clipboard$()](#)

CLng( )   **Function**

Description: This function converts the specified numeric expression to a long and returns that long. A run-time error occurs if the specified expression is not within the correct range. The function is equivalent to assigning a numeric expression to a long variable.

Syntax: CLng(exprN)

Parameter: exprN

A numeric expression in the range from -2147483648 to 2147483647.

Example: The following two assignments are equivalent.

```
x& = Clng(4.5)   'Explicit conversion
x& = 4.5         'Implicit conversion
```

See Also

Close   **Statement**

Description: This statement closes the files whose numbers are specified, or if there are no parameters, it closes all files.

Syntax: Close [[#]fileNum [,[#]fileNum]]

Parameter: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

Example: The following statements show the three possible ways to use the Close statement.

```
' Open five files with file numbers 1 through 5

Open "testfil1" As #1
Open "testfil2" As #2
Open "testfil3" As #3
Open "testfil4" As #4
Open "testfil5" As #5

' Now close the files

Close #3            'Closes file #3
Close #2, #4  'Closes files #2 and #4
Close 'Closes the rest of the files (#1 and #5)
```

See Also

[Open](#)
[Reset](#)
[End](#)

ComboBox

Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a combination box for a dialog box template.

Syntax: ComboBox x, y, width, height, itemsArray, .field

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the combination box in dialog units.

width, height

The integers indicating the width and height of the combination box in dialog units.

itemsArray

The name of a one-dimensional string array that contains the elements to be placed into the combination box.

.field

A string variable used to select and/or retrieve an item from the combination box.

Example: The following example displays a dialog box containing a combination box.

```
Dim listOfItems$(9)

'Initialize the array of items
For i = 0 To 9
    listOfItems$(i) = "Item " + Str$(i)
Next
'Declares a dialog box template
Begin Dialog listDialog 15,24,100,84, "Lists"
    ComboBox 5,65,45,100, listOfItems, .ComboBox1
    OKButton 55,64,41,14
End Dialog

'Declares an instance of the template
Dim dialog1 As listDialog

'Displays the instance of the template
Dialog dialog1

'Display the user's selection
MsgBox dialog1.ComboBox1
```

See Also

Command$( )   Function

Description: This function returns a string containing the parameters from the command line that started the script. Command$( ) works only with scripts that have been saved as executable files with the extension .EXE.

Syntax: Command$[()]

Example: The following example shows the use of Command$( ) to obtain the command-line options (for a group and filename) required by the script.

```
'Get Group and Filename from the command line
Group_and_File$ = Command$
'Break it into the group and the filename
Lngth = Len(Group_and_File)
Brk = InStr(Group_and_File," ")
If Brk = 0 Then
    'An argument must be missing
    MsgBox "Improper Arguments Supplied. Proper syntax is: AddApp
Groupname Filename", 16, "FATAL ERROR"
Else
    Group$ = Trim$(Left$(Group_and_File, Brk-1))
    File$ = Trim$(Right$(Group_and_File, Lngth-Brk))
...
End If
```

See Also

Environ$()

Const   **Statement**

Description: This statement declares a constant. Constants are never declared inside subroutines or functions.

Syntax: Const name = expr [,name = expr]...

Parameters: name

The name of the constant that you are declaring.

expr

The value of the constant. It may include string or numeric literals; the predefined constants, TRUE or FALSE; or previously declared user-defined constants.

Example: The following messages are constants because they are used repeatedly in a variety of predefined dialog boxes.

```
Const Message1 = "Are you sure?", Message2 = "Please wait..."
Sub Main
    MsgBox Message2
    ...
End Sub
```

See Also

[Deftype](#)
[Let](#)
[= (equal To)](#)

Cos( )   Function

Description: This function returns the cosine of the specified angle. The value returned is a number of type double.

Syntax: Cos(angle)

Parameter: angle

A numeric expression specifying the number of radians in the angle.

Example: The x coordinate of a point on a circle of radius 1, centered at the origin can be found by computing the cosine of the angle at which the point lies on the circle.

```
'Calculate the x coordinate of the point
'at 30 degrees
x = Cos(30*PI/180)
```

See Also

[Tan()](#)
[Sin()](#)
[Atn()](#)

CSng( )　　　　　　**Function**

Description: This function converts the specified numeric expression to a single-precision number and returns that number. A run-time error occurs if the specified expression is not within the correct range. The function is equivalent to assigning a numeric expression to a variable of type single.

Syntax: CSng(exprN)

Parameter: exprN

A numeric expression (within the range for a single: approximately +/-3.4E+/-38).

Example: The following two assignments are equivalent.

```
x! = Csng(4)   'Explicit conversion
x! = 4         'Implicit conversion
```

See Also

[CCur (function)](#)
[CBool (function)](#)
[CDate, CVDate (functions)](#)
[CDbl()](#)
[CInt()](#)
[CStr()](#)
[CVar (function)](#)
[CVErr (function)](#)
[Single (data type)](#)

CStr( ) **Function**

Description:	This function converts a numeric expression to a string and returns that string. The first character of the string is a space if the number is positive or a minus if the number is negative.
Syntax:	CStr(exprN)
Parameter:	exprN
	A numeric expression to be converted to a string.
Example:	The following example converts the number 4.0 to a string.

```
string40$ = CStr(4.0)
```

See Also

CurDir$( )   Function

Description: This function returns the current directory on the specified drive. if the drive letter is invalid, a run-time error occurs.

Syntax: CurDir$[(drive)]

Parameter: drive

A string expression whose first letter is used as the drive specification. The default is the current drive.

Examples: The following example returns the current directory on the current drive and stores the result in a string.

```
currentDirectory$ = CurDir$
```

The following example returns the current directory on the C drive and stores the result in a string.

```
currentDirectory$ = CurDir$("C")
```

See Also

[ChDir](#)
[ChDrive](#)
[Dir$()](#)
[MkDir](#)
[RmDir](#)

Currency

Data type

Syntax:	Currency
Description:	A data type used to declare variables capable of holding fixed-point numbers with 15 digits to the left of the decimal point and 4 digits to the right.
Comments:	Currency variables are used to hold numbers within the following range:

$$-922{,}337{,}203{,}685{,}477.5808 \leq currency \leq 922{,}337{,}203{,}685{,}477.5807$$

Due to their accuracy, Currency variables are useful within calculations involving money.

The type-declaration character for Currency is @.

Storage

Internally, currency values are 8-byte integers scaled by 10000. Thus, when appearing within a structure, currency values require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

Platform(s): All.

See Also

[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CCur (function)](#)

CVar   **Function**

Syntax:	CVar(expression)
Description:	Converts expression to a Variant.
Comments:	This function is used to convert an expression into a variant. Use of this function is not necessary (except for code documentation purposes) because assignment to variant variables automatically performs the necessary conversion:

```
Sub Main()
    Dim v As Variant
    v = 4 & "th"            'Assigns "4th" to v.
    MsgBox "You came in: " & v
    v = CVar(4 & "th")      'Assigns "4th" to v.
    MsgBox "You came in: " & v
End Sub
```

Example:

```
'This example converts an expression into a Variant.

Sub Main()
    Dim s As String
    Dim a As Variant
    s = CStr("The quick brown fox ")
    msg = CVar(s & "jumped over the lazy dog.")
    MsgBox msg
End Sub
```

Platform(s):	All.

See Also

CVErr   Function

Syntax: CVErr(expression)

Description: Converts expression to an error.

Comments: This function is used to convert an expression into a user-defined error number.

A runtime error is generated under the following conditions:

If expression is Null.

If expression is a number outside the legal
range for errors, which is as follows:

$$0 <= \textit{expression} <= 65535$$

If expression is Boolean.

If expression is a String that can't be converted
to a number within the legal range.

Empty is treated as 0.

Example:

```
'This example simulates a user-defined error and displays the error
number.

Sub Main()
    MsgBox "The error is: " & CStr(CVErr(2046))
End Sub
```

Platform(s):   All.

See Also

[CCur (function)](#)
[CBool (function)](#)
[CDate, CVDate (functions)](#)
[CDbl( )](#)
[CInt( )](#)
[CLng( )](#)
[CSng( )](#)
[CStr( )](#)
[CVar (function)](#)
[IsError (function)](#)

Date   **Data type**

Syntax:	Date
Description:	A data type capable of holding date and time values.
Comments:	Date variables are used to hold dates within the following range:

January 1, 100 00:00:00 <= *date* <= December 31, 9999 23:59:59

-6574340 <= *date* <= 2958465.99998843

Internally, dates are stored as 8-byte IEEE double values. The integer part holds the number of days since December 31, 1899, and the fractional part holds the number of seconds as a fraction of the day. For example, the number 32874.5 represents January 1, 1990 at 12:00:00.

When appearing within a structure, dates require 8 bytes of storage. Similarly, when used with binary or random files, 8 bytes of storage are required.

There is no type-declaration character for Date.

Date variables that haven't been assigned are given an initial value of 0 (i.e., December 31, 1899).

Date Literals

Literal dates are specified using number signs, as shown below:

```
Dim d As Date
d = #January 1, 1990#
```

The interpretation of the date string (i.e., January 1, 1990 in the above example) occurs at runtime, using the current country settings. This is a problem when interpreting dates such as 1/2/1990. If the date format is M/D/Y, then this date is January 2, 1990. If the date format is D/M/Y, then this date is February 1, 1990. To remove any ambiguity when interpreting dates, use the universal date format:

date_variable = #*YY/MM/DD HH:MM:SS*#

The following example specifies the date June 3, 1965 using the universal date format:

```
Dim d As Date
d = #1965/6/3 10:23:45#
```

Platform(s):	All.

See Also

[Currency (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CDate, CVDate (functions)](#)

Date$ **Statement**

Description: This statement sets the system date to the specified date.

Syntax: Date$ = newDate

Parameter: newDate

A string expression in any of the following formats: MM-DD-YYYY, MM-DD-YY, MM/DD/YYYY, or MM/DD/YY.

Example: The following two statements for setting the date to June 2, 1993 are equivalent.

```
Date$ = 6/02/93
Date$ = 6/2/93
```

See Also

CDate, CVDate (functions)
Date$()
Time$()
Time$

Date$( )   Function

Description: This function returns the current system date as a 10-character string. The format for the returned date is MM-DD-YYYY.

Syntax: Date$[( )]

Example: The following statement saves the current date in a string.

currentDate$ = Date$( )

See Also

[Time$()](#)
[Time$](#)
[Date$](#)
[Now()](#)
[Format, Format$](#)
[DateSerial()](#)
[DateValue()](#)

DateAdd

Function

Description: Returns a Date variant representing the sum of *date* and a specified number (*increment*) of time intervals (*interval$*).

Syntax: DateAdd(*interval$, increment&, date*)

Comments: This function adds a specified number (*increment*) of time intervals (*interval$*) to the specified date (*date*). The following table describes the parameters to the DateAdd function:

Parameter	Description
interval$	String expression indicating the time interval used in the addition.
increment	Integer indicating the number of time intervals you wish to add. Positive values result in dates in the future; negative values result in dates in the past.
date	Any expression convertible to a Date. string expression. An example of a valid date/time string would be "January 1, 1993".

The *interval$* parameter specifies what unit of time is to be added to the given date. It can be any of the following:

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

To add days to a date, you may use either day, day of the year, or weekday, as they are all equivalent ("d", "y", "w").

The DateAdd function will never return an invalid date/time expression. The following example adds two months to December 31, 1992:

```
s# = DateAdd("m", 2, "December 31, 1992")
```

In this example, s is returned as the double-precision number equal to "February 28, 1993", not "February 31, 1993".

Symantec Basic generates a runtime error if you try subtracting a time interval that is larger than the time value of the date.

Example: This example gets today's date using the Date$ function; adds three years, two months, one week, and two days to it; and then displays the result in a dialog box.

```
Sub Main()
    Dim dDate#
    Dim sDate$
    sDate$ = Date$
    NewDate# = DateAdd("yyyy", 3, sDate)
    NewDate = DateAdd("m", 2, NewDate)
    NewDate = DateAdd("ww", 1, NewDate)
    NewDate = DateAdd("d", 2, NewDate)
    s$ = "Three years, two months, one week, and two days from now will
be: "
    s$ = s$ + Format$(NewDate, "long date")
    MsgBox s$
End Sub
```

See Also

DateDiff

DateDiff   Function

Description: Returns a Date variant representing the number of given time intervals between *date1* and *date2*.

Syntax: DateDiff(*interval$,date1,date2*)

Comments: The following table describes the parameters:

Parameter	Description
interval$	String expression indicating the specific time interval you wish to find the difference between.
date1	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".
date2	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".

The following table lists the valid time interval strings and the meanings of each. The Format$ function uses the same expressions.

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

To find the number of days between two dates, you may use either day or day of the year, as they are both equivalent ("d", "y").

The time interval weekday ("w") will return the number of weekdays occurring between *date1* and *date2*, counting the first occurrence but not the last. However, if the time interval is week ("ww"), the function will return the number of calendar weeks between *date1* and *date2*, counting the number of Sundays. If *date1* falls on a Sunday, then that day is counted, but if *date2* falls on a Sunday, it is not counted.

The DateDiff function will return a negative date/time value if *date1* is a date later in time than *date2*.

Example: This example gets today's date and adds ten days to it. It then calculates the difference between the two dates in days and weeks and displays the result.

```
Sub Main()
    Today$ = Date$
    TodayR#  = DateValue(Date$)
    NextWeek# = DateAdd("d", 10, Today)
    Difdays# = DateDiff("d", Today, NextWeek)
    Difweek# = DateDiff("ww", Today, NextWeek)
    S$ = "The difference between " + Today + " and " + Str$(NextWeek)
    S$ = S$ + " is: " + Str$(Difdays) + " days or " + Str$(DifWeek) + "
weeks"
    MsgBox S$
End Sub
```

See Also

[DateAdd](#)

DatePart   Function

Description: Returns an Integer representing a specific part of a date/time expression.

Syntax: DatePart(*interval$,date*)

Comments: The DatePart function decomposes the specified date and returns a given date/time element. The following table describes the parameters:

Parameter	Description
interval$	String expression that indicates the specific time interval you wish to identify within the given date.
date	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1995".

The following table lists the valid time interval strings and the meanings of each. The Format$ function uses the same expressions.

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

The weekday expression starts with Sunday as 1 and ends with Saturday as 7.

Example: This example displays the parts of the current date.

```
Const crlf$ = Chr$(13) + Chr$(10)

Sub Main()
    Today$ = Date$
    Qtr = DatePart("q",Today)
    Yr  = DatePart("yyyy",Today)
    Mo  = DatePart("m",Today)
    Wk  = DatePart("ww",Today)
```

```
        Da  = DatePart("d",Today)
        S$ = "Quarter: " + Str$(Qtr) + crlf
        S$ = S$ + "Year        : " + Str$(Yr) + crlf
        S$ = S$ + "Month       : " + Str$(Mo) + crlf
        S$ = S$ + "Week        : " + Str$(Wk) + crlf
        S$ = S$ + "Day         : " + Str$(Da) + crlf
        MsgBox S$
End Sub
```

See Also

[Day()](#)
[Minute()](#)
[Second()](#)
[Month()](#)
[Year()](#)
[Hour()](#)
[Weekday()](#)
[Format, Format$](#)

DateSerial( )   **Function**

Description:	This function returns a double-precision number that is a serial representation of the specified date. It is the number of days sinceDec. 30, 1899, which is the zero date.
Syntax:	DateSerial(year, month, day)
Parameters:	year
	A numeric expression specifying a year as YYYY or YY. When YY is used, the year is assumed to be in the 20th century.
	month
	A numeric expression specifying a month as 1 to 12.
	day
	A numeric expression specifying a day as 1 to 31.
Examples:	The following example obtains the serial date for December 12, 1912.

```
serialDT# = DateSerial(12,12,12)
```

The next example obtains the serial date for January 1, 2010.

```
serialDT# = DateSerial(2010,1,1)
```

See Also

[DateValue()](#)
[TimeSerial()](#)
[TimeValue()](#)
[CDate, CVDate (functions)](#)

DateValue( )   Function

Description: This function returns a double-precision number that is the serial representation of the specified date.

Syntax: DateValue(dateStr)

Parameter: dateStr

A string expression for a date. The order of the date items depends on the settings contained in the [intl] section of the WIN.INI file. Check the International dialog box from the Control Panel to review the settings. The month can be specified as a word, three-letter abbreviation (minus the period), or a number. Valid date separators are the slash (/), hyphen (-), and comma (,). Dates can contain an optional time specification, but this is not used in the formation of the returned value. If the day is missing, the first day of the month is assumed. If the year is missing, the current year is assumed.

Example: The following example obtains the serial date for December 12, 1912.

```
serialDT# = DateValue("12-12-12")
```

See Also

[TimeSerial()](#)
[TimeValue()](#)
[DateSerial()](#)

Day( )   **Function**

Description: This function returns the day of the date encoded in the specified serial date. The value returned is an integer ranging from 1 to 31.

Syntax: Day(serial)

Parameter: serial

A double-precision number containing a serial date.

Example: After calling the Now( ) function, you can extract the current day from the date and time.

```
'Get the current date and time
serialDT# = Now( )

'Now extract the value
theDay% = Day(serialDT)
```

See Also

[Minute()](#)
[Second()](#)
[Month()](#)
[Year()](#)
[Hour()](#)
[Weekday()](#)
[DatePart](#)

DDB   Function

Description: Calculates the depreciation of an asset for a specified *Period* of time using the double-declining balance method.

Syntax: DDB(*Cost, Salvage, Life, Period*)

Comments: The double-declining balance method calculates the depreciation of an asset at an accelerated rate. The depreciation is at its highest in the first period and becomes progressively lower in each additional period. DDB uses the following formula to calculate the depreciation:

$$DDB = ((Cost - Total\_depreciation\_from\_all\_other\_periods) * 2)/Life$$

The DDB function uses the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset
Salvage	Double representing the estimated value of the asset at the end of its predicted useful life
Life	Double representing the predicted length of the asset's useful life
Period	Double representing the period for which you wish to calculate the depreciation

Life and *Period* must be expressed using the same units.

For example, if *Life* is expressed in months, then *Period* must also be expressed in months.

Example: This example calculates the depreciation for capital equipment that cost $10,000, has a service life of ten years, and is worth $2,000 as scrap. The dialog box displays the depreciation for each of the first four years.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    S$ = "Depreciation Table" + crlf + crlf
    For yy = 1 To 4
       CurDep# = DDB(10000.0, 2000.0, 10, yy)
       S$ = S$ + "Year " + Str$(yy) + " : " + Str$(CurDep) + crlf
    Next yy
    MsgBox S$
```

```
        End Sub
```

See Also

[Sln (function)](#)
[SYD](#)

Declare

Statement

Description:	This statement, which declares a procedure or function from a DLL, must precede any call to that DLL routine and cannot be inside a subroutine or function. Declare statements are valid only for the script in which they are declared.
	String parameters are always passed from the script to DLL routines by reference. If a DLL routine modifies a specified string variable, then there must be sufficient space within the string to hold the returned characters. Use the Space$( ) function to create a string of sufficient length.
	DLLs containing the routines are loaded when the routine is called for the first time. If a script references an external DLL that does not exist, a run-time error occurs.
Syntax1:	Declare Sub name [Lib libName [Alias realName]] [([parameterList])]
Syntax2:	Declare Function name [Lib libName [Alias realName]] [([parameterList])] [As type]
Parameters:	name
	Any valid name in BASIC. This is either the name of the external routine or an internal alias for it. When this is the name of a function, it can include a type declarator to indicate the type of value the function returns.
	libName
	A string expression specifing a complete or relative pathname for the DLL that contains the external routine. If no directory is specified, \WINDOWS\SYSTEM is the default.)
	realName
	A string expression containing the external routine's name. Used when name is not the real name of the external routine as it appears within the DLL. Use the Alias clause when the name of an external routine contains invalid characters or conflicts with a name in your script.
	parameterList
	A list of parameters for the routine separated by commas. The syntax is: parameter [, parameter]... and the syntax for each parameter is: [ByVal] parameterName [( )] [As type]
	The parameter list specifies the parameters received by the external routine. The parameter list must match the syntax of the referenced routine exactly; otherwise, unpredictable results may occur. By default, BASIC passes parameters by reference. When a routine requires a value rather than a reference, use the ByVal reserved word to indicate this.
	type
	The data type that the function returns. This is used when no type declarator is appended to name. If neither is used, the type is as determined by a Deftype statement or, by default, the type is an

integer.

Examples: All of the following Declare statements allow the script to use the GetCurrentTime function in USER.EXE. The third example uses GetTime as an alias for the GetCurrentTime function because the name GetCurrentTime is already used in the script that calls the function.

```
Declare Function GetCurrentTime Lib "USER"( ) As Long
Declare Function GetCurrentTime& Lib "USER"( )
Declare Function GetTime Lib "USER" Alias "GetCurrentTime" As Long
```

See Also

[Call](#)
[Sub . . .End Sub](#)
[Function . . .End Function](#)

Deftype

Statement

Description:	This statement controls automatic type declaration of variables.
	Explicit type declarations, such as with Dim statements or type declarators, take precedence over the Deftype statement. With no Deftype statement, Dim statement, or declarator, a variable is declared implicitly as an integer (DefInt A-Z). This statement affects the compiling of scripts.
Syntax:	DefInt letterRange
	DefLng letterRange
	DefDbl letterRange
	DefSng letterRange
	DefStr letterRange
Parameter:	letterRange
	Range of letters. Its syntax is:
	letter [-letter] [, letter [-letter]]...
	For example: a, c-f, n
Example:	The Deftype statements in the following example make any variables (that are not explicitly declared) into integers if their names start with I, M, or Q; into longs if their names start with A, B, C, or N; and into strings if their names start with T through Z.

```
DefInt I, M, Q
DefLng A-C, N
DefStr T-Z
Sub Main
    ...
End Sub
```

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Integer (data type)](#)

Dialog   Statement

Description: This statement displays an instance of the specified user-defined dialog box template. The Dialog statement ends when the user selects a command button.

Syntax: Dialog userDlg

Parameter: userDlg

The name of an instance of a dialog box template.

Example: The following example displays a dialog box containing a text control and a text box for entering a serial number. When the Dialog statement ends, a message box displays the serial number that was entered.

```
Begin Dialog SerialNumDialog 16,32,110,33, "Serial Number"
    Text 5,6,57,8, "Serial Number:"
    TextBox 5,15,51,12, .SerialNumber
    OKButton 64,13,41,14
End Dialog

Dim dialog1 As SerialNumDialog

Dialog dialog1

'Display the entered serial number
MsgBox dialog1.SerialNumber
```

See Also

Dialog()

Dialog( )   Function

Description: This function displays an instance of the specified user-defined dialog box template. It returns -1 (TRUE) when the OK button is selected, 0 (FALSE) when the Cancel button is selected, or the positive integer associated with the selected user-defined command button.

Syntax: Dialog(userDlg)

Parameter: userDlg

The name of an instance of a dialog box template.

Example: The following example displays a dialog box containing four buttons labeled with the compass directions and arranged in a circle.
The Dialog( ) function returns the number of the selected button, which is then used as a subscript to display the text of the selected direction. The buttons are defined clockwise, which also determines their numbering, starting from North.

```
Dim direction$(4)
direction(1) = "N"
direction(2) = "E"
direction(3) = "S"
direction(4) = "W"
'Define 4 command (push) buttons
Begin Dialog DirectionsDialog 16,32,122,119, "Directions"
    PushButton 50,6,21,21, direction(1)
    PushButton 93,48,21,21, direction(2)
    PushButton 50,91,21,21, direction(3)
    PushButton 8,48,21,21, direction(4)
End Dialog
Dim DirDialog As DirectionsDialog
'Which direction was selected?
MsgBox direction(Dialog(DirDialog))
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog](#)

[DropListBox](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[Picture](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog . . .End Dialog](#)

[PictureButton](#)

Dim   **Statement**

Description: This statement declares variables and their types. The maximum number of dimensions for an array variable is 60. Dynamic arrays can be declared without bounds: for example, Dim Array1( ).

Syntax: Dim name[(subscripts)][As type]

   [,name[(subscripts)][As type]]...

Parameters: name

The name of the variable, followed by an optional type declarator.

subscripts

Numeric expressions indicating the lower and upper bounds for elements in array variables. If the lower bound is not specified, it is 0 or 1, depending on the base option. Its syntax is:

[lower To] upper [, [lower To] upper]…

type

Specifies the type of the variable. This is used when a type declarator is not appended to the name. If neither is used, the type is as determined by a Deftype statement or, by default, the type is an integer.

Examples: Both of the Dim statements in the following example declare a string variable.

```
Sub Main
    Dim first_name As String     'User's first name
    Dim last_name$                    'User's last name
    ...
End Sub
```

The following statement declares a two-dimensional string array with subscripts from 0 to 2 in the first dimension and from 0 to 10 in the second.

```
Dim MyStrings$(2,10)
```

The following statement declares a one-dimensional string array with subscripts from 5 to x (where x is a numeric variable that has been declared prior to this Dim statement).

```
Dim FileNames(5 To x) As String
```

See Also

[Redim](#)
[Public](#)
[Private](#)
[Option Base](#)

Dir$( )   **Function**

Description: This function returns a string containing the first file matching the file specification string or the empty string when no file matches. It returns an error if a valid file specification is not used the first time this function is called.

Syntax: Dir$[(fileSpec)]

Parameters: fileSpec

A string expression containing a complete or relative path and the wildcards (* and ?) to specify a filename. The default is the previous file specification.

Example: The following example processes each file in a directory, one at a time.

```
'Find first file
file$ = Dir$("*.*")
'Check if all files have already been found
While file <> ""
    ...'Process another one
    'Call with no specification to find next file
    file = Dir$
Wend
```

See Also

[ChDir](#)
[ChDrive](#)
[CurDir$()](#)
[MkDir](#)
[RmDir](#)
[FileList](#)

DiskDrives   Statement

Description: This statement fills an array with all the valid drive letters. Use the functions LBound( ) and UBound( ) to determine the size of the resulting array.

Syntax: DiskDrives list

Parameter: list

                   One-dimensional string array to be filled with the drive letters. The array is resized to hold the exact number of valid drives.

Example: The following example processes all the valid drive letters.

```
'Declare a dynamic string array
Dim buffer$( )

'Put all the valid drive letters into buffer
DiskDrives buffer

    ... 'Do something with each drive
numDrivesFound% = UBound(buffer)-LBound(buffer)+1
For i = UBound(buffer) To LBound(buffer)rive letter
Next i
```

See Also

[ChDrive](#)
[DiskFree()](#)

DiskFree( )   Function

Description: This function returns the free space available on the specified drive in bytes. The value returned is a number of type long.

Syntax: DiskFree[(drive)]

Parameter: drive

A string expression whose first letter is the drive whose free space is to be determined. The default is the current drive.

Examples: The following example finds the amount of free space on the current drive and stores it in the long variable freeSpace.
freeSpace& = DiskFree( )
    The next example finds the amount of free space on the C drive and stores it in the long variable moreFreeSpace.
moreFreeSpace& = DiskFree("C")

See Also

[ChDrive](#)
[DiskDrives](#)

DlgControlId   Function

Syntax:	DlgControlId(ControlName$)
Description:	Returns an Integer containing the index of the specified control as it appears in the dialog box template.
Comments:	The first control in the dialog box template is at index 0, the second is at index 1, and so on.
	The ControlName$ parameter contains the name of the .Identifier parameter associated with that control in the dialog box template.
	The BasicScript statements and functions that dynamically manipulate dialog box controls identify individual controls using either the .Identifier name of the control or the control's index. Using the index to refer to a control is slightly faster but results in code that is more difficult to maintain.
Example:	

```
'This example uses DlgControlId to verify which control was triggered
'and branches the dynamic dialog script accordingly.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 Then
        'Enable the next three controls.
        If DlgControlId(ControlName$) = 2 Then
                For i = 3 to 5
                        DlgEnable i,DlgValue("CheckBox1")
                Next i
                DlgProc = 1          'Don't close the dialog box.
        End If
    ElseIf Action% = 1 Then
        'Set initial state upon startup
        For i = 3 to 5
                DlgEnable i,DlgValue("CheckBox1")
        Next i
    End If
End Function

Sub Main()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        CheckBox 24,16,72,8,"Click Here",.CheckBox1
        CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2
        CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3
        CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4
        CheckBox 24,72,76,8,"Main Option 2",.CheckBox5
```

```
        End Dialog
        Dim d As UserDialog
        Dialog d
    End Sub
```

Platform(s):   Windows, DOS, Win32, Macintosh, OS/2.

See Also

[DlgEnable (function)](#)
[DlgEnable (statement)](#)
[DlgFocus (function)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray (function)](#)
[DlgListBoxArray (statement)](#)
[DlgSetPicture](#)
[DlgText$ (function)](#)
[DlgText](#)
[DlgValue (function)](#)
[DlgValue (statement)](#)
[DlgVisible (function)](#)
[DlgVisible (statement)](#)

Do...Loop Construct

Description: This construct repeats a block of statements while or until a condition is true. If no condition is specified, the loop repeats until an Exit Do statement is encountered.

Syntax1:
Do {While | Until} logicalExpr
   [statements]
Loop

Syntax2:
Do
   [statements]
Loop {While | Until} logicalExpr

Syntax3:
Do
   [statements]
Loop statement

s   Any series of executable statements.

logicalExpr   Expression containing relational and/or logical operators.

Example: The following example loops until a user inputs a positive integer or zero. The statements in the Do loop execute at least once because the condition is at the end of the loop.

```
' Input number for calculation.
Dim FactNum As Integer

Do
    ' get number greater than zero
    FactNum = Val(InputBox$ ("Enter a positive integer."))
    If FactNum <= 0 Then
       MsgBox "Try again"
    End If
Loop Until FactNum > 0
' Now FactNum is greater than or equal to zero.
```

See Also

[For . . .Next](#)
[While . . .Wend](#)

DlgEnable Function

Syntax: DlgEnable(ControlName$ | ControlIndex)

Description: Returns True if the specified control is enabled; returns False otherwise.

Comments: Disabled controls are dimmed and cannot receive keyboard or mouse input.

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

You cannot disable the control with the focus.

Example:
```
If DlgEnable("SaveOptions") Then
    MsgBox "The Save Options are enabled."
End If

    If DlgEnable(10) And DlgVisible(12) Then code = 1 Else code = 2
```

See Also

DlgEnable(statement)
DlgFocus(Function)
DlgFocus(statement)
DlgListBoxArray(Function)
DlgListBoxArray(statement)
DlgText
DlgText$ (function)
DlgSetPicture
DlgValue(Function)
DlgValue(statement)
DlgVisible(Function)
DlgVisible(statement)

DlgEnable   Statement

Syntax:	DlgEnable {ControlName$	ControlIndex} [,isOn]
Description:	Enables or disables the specified control.
Comments:	Disabled controls are dimmed and cannot receive keyboard or mouse input.

The *isOn* parameter is an Integer specifying the new state of the control. It can be any of the following values:

0   The control is disabled.

1   The control is enabled.

Omitted   Toggles the control between enabled and disabled.

Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example:
```
DlgEnable "SaveOptions", False   'Disable the Save Options control.

    DlgEnable "EditingOptions"              'Toggle a group of option
buttons.

    For i = 0 To 5
      DlgEnable i,True                          'Enable six
controls.
    Next I
```

See Also

[DlgEnable(Function)](#)
[DlgFocus(Function)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray(Function)](#)
[DlgListBoxArray(statement)](#)
[DlgSetPicture](#)
[DlgText](#)
[DlgText$ (function)](#)
[DlgValue(Function)](#)
[DlgValue(statement)](#)
[DlgVisible(statement)](#)
[DlgVisible(Function)](#)

DlgFocus   Function

Syntax: DlgFocus$[()]

Description: Returns a String containing the name of the control with the focus.

Comments: The name of the control is the *.Identifier* parameter associated with the control in the dialog box template.

Example: This code fragment makes sure that the control being disabled does not currently have the focus (otherwise, a runtime error would occur).

```
If DlgFocus$ = "Files" Then          'Does it have the focus?
    DlgFocus "OK"                          'Change the focus to
another control.
End If
DlgEnable "Files", False             'Now we can disable the control.
```

See Also

[DlgEnable(Function)](#)
[DlgEnable(statement)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray(Function)](#)
[DlgListBoxArray(statement)](#)
[DlgSetPicture](#)
[DlgText](#)
[DlgText$ (function)](#)
[DlgValue(Function)](#)
[DlgValue(statement)](#)
[DlgVisible(Function)](#)
[DlgVisible(statement)](#)

DlgFocus   **Statement**

Syntax: DlgFocus ControlName$ | ControlIndex

Description: Sets focus to the specified control.

Comments: A runtime error results if the specified control is hidden, disabled, or nonexistent.

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example: This code fragment makes sure that the control being disabled does not currently have the focus (otherwise, a runtime error would occur).

```
If DlgFocus$ = "Files" Then      'Does it have the focus?
    DlgFocus "OK"                'Change the focus to another control.
End If
DlgEnable "Files", False         'Now we can disable the control.
```

See Also

DlgEnable(Function)

DlgEnable(statement)

DlgFocus(Function)

DlgListBoxArray(Function)

DlgListBoxArray(statement)

DlgSetPicture

DlgText

DlgValue(Function)

DlgValue(statement)

DlgVisible(Function)

DlgVisible(statement)

DlgListBoxArray   Function

Syntax:	DlgListBoxArray({*ControlName$* | *ControlIndex*}, *ArrayVariable*)
Description:	Fills a list box, combo box, or drop list box with the elements of an array, returning an Integer containing the number of elements that were actually set into the control.
Comments:	The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
	The *ArrayVariable* parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. *ArrayVariable* can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
Example:	This dialog function refills an array with files.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 And ControlName$ = "Files" Then
        Dim NewFiles$()
    'Create a new dynamic array.
        FileList NewFiles$,"*.txt"                        'Fill the array with
files.
        R% = DlgListBoxArray "Files",NewFiles$  'Set items in the list
box.
        DlgValue "Files",0                                'Set the
selection to the first item.
        DlgProc = 1                                               'Don't
close the dialog box
    End If
    MsgBox Str$(R) + " items were added to the list box."
End Function
```

See Also

DlgListBoxArray   Statement

Syntax:	DlgListBoxArray {ControlName$ | ControlIndex}, ArrayVariable
Description:	Fills a list box, combo box, or drop list box with the elements of an array.
Comments:	The ControlName$ parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
	The ArrayVariable parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. *ArrayVariable* can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
Example:	This dialog function refills an array with files.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 And ControlName$ = "Files" Then
       Dim NewFiles$()
    'Create a new dynamic array.
       FileList NewFiles$,"*.txt"                      'Fill the array with
files.
       DlgListBoxArray "Files",NewFiles$       'Set items in the list
box.
       DlgValue "Files",0                                    'Set the
selection to the first item.
    End If
End Function
```

See Also

[DlgEnable(Function)](#)
[DlgEnable(statement)](#)
[DlgFocus(Function)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray(Function)](#)
[DlgSetPicture](#)
[DlgText](#)
[DlgValue(Function)](#)
[DlgValue(statement)](#)
[DlgVisible(Function)](#)
[DlgVisible(statement)](#)

DlgProc   **Function**

Syntax:	Function DlgProc(ControlName$, Action, SuppValue) As Integer
Description:	Describes the syntax, parameters, and return value for dialog functions.
Comments:	Dialog functions are called by Symantec Basic during the processing of a custom dialog box. The name of a dialog function (*DlgProc*) appears in the Begin Dialog statement as the *.DlgProc* parameter.

Dialog functions require the following parameters:

Parameter	Description
ControlName$	String containing the name of the control associated with *Action*.
Action	Integer containing the action that called the dialog function.
SuppValue	Integer of extra information associated with *Action*. For some actions, this parameter is not used.

When Symantec Basic displays a custom dialog box, the user may click on buttons, type text into edit fields, select items from lists, and perform other actions. When these actions occur, Symantec Basic calls the dialog function, passing it the action, the name of the control on which the action occurred, and any other relevent information associated with the action.

The following table describes the different actions sent to dialog functions:

Action	Description
1	This action is sent immediately before the dialog box is shown for the first time. This gives the dialog function a chance to prepare the dialog box for use. When this action is sent, *ControlName$* contains a zero-length string, and *SuppValue* is 0.

The return value from the dialog function is ignored in this case.

Before Showing the Dialog Box

After action 1 is sent, Symantec Basic performs additional processing before the dialog box is shown. Specifically, it cycles though the dialog box controls checking for visible picture or picture button controls. For each visible picture or picture button control, Symantec Basic attempts to load the associated picture.

In addition to checking picture or picture button controls, Symantec Basic will automatically hide any control outside the confines of the visible portion of the dialog box. This prevents the user from tabbing to controls that cannot be seen. However, it does not prevent you from showing these controls with the DlgVisible statement in the dialog function.

2   This action is sent when:

A button is clicked, such as OK, Cancel, or a push button. In this case, *ControlName$* contains the name of the button. *SuppValue* contains 1 if an OK button was clicked and 2 if a Cancel button was clicked; *SuppValue* is undefined otherwise.

If the dialog function returns 0 in response to this action, then the dialog box will be closed. Any other value causes Symantec Basic to continue dialog processing.

A check box's state has been modified. In this case, *ControlName$* contains the name of the check box, and *SuppValue* contains the new state of the check box (1 if on, 0 if off).

An option button is selected. In this case, *ControlName$* contains the name of the option button that was clicked, and *SuppValue* contains the index of the option button within the option button group (0-based).

The current selection is changed in a list box, drop list box, or combo box. In this case, *ControlName$* contains the name of the list box, combo box, or drop list box, and *SuppValue* contains the index of the new item (0 is the first item, 1 is the second, and so on).

3   This action is sent when the content of a text box or combo box has been changed. This action is only sent when the control loses focus. When this action is sent, *ControlName$* contains the name of the text box or combo box, and *SuppValue* contains the length of the new content.

The dialog function's return value is ignored with this action.

4   This action is sent when a control gains the focus. When this action is sent, *ControlName$* contains the name of the control gaining the focus, and *SuppValue* contains the index of the control that lost the focus (0-based).

The dialog function's return value is ignored with this action.

5   he dialog box is idle. If the dialog function returns 1 in response to this action, then the idle action will continue to be sent. If the dialog function returns 0, then Symantec Basic will not send any additional idle actions.

When the idle action is sent, *ControlName$* contains a zero-length string, and *SuppValue* contains the number of times the idle action has been sent so far.

6   This action is sent when the dialog box is moved. The *ControlName$* parameter contains a zero-length string, and *SuppValue* is 0.

The dialog function's return value is ignored with this action.

User-defined dialog boxes cannot be nested. In other words, the dialog function of one dialog box cannot create another user-defined dialog box. You can, however, invoke any built-in dialog box, such as MsgBox or InputBox$.

Within dialog functions, you can use the following additional Symantec Basic statements and functions. These statements allow you to manipulate the dialog box controls dynamically.

DlgVisible	DlgText$	DlgText
DlgSetPicture	DlgListBoxArray	DlgFocus
DlgEnable	DlgControlId	

For compatibility with previous versions of Symantec Basic, the dialog function can optionally be declared to return a Variant. When returning a variable, Symantec Basic will attempt to convert the variant to an Integer. If the returned variant cannot be converted to an Integer, then 0 is assumed to be returned from the dialog function.

Example: This dialog function enables/disables a group of option buttons when a check box is clicked.

```
Function SampleDlgProc(ControlName$, Action%, SuppValue%)
    If Action = 2 And ControlName$ = "Printing" Then
        DlgEnable "PrintOptions",SuppValue%
        SampleDlgProc = 1   'Don't close the dialog box
    End If
End Function

Sub Main()
    Begin Dialog SampleDialogTemplate
```

```
34,39,106,45,"Sample",.SampleDlgProc
        OKButton 4,4,40,14
        CancelButton 4,24,40,14
        CheckBox 56,8,38,8,"Printing",.Printing
        OptionGroup .PrintOptions
                OptionButton 56,20,51,8,"Landscape",.Landscape
                OptionButton 56,32,40,8,"Portrait",.Portrait
    End Dialog
    Dim SampleDialog As SampleDialogTemplate
    SampleDialog.Printing = 1
    r% = Dialog(SampleDialog)
End Sub
```

See Also

Begin Dialog . . .End Dialog

DlgSetPicture   Statement

Description: Changes the content of the specified picture or picture button control.

Syntax: DlgSetPicture {ControlName$ | ControlIndex},PictureName$,PictureType

Comments: The DlgSetPicture statement accepts the following parameters:

Parameter	Description
ControlName$	String containing the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specified control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
PictureName$	String containing the name of the picture. If *PictureType* is 0, then this parameter specifies the name of the file containing the image. If *PictureType* is 10, then *PictureName$* specifies the name of the image within the resource of the picture library.
	If *PictureName$* is empty, then the current picture associated with the specified control will be deleted. Thus, a technique for conserving memory and resources would involve setting the picture to empty before hiding a picture control.
PictureType	Integer specifying the source for the image. The following sources are supported:
0	The image is contained in a file on disk.
10	The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the *PictureName$* parameter must be specified with the Begin Dialog statement.

Examples:

```
DlgSetPicture "Picture1","\windows\checks.bmp",0      'Set picture from a
file.
```

```
DlgSetPicture 27,"FaxReport",10                              'Set control
10's image

         'from a library.
```

See Also

[DlgEnable(Function)](#)
[DlgEnable(statement)](#)
[DlgFocus(Function)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray(Function)](#)
[DlgListBoxArray(statement)](#)
[DlgText](#)
[DlgValue(Function)](#)
[DlgValue(statement)](#)
[DlgVisible(Function)](#)
[DlgVisible(statement)](#)
[Picture](#)
[PictureButton](#)

DlgText

Statement

Description: Changes the text content of the specified control.

Syntax: DlgText {ControlName$ | ControlIndex}, NewText$

Comments: The effect of this statement depends on the type of the specified control:

Control Type	Effect of DlgText
Picture	Runtime error.
Option group	Runtime error.

Drop list box Sets the current selection to the item matching *NewText$*. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with *NewText$*. If no match is found, then the selection is removed.

OK button   Sets the label of the control to *NewText$*.

Cancel button Sets the label of the control to *NewText$*.

Push button   Sets the label of the control to *NewText$*.

List box   Sets the current selection to the item matching *NewText$*. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with *NewText$*. If no match is found, then the selection is removed.

Combo box   Sets the content of the edit field of the combo box to *NewText$*.

Text Sets the label of the control to *NewText$*.

Text box   Sets the content of the text box to *NewText$*.

Group box   Sets the label of the control to *NewText$*.

Option button Sets the label of the control to *NewText$*.

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example:
```
DlgText "GroupBox1","Save Options"              'Change text of group box
1.

If DlgText$(9) = "Save Options" Then
    DlgText 9,"Editing Options"              'Change text to "Editing
Options".
End If
```

See Also

[DlgEnable(Function)](#)

[DlgEnable(statement)](#)

[DlgFocus(Function)](#)

[DlgFocus(statement)](#)

[DlgListBoxArray(Function)](#)

[DlgListBoxArray(statement)](#)

[DlgSetPicture](#)

[DlgText](#)

[DlgValue(Function)](#)

[DlgValue(statement)](#)

[DlgVisible(Function)](#)

[DlgVisible(statement)](#)

DlgText$ Function

Syntax:	DlgText$(ControlName$ | ControlIndex)
Description:	Returns the text content of the specified control.
Comments:	The text returned depends on the type of the specified control:

Control Type	Value Returned by DlgText$
Picture	No value is returned. A runtime error occurs.
Option group	No value is returned. A runtime error occurs.
Drop list box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
OK button	Returns the label of the control.
Cancel button	Returns the label of the control.
Push button	Returns the label of the control.
List box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
Combo box	Returns the content of the edit field portion of the combo box.
Text	Returns the label of the control.
Text box	Returns the content of the control.
Group box	Returns the label of the control.
Option button	Returns the label of the control.

The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example:

```
'This code fragment makes sure the user enters a correct value.
'If not, the control returns focus back to the TextBox for correction.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 and ControlName$ = "OK" Then
        If IsNumeric(DlgText$("TextBox1")) Then
                Msgbox "Duly Noted."
        Else
                Msgbox "Sorry, you must enter a number."
                DlgFocus "TextBox1"
                DlgProc = 1
        End If
    End If
End Function

Sub Main()
    Dim ListBox1$()
    Begin Dialog UserDialog ,,112,74,"Untitled",.DlgProc
        TextBox 12,20,88,12,.TextBox1
        OKButton 12,44,40,14
        CancelButton 60,44,40,14
        Text 12,11,88,8,"Enter Desired Salary:",.Text1
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

Platform(s):   Windows, DOS, Win32, Macintosh, OS/2.

See Also

[DlgEnable (function)](#)

[DlgEnable (statement)](#)

[DlgFocus (function)](#)

[DlgFocus(statement)](#)

[DlgListBoxArray (function)](#)

[DlgListBoxArray (statement)](#)

[DlgSetPicture](#)

[DlgText](#)

[DlgValue (function)](#)

[DlgValue (statement)](#)

[DlgVisible (function)](#)

[DlgVisible (statement)](#)

DlgValue   **Function**

Description:	Returns an Integer indicating the value of the specified control.
Syntax:	DlgValue(ControlName$ | ControlIndex)
Comments:	The value of any given control depends on its type, according to the following table:

Control Type	DlgValue Returns
Option group	The index of the selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the selected item.
Drop list box	The index of the selected item.
Check box	1 if the check box is checked; 0 otherwise.

A runtime error is generated if DlgValue is used with controls other than those listed in the above table.

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example:	See DlgValue (statement).

See Also

[DlgEnable(Function)](#)
[DlgEnable(statement)](#)
[DlgFocus(Function)](#)
[DlgFocus(statement)](#)
[DlgListBoxArray(Function)](#)
[DlgListBoxArray(statement)](#)
[DlgSetPicture](#)
[DlgText](#)
[DlgValue(statement)](#)
[DlgVisible(Function)](#)
[DlgVisible(statement)](#)

DlgValue

Statement

Description:	Changes the value of the given control.
Syntax:	DlgValue {ControlName$	ControlIndex},Value
Comments:	The value of any given control is an Integer and depends on its type, according to the following table:

Control Type	Description of *Value*
Option group	The index of the new selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the new selected item.
Drop list box	The index of the new selected item.
Check box	1 if the check box is to be checked; 0 if the check is to be removed.

A runtime error is generated if DlgValue is used with controls other than those listed in the above table.

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example: This code fragment toggles the value of a check box.

```
If DlgValue("MyCheckBox") = 1 Then
    DlgValue "MyCheckBox",0
Else
    DlgValue "MyCheckBox",1
End If
```

See Also

DlgVisible   Function

Description: Returns True if the specified control is visible; returns False otherwise.

Syntax: DlgVisible(ControlName$ | ControlIndex)

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the template (0 is the first control in the template, 1 is the second, and so on).

A runtime error is generated if DlgVisible is called with no user dialog is active.

Example: If **DlgVisible**("Portrait") Then Beep

If **DlgVisible**(10) And **DlgVisible**(12) Then
    MsgBox "The 10th and 12th controls are visible."
End If

See Also

DlgVisible   **Statement**

Description: Hides or shows the specified control.

Syntax: DlgVisible {*ControlName$* | *ControlIndex*} [,*isOn*]

Comments: Hidden controls cannot be seen in the dialog box and cannot receive the focus using Tab.

The *isOn* parameter is an Integer specifying the new state of the control. It can be any of the following values:

1   The control is shown.

0   The control is hidden.

Omitted   Toggles the visibility of the control.

Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).

The *ControlName$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Picture Caching

When the dialog box is first created and before it is shown, Symantec Basic calls the dialog function with *action* set to 1. At this time, no pictures have been loaded into the picture controls contained in the dialog box template. After control returns from the dialog function and before the dialog box is shown, Symantec Basic will load the pictures of all visible picture controls. Thus, it is possible for the dialog function to hide certain picture controls, which prevents the associated pictures from being loaded and causes the dialog box to load faster. When a picture control is made visible for the first time, the associated picture will then be loaded.

Example: This example creates a dialog box with two panels. The DlgVisible statement is used to show or hide the controls of the different panels.

```
Sub EnableGroup(Start%, Finish%)
    For i = 6 To 13                             'Disable all
options.
        DlgVisible I, False
    Next i
    For i = Start% To Finish%                   'Enable only the
right ones.
        DlgVisible I, True
    Next i
End Sub
```

```
Function DlgProc(ControlName$, Action%, SuppValue%)
    If Action = 1 Then
        DlgValue "WhichOptions",0         'Set to save options.
        EnableGroup 6, 8                          'Enable the save options.
    End If
    If Action = 2 And ControlName$ = "SaveOptions" Then
        EnableGroup 6, 8                          'Enable the save options.
        DlgProc = 1                                       'Don't close the
dialog
    End If
    If Action = 2 And ControlName$ = "EditingOptions" Then
        EnableGroup 9, 13                             'Enable the editing
options.
        DlgProc = 1                                       'Don't close the
dialog
    End If
End Function

Sub Main()
    Begin Dialog OptionsTemplate 33, 33, 171, 134, "Options", .DlgProc
        'Background (controls 0-5)
        GroupBox 8, 40, 152, 84, ""
        OptionGroup .WhichOptions
                OptionButton 8, 8, 59, 8, "Save Options",.SaveOptions
                OptionButton 8, 20, 65, 8, "Editing Options",.EditingOptions
        OKButton 116, 7, 44, 14
        CancelButton 116, 24, 44, 14

        'Save options (controls 6-8)
        CheckBox 20, 56, 88, 8, "Always create backup",.CheckBox1
        CheckBox 20, 68, 65, 8, "Automatic save",.CheckBox2
        CheckBox 20, 80, 70, 8, "Allow overwriting",.CheckBox3

        'Editing options (controls 9-13)
        CheckBox 20, 56, 65, 8, "Overtype mode",.OvertypeMode
        CheckBox 20, 68, 69, 8, "Uppercase only",.UppercaseOnly
        CheckBox 20, 80, 105, 8, "Automatically check
syntax",.AutoCheckSyntax
        CheckBox 20, 92, 73, 8, "Full line selection",.FullLineSelection
        CheckBox 20, 104, 102, 8, "Typing replaces
selection",.TypingReplacesText
    End Dialog

    Dim OptionsDialog As OptionsTemplate
    Dialog OptionsDialog
End Sub
```

See Also

DropListBox   Statement

Description:	Creates a drop list box within a dialog box template.
Syntax:	DropListBox X, Y, width, height, ArrayVariable, .Identifier
Comments:	When the dialog box is invoked, the drop list box will be filled with the elements contained in *ArrayVariable*. Drop list boxes are similar to combo boxes, with the following exceptions:

The list box portion of a drop list box is not opened by default. The user must open it by clicking the down arrow.

The user cannot type into a drop list box. Only items from the list box may be selected. With combo boxes, the user can type the name of an item from the list directly or type the name of an item that is not contained within the combo box.

This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).

The DropListBox statement requires the following parameters:

Parameter	Description
X, *Y*	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, *height*	Integer coordinates specifying the dimensions of the control in dialog units.
ArrayVariable	Single-dimensioned array used to initialize the elements of the drop list box. If this array has no dimensions, then the drop list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension.
	ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates

an integer variable whose value corresponds to the index of the drop list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax:

DialogVariable.Identifier

Example: This example allows the user to choose a field name from a drop list box.

```
Sub Main()
    Dim FieldNames$(4)
    FieldNames$(0) = "Last Name"
    FieldNames$(1) = "First Name"
    FieldNames$(2) = "Zip Code"
    FieldNames$(3) = "State"
    FieldNames$(4) = "City"
    Begin Dialog FindTemplate 16,32,168,48,"Find"
        Text 8,8,37,8,"&Find what:"
        DropListBox 48,6,64,80,FieldNames,.WhichField
        OKButton 120,7,40,14
        CancelButton 120,27,40,14
    End Dialog
    Dim FindDialog As FindTemplate
    FindDialog.WhichField = 1
    Dialog FindDialog
End Sub
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog](#)

[Dialog()](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[Picture](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog . . .End Dialog](#)

[PictureButton](#)

DoEvents Statement

Description: This statement yields control to other applications. When running in exclusive mode as a result of using the Exclusive statement, it allows Windows to perform a round of multitasking. When the script gets its turn to execute, it returns to exclusive mode.

This statement does nothing under Chicago and NT, since multitasking is preemptive on those systems.

Syntax: DoEvents

Example: The following example uses the DoEvents statement to stop between iterations and allow Windows to execute other applications. It assumes you are looping through iterations of a time-consuming computation.

```
Exclusive TRUE      'Enter exclusive mode
For i = 1 To 100
    ...
    DoEvents
    'Gives other applications a chance
    'to be processed
Next i
Exclusive FALSE     'Leave exclusive mode
```

See Also

DoEvents()

DoEvents( )   **Function**

Description: This function yields control to other applications. When running in exclusive mode as a result of using the Exclusive statement, it allows Windows to perform a round of multitasking. The function returns 0 when Windows next allows the script to execute. At that time, the script returns to exclusive mode.

This statement does nothing under Chicago and NT, since multitasking is preemptive on those systems.

Syntax: DoEvents[( )]

Example: The example checks to see if multitasking has taken place.

```
x = 5
If someCondition Then
    x = DoEvents
End If
...
If x = 0 Then
    'Multitasking occurred so ...
    ...
End If
```

See Also

DoEvents

DoKeys   **Statement**

Description:	This statement is generated by the Recorder as an optimized way to send keystrokes to applications. It is generated when there are no mouse movements and partial keystrokes in the event queue.
Syntax:	DoKeys keyStr [, timeout]
Parameters:	keyStr
	A string expression specifying one or more full keystrokes.
	timeout
	The time in milliseconds to wait for the keystroke to be sent. The default is zero (one attempt).
Example:	The following examples sends ten copies of the letter a to the active application.

```
DoKeys "{a 10}"
```

See Also

SendKeys

Double   **Data type**

Syntax:	Double
Description:	A data type used to declare variables capable of holding real numbers with 15Ð16 digits of precision.
Comments:	Double variables are used to hold numbers within the following ranges:

Sign	Range
Negative	–1.797693134862315E308 <= *double* <= -4.94066E-324
Positive	4.94066E-324 <= *double* <= 1.797693134862315E308

The type-declaration character for Double is #.

Storage

Internally, doubles are 8-byte (64-bit) IEEE values. Thus, when appearing within a structure, doubles require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

Each Double consists of the following

    A 1-bit sign

    An 11-bit exponent

    A 53-bit significand (mantissa)

Platform(s):	All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CDbl( )](#)

ebAbort Constant

Description: Returned by the MsgBox function when the Abort button is chosen.

Comments: This constant is equal to 3.

Example: This example displays a dialog box with Abort, Retry, and Ignore buttons.

```
Sub Main()
rc% = MsgBox("Do you want to continue?",ebAbortRetryIgnore)
If rc = ebAbort Then
MsgBox Str$(ebAbort)
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebAbortRetryIgnore   Constant

Description: Used by the MsgBox statement and function.

Comments: This constant is equal to 2.

Example: This example displays a dialog box with Abort, Retry, and Ignore buttons.

```
Sub Main()
rc% = MsgBox("Wicked disk error!",ebAbortRetryIgnore)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebApplicationModal   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 0.

Example: This example displays an application-modal dialog box (which is the default).

```
Sub Main()
MsgBox "This is application-modal.",ebOKOnly Or ebApplicationModal
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebArchive   Constant

Description: This numeric constant represents the bit position of a file attribute for checking the archive flag of a file. If a file has not been backed up, this bit position is set. The constant's value is 32. It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebArchive

Example: The following example retrieves the names of all the files in the current directory whose archive attribute is set, as well as the names of normal files. The retrieved files are shown in a predefined dialog box that displays a list box.

```
Dim archiveNames$(1 To 100)
FileList archiveNames, "*.*", ebArchive
selectedFile = SelectBox ("Archive Files", "Select a File", archiveNames)
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebBold   **Constant**

Description: Used with the Text and TextBox statement to specify a bold font.

Comments: This constant is equal to 2.

Example:

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Bold Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBold
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBold
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

Platform(s):   Windows, Win32, OS/2, Macintosh.

See Also

[Text](#)
[TextBox](#)

ebBoldItalic   **Constant**

Description: Used with the Text and TextBox statement to specify a bold-italic font.

Comments: This constant is equal to 6.

Example:

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Bold-Italic Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBoldItalic
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBoldItalic
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

Platform(s):   Windows, Win32, OS/2, Macintosh.

See Also

[Text](#)
[TextBox](#)

ebBoolean   Constant

Description:	Number representing the type of a Boolean variant.
Comments:	This constant is equal to 11.
Example:	

```
Sub Main()
    Dim MyVariant as variant
    MyVariant = True
    If VarType(MyVariant) = ebBoolean Then
        MyVariant = 5.5
    End If
End Sub
```

Platform(s):	All.

See Also

VarType (function)
Variant (data type)

ebCancel Constant

Description: Returned by the MsgBox function when the Cancel button is chosen.

Comments: This constant is equal to 2.

```
Example:    Sub Main()
'Invoke MsgBox and check whether the Cancel button was pressed.
rc% = MsgBox("Are you sure you want to quit?",ebOKCancel)
If rc% = ebCancel Then
    'The user selected Cancel from the dialog box.
    MsgBox "The user clicked Cancel."
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebCritical Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 16.

Example:
```
Sub Main()
'Invoke MsgBox with Abort, Retry, and Ignore buttons and a Stop icon.
rc% = MsgBox("Disk drive door is open.",ebAbortRetryIgnore Or
ebCritical)
 If rc% = 3 Then
    'The user selected Abort from the dialog box.
    MsgBox "The user clicked Abort."
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebCurrency   **Constant**

Description: Number representing the type of a Currency variant.

Comments: This constant is equal to 6.

Example:

```
'This example checks to see whether a variant is of type Currency.

Sub Main()
   Dim MyVariant
   If VarType(MyVariant) = ebCurrency Then
      MsgBox "Variant is Currency."
   End If
End Sub
```

Platform(s): All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebDataObject

Constant

Description: Number representing the type of a data object variant.

Comments: This constant is equal to 13.

Example:

```
'This example checks to see whether a variable is a data object.

Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebDataObject Then
       MsgBox "Variant contains a data object."
    End If
End Sub
```

Platform(s):   All.

See Also

VarType (function)
Variant (data type)

ebError

Constant

Description: Number representing the type of an error variant.

Comments: This constant is equal to 10.

Example:

```
'This example checks to see whether a variable is an error.

Function Div(ByVal a As Variant,ByVal b As Variant) As Variant
    On Error Resume Next
    Div = a / b
    If Err <> 0 Then Div = CVErr(Err)
End Function

Sub Main()
    a = InputBox("Please enter 1st number","Division Sample")
    b = InputBox("Please enter 2nd number","Division Sample")

    res = Div(a,b)

    If VarType(res) = ebError Then
       res = CStr(res)
       res = Error(Mid(res,7,Len(res)))
       MsgBox "'" & res & "' occurred"
    Else
       MsgBox "The result of the division is: " & res
    End If
End Sub
```

Platform(s): All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebDate   **Constant**

Description: Number representing the type of a Date variant.

Comments: This constant is equal to 7.

Example:

```
Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebDate Then
       MsgBox "This variable is a Date type!"
    Else
       MsgBox "This variable is not a Date type!"
    End If
End Sub
```

Platform(s):   All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebDefaultButton1   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 0.

Example: This example invokes MsgBox with the focus on the OK button by default.

```
Sub Main()
rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or
ebDefaultButton1)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebDefaultButton2   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 256.

Example: This example invokes MsgBox with the focus on the Cancel button by default.

```
Sub Main()
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or
ebDefaultButton2)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebDefaultButton3   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 512.

Example: This example invokes MsgBox with the focus on the Ignore button by default.

```
Sub Main()
    rc% = MsgBox("Disk drive door open.",ebAbortRetryIgnore Or
ebDefaultButton3)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebDirectory   **Constant**

Description: This numeric constant represents the bit position of a file attribute indicating that a file is a directory entry. The constant's value is 16.
It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebDirectory

Example: The following example retrieves the names of all the directory names in the current directory as well as the names of normal files:

```
Dim directoryNames$(1 To 100)
FileList directoryNames, "*.*", ebDirectory
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebDOS   **Constant**

Description: This numeric constant is used with the FileType( ) and AppType( ) functions to indicate a DOS application. Its value is 1.

Syntax: ebDOS

Example: The following example determines the type of the file named TESTFILE.

```
theType% = FileType("SYSINFO.EXE")

If theType = ebDOS Then
    ...'It is a DOS executable file.
ElseIf theType = ebWindows Then
    ...'It is a Windows executable file.
Else
    ...'The file type is unknown.
End If
```

See Also

[FileType()](#)

ebWin16   **Constant**

Description: Used with the Basic.OS property to indicate the 16-bit Windows version of BasicScript.

Comments: This constant is equal to 0.

                   The Basic.OS property returns this value when BasicScript is running under the Windows 3.1 operating system

Example:

```
Sub Main()
    If Basic.OS = ebWin16 Then MsgBox "Running under Windows 3.1."
End Sub
```

Platform(s):   All.

See Also

Basic.OS

ebWin32   **Constant**

Description: Used with the Basic.OS property to indicate the 32-bit Windows version of BasicScript.

Comments: This constant is equal to 2.

The Basic.OS property returns this value when running under any of the following operating systems:

Microsoft Windows 95

Microsoft Windows NT Workstation
(Intel, Alpha, MIPS, PowerPC)

Microsoft Windows NT Server
(Intel, Alpha, MIPS, PowerPC)

Microsoft Win32s running under Windows 3.1

Example:

```
Sub Main()
    If Basic.OS = ebWin32 Then MsgBox "Running under Win32."
End Sub
```

Platform(s):   All.

See Also

Basic.OS

ebDOS16   **Constant**

Description:	Used with the Basic.OS property to indicate the 16-bit DOS version of BasicScript.
Comments:	This constant is equal to 1.
Example:	

```
Sub Main()
    If Basic.OS = ebDOS16 Then MsgBox "Running under 16-bit DOS."
End Sub
```

Platform(s):	All.

See Also

Basic.OS

ebDOS32   **Constant**

Description: Used with the Basic.OS property to indicate the 32-bit DOS version of BasicScript.

Comments: This constant is equal to 12.

Example:

```
Sub Main()
    If Basic.OS = ebDOS32 Then MsgBox "Running under 32-bit DOS."
End Sub
```

Platform(s):   All.

See Also

[Basic.OS](#)

ebDouble   Constant

Description: Number representing the type of a Double variant.

Comments: This constant is equal to 5.

Example: See ebSingle (constant).

Platform(s):   All.

See Also

MsgBox
MsgBox( )
VarType (function)
Variant (data type)

ebEmpty

Constant

Description: Number representing the type of an Empty variant.

Comments: This constant is equal to 0.

Example:

```
Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebEmpty Then
        MsgBox "This variant has not been assigned a value yet!"
    End If
End Sub
```

Platform(s): All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebExclamation Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 48.

Example: This example displays a dialog box with an OK button and an exclamation icon.

```
Sub Main()
MsgBox "Out of memory saving to disk.",ebOKOnly Or ebExclamation
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebHidden   **Constant**

Description: This numeric constant represents the bit position of a file attribute indicating that a file is hidden. The constant's value is 2. It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebHidden

Example: The following example retrieves the names of all the hidden files in the current directory, as well as the names of normal files.

```
Dim hiddenNames$(1 To 100)
FileList hiddenNames, "*.*", ebHidden
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebIgnore   **Constant**

Description: Returned by the MsgBox function when the Ignore button is chosen.

Comments: This constant is equal to 5.

Example: This example displays a critical error dialog box and sees what the user wants to do.

```
Sub Main()
rc% = MsgBox("Printer out of paper.",ebAbortRetryIgnore)
If rc% = ebIgnore Then
    'Continue printing here.
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebInformation   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 64.

Example: This example displays a dialog box with the Information icon.

```
Sub Main()
MsgBox "For your information, you just deleted your file!",ebOKOnly Or
ebInformation
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebInteger   Constant

Description: Number representing the type of an Integer variant.

Comments: This constant is equal to 2.

Example:

```
'This example defines a function that returns True if a variant
'contains an Integer value (either a 16-bit or 32-bit Integer).

Function IsInteger(v As Variant) As Boolean
    If VarType(v) = ebInteger Or VarType(v) = ebLong Then
       IsInteger = True
    Else
       IsInteger = False
    End If
End Function

Sub Main()
    Dim i as Integer
    i = 123
    If IsInteger(i) then
       Msgbox "i is an Integer."
    End If
End Sub
```

Platform(s):   All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebItalic   **Constant**

Description: Used with the Text and TextBox statement to specify an italic font.

Comments: This constant is equal to 4.

Example:

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Italic Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebItalic
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebItalic
        OKButton 96,110,40,14
    End Dialog

    Dim a As UserDialog
    Dialog a
End Sub
```

Platform(s):   Windows, Win32, OS/2, Macintosh.

See Also

[Text](#)
[TextBox](#)

ebLandscape   **Constant**

Description: This numeric constant sets the orientation of printed output to landscape. Its value is 2. Use this function with the PrinterSetOrientation statement to align the paper horizontally.

The PrinterGetOrientation( ) function returns this value to indicate that the page orientation is landscape.

Syntax: ebLandscape

Example: The following example asks the user for a page orientation and adjusts the printer accordingly.

```
If AnswerBox("Orientation?", "Portrait", "Landscape") = 1 Then
    PrinterSetOrientation ebPortrait
Else
    PrinterSetOrientation ebLandscape
End If
```

See Also

[PrinterSetOrientation](#)
[PrinterGetOrientation()](#)

ebLeftButton   Constant

Description: This numeric constant represents the left mouse button in QueMouse commands. Its value is 1.

Syntax: ebLeftButton

Example: The following example simulates a mouse click using the left mouse button.

```
'Left mouse button click at (x=167, y=205)
QueMouseClick ebLeftButton, 167, 205
'Play the click
QueFlush TRUE
```

ebLong   **Constant**

Description:	Number representing the type of a Long variant.
Comments:	This constant is equal to 3.
Example:	See ebInteger (constant).
Platform(s):	All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebNo   **Constant**

Description: Returned by the MsgBox function when the No button is chosen.

Comments: This constant is equal to 7.

Example: This example asks a question and queries the user's response.

```
Sub Main()
rc% = MsgBox("Do you want to update the glossary?",ebYesNo)
    If rc% = ebNo Then
        MsgBox "The user clicked the No button."        'Do not update the
glossary.
    End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebNone   **Constant**

Description: Bit value used to select files with no other attributes.

Comments: This value can be used with the Dir$ and FileList commands. These functions will return only files with no attributes set when used with this constant. This constant is equal to 7.

Example: This example dimensions an array and fills it with filenames with no attributes set.

```
Sub Main()
    Dim s$()
    FileList S$, "*.*", ebNone
    a% = SelectBox("No Attributes", "Choose one", S$)
    If a >= 0 Then
       MsgBox "You selected file " + S$(a)
    Else
       MsgBox "No selection made."
    End If
End Sub
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebNormal   **Constant**

Description: This numeric constant represents the bit position of a file attribute indicating that a file has no other attributes set. The constant's value is 0. It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebNormal

Example: The following example retrieves the names of all the normal files in the current directory.

```
Dim normalNames$(1 To 100)
FileList normalNames, "*.*", ebNormal
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebNull   **Constant**

Description: Number representing the type of a Null variant.

Comments: This constant is equal to 1.

Example:

```
Sub Main()
    Dim MyVariant
    MyVariant = Null
    If VarType(MyVariant) = ebNull Then
        MsgBox "This variant is Null"
    End If
End Sub
```

Platform(s):   All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebObject   **Constant**

Description: Number representing the type of an Object variant (an OLE automation object).

Comments: This constant is equal to 9.

Example:

```
Sub Main()
    Dim MyVariant
    If VarType(MyVariant) = ebObject Then
      MsgBox MyVariant.Value
    Else
      MsgBox "'MyVariant' is not an object."
    End If
End Sub
```

Platform(s):   All.

See Also

VarType (function)
Variant (data type)

ebOK   **Constant**

Description: Returned by the MsgBox function when the OK button is chosen.

Comments: This constant is equal to 1.

Example: This example displays a dialog box that allows the user to cancel.

```
Sub Main()
rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)
    If rc% = ebOK Then System.Exit
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebOKCancel Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 1.

Example: This example displays a dialog box that allows the user to cancel.

```
Sub Main()
rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)
If rc% = ebOK Then System.Exit
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebOKOnly   Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 0.

Example: This example informs the user of what is going on (no options).

```
Sub Main()
MsgBox "Windows is now shutting down.",ebOKOnly
End Sub
```

See Also

MsgBox()
MsgBox

ebPortrait

Constant

Description: This numeric constant sets the orientation of printed output to portrait. Its value is 1. Use this function with the PrinterSetOrientation statement to align the paper vertically. The PrinterGetOrientation( ) function returns this value to indicate that the page orientation is portrait.

Syntax: ebPortrait

Example: The following example determines whether the current page orientation is landscape or portrait.

```
If PrinterGetOrientation( ) = ebLandscape Then
    MsgBox "Landscape"
Else
    MsgBox "Portrait"
End If
```

See Also

[PrinterSetOrientation](#)
[PrinterGetOrientation()](#)

ebQuestion   **Constant**

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 32.

Example: This example displays a dialog box with OK and Cancel buttons and a question icon.

```
Sub Main()
rc% = MsgBox("OK to delete file?",ebOKCancel Or ebQuestion)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebReadOnly   Constant

Description: This numeric constant represents the bit position of a file attribute indicating that a file is read-only. The constant's value is 1. It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebReadOnly

Example: The following example retrieves the names of all the names of read-only files in the current directory, as well as the names of normal files.

```
Dim readonlyNames$(1 To 100)
FileList readonlyNames, "*.*", ebReadOnly
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebRegular   Constant

Description: Used with the Text and TextBox statement to specify an normal-styled font (i.e., neither bold or italic).

Comments: This constant is equal to 1.

Example:

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Regular Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebRegular
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebRegular
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

Platform(s):   Windows, Win32, OS/2, Macintosh.

See Also

[Text](#)
[TextBox](#)

ebRetry   Constant

Description: Returned by the MsgBox function when the Retry button is chosen.

Comments: This constant is equal to 4.

Example: This example displays a Retry message box.

```
Sub Main()
rc% = MsgBox("Unable to open file.",ebRetryCancel)
If rc% = ebRetry Then
    MsgBox("User selected Retry.")
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebRetryCancel Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 5.

Example: This example invokes a dialog box with Retry and Cancel buttons.

```
Sub Main()
rc% = MsgBox("Unable to open file.",ebRetryCancel)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebRightButton Constant

Description: This numeric constant represents the right mouse button in QueMouse commands. Its value is 2.

Syntax: ebRightButton

Example: The following example performs a double-click using the right mouse button.

```
'Right mouse double-click at (x=100, y=101)
QueMouseDblClk ebRightButton, 100, 101
'Play the double-click
QueFlush TRUE
```

ebSingle   **Constant**

Description: Number representing the type of a Single variant.

Comments: This constant is equal to 4.

Example:

```
'This example defines a function that returns True if the passed
'variant is a Real number.

Function IsReal(v As Variant) As Boolean
    If VarType(v) = ebSingle Or VarType(v) = ebDouble Then
       IsReal = True
    Else
       IsReal = False
    End If
End Function

Sub Main()
    Dim i as Integer
    i = 123
    If IsReal(i) then
       Msgbox "i is Real."
    End If
End Sub
```

Platform(s):   All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebString   **Constant**

Description:	Number representing the type of a String variant.
Comments:	This constant is equal to 8.
Example:	

```
Sub Main()
    Dim MyVariant as variant
    MyVariant = "This is a test."
    If VarType(MyVariant) = ebString Then
        MsgBox "Variant is a string."
    End If
End Sub
```

Platform(s):	All.

See Also

VarType (function)
Variant (data type)

ebSystem

Constant

Description: This numeric constant represents the bit position of a file attribute indicating that a file is a system file. The constant's value is 4. It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebSystem

Example: The following example retrieves the names of all the system files in the current directory, as well as the names of normal files.

```
Dim systemNames$(1 To 100)
FileList systemNames, "*.*", ebSystem
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)
[FileAttr()](#)

ebSystemModal Constant

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 4096.

Example:
```
Sub Main()
MsgBox "A very bad thing occurred. All applications are
paused!",ebSystemModal
End Sub
```

See Also

[ebApplicationModal](#)
[MsgBox()](#)
[MsgBox](#)

ebVariant   **Constant**

Description:	Number representing the type of a Variant.
Comments:	Currently, it is not possible for variants to use this subtype. This constant is equal to 12.
Platform(s):	All.

See Also

[VarType (function)](#)
[Variant (data type)](#)

ebVolume

Constant

Description: This numeric constant represents the bit position of a file attribute indicating that a file is the volume label. The constant's value is 8.
It is used with the FileList and SetAttr statements and the GetAttr( ) function.

Syntax: ebVolume

Example: The following example retrieves the name of the volume label, as well as the names of normal files.

```
Dim volumeNames$(1 To 100)
FileList volumeNames, "*.*", ebVolume
```

See Also

[Dir$()](#)
[FileList](#)
[SetAttr](#)
[GetAttr()](#)

ebWindows   Constant

Description: This numeric constant is used with the FileType( ) and AppType( ) functions to indicate a Windows application. Its value is 2.

Syntax: ebWindows

Example: The following example determines the platform on which the current application runs and displays the result in a message box.

```
If AppType = ebDOS Then

    MsgBox "Current application is a DOS application."
Else
    MsgBox "Current application is a Windows application."
End If
```

ebYes   **Constant**

Description: Returned by the MsgBox function when the Yes button is chosen.

Comments: This constant is equal to 6.

Example: This example queries the user for a response.

```
Sub Main()
rc% = MsgBox("Overwrite file?",ebYesNoCancel)
    If rc% = ebYes Then
        MsgBox "You elected to overwrite the file."
    End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebYesNo   **Constant**

Description: Used with the MsgBox statement and function.

Comments: This constant is equal to 4.

Example: This example displays a dialog box with Yes and No buttons.

```
Sub Main()
rc% = MsgBox("Are you sure you want to remove all formatting?",ebYesNo)
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

ebYesNoCancel Constant

Description:	Used with the MsgBox statement and function.
Comments:	This constant is equal to 3.
Example:	This example displays a dialog box with Yes, No, and Cancel buttons.

```
Sub Main()
rc% = MsgBox("Format drive C?",ebYesNoCancel)
If rc = ebYes Then
    MsgBox "User chose Yes."
End If
End Sub
```

See Also

[MsgBox()](#)
[MsgBox](#)

Empty   **Constant**

Description: Constant representing a variant of type 0.

Comments: The Empty value has special meaning indicating that a Variant is uninitialized.

When Empty is assigned to numbers, the value 0 is assigned. When Empty is assigned to a String, the string is assigned a zero-length string.

Example:

```
Sub Main()
    Dim a As Variant
    a = Empty
    MsgBox "This string is" & a & "concatenated with Empty"
    MsgBox "5 + Empty = " & (5 + a)
End Sub
```

Platform(s):   All.

See Also

Null (constant)
Variant (data type)
VarType (function)

End   **Statement**

Description: This statement ends the execution of the current script and closes all open files and DDE channels.

Syntax: End

Example: In the following example, the user enters a password. If after three attempts, the password has not been entered correctly, the whole script terminates using the End statement.

```
i% = 0
Do
    s$ = AskPassword$("Type in the password:")
    If s$ = "password" Then
        Exit Do
    End If
    i = i + 1
    If i = 3 Then
        End
    End If
Loop
```

See Also

Close
Stop
Exit For
Exit Do
Exit Function
Exit Sub

Environ$( )   Function

Description: This function returns a string expression containing the environment variable and its value in the form: variable = value. If the variable name is not found, the function returns an empty string.

Syntax: Environ$(var|varNum)

Parameters: var

A string expression containing the name of the environment variable.

varNum

A numeric expression containing the position of the environment variable in the environment variable table. It is rounded to an integer before it is used. The first variable is at position 1.

Example: The following example uses the Environ$( ) function to process each of the environment variables present in the environment.

```
'Initialize the count to 1 for the first variable
count% = 1

'Get the first environment variable
envVar$ = Environ$(count)

'Check if the last variable has already been found
While envVar <> ""
    . . . 'Process the variable and its value

    'Increment the count
    count = count + 1
    'Get the next environment variable
    envVar = Environ$(count)
Wend
```

See Also

Command$()

EOF( )   **Function**

Description: This function returns TRUE if the end of file has been reached, or FALSE if the end of file has not been reached.

Syntax: EOF(fileNum)

Parameter: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

Example: The file NAMEFILE contains a list of names as follows:

```
"John","Jane","Smith","Mary"
    The following example reads the names from the file into the string
array NAMES and stops when the end of file has been reached.
Dim NAMES(1 to 100) As String

Open "namefile" For Input As #5
i% = 1

'While the end of the file has not been reached
While not EOF(5)
    'Read in a name
    Input #5, NAMES(i)
    'Increment the index
    i = i + 1
Wend
Close #5
```

See Also

[Open](#)
[Lof()](#)

Eqv   **Operator**

Description: Performs a logical or binary equivalence on two expressions.

Syntax: expression1 Eqv expression2

Comments: If both expressions are either Boolean, Boolean variants, or Null variants, then a logical
                    equivalence is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
False	True	False
False	False	True

If either expression is Null, then Null is returned.

Binary Equivalence

If the two expressions are Integer, then a binary equivalence is performed, returning an Integer
result. All other numeric types (including Empty variants) are converted to Long and a binary
equivalence is then performed, returning a Long result.

Binary equivalence forms a new value based on a bit-by-bit comparison of the binary
representations of the two expressions, according to the following table:

```
1     Eqv   1     =     1           Example:
0     Eqv   1     =     0           5      01101001
1     Eqv   0     =     0           6      10101010
0     Eqv   0     =     1           Eqv    00101000
```

Example: This example assigns False to A, performs some equivalent operations, and displays a dialog box
                    with the result. Since A is equivalent to False, and False is equivalent to 0, and by definition, A =
                    0, then the dialog box will display "A is False."

```
Sub Main()
    A = False
    If ((A Eqv False) And (False Eqv 0) And (A = 0)) Then
        MsgBox "A is False."
    Else
        MsgBox "A is True."
    End If
End Sub
```

See Also

[Or](#)
[XOR](#)
[Imp](#)
[And](#)

Erase   **Statement**

Description: Erases the elements of the specified arrays.

Syntax: Erase array1 [,array2]...

Comments: For dynamic arrays, the elements are erased, and the array is redimensioned to have no dimensions (and therefore no elements). For fixed arrays, only the elements are erased; the array dimensions are not changed.

After a dynamic array is erased, the array will contain no elements and no dimensions. Thus, before the array can be used by your program, the dimensions must be reestablished using the Redim statement.

Up to 32 parameters can be specified with the Erase statement.

The meaning of erasing an array element depends on the type of the element being erased:

Element Type	What Erase Does to That Element
Integer	Sets the element to 0.
Boolean	Sets the element to False.
Long	Sets the element to 0.
Double	Sets the element to 0.0.
Date	Sets the element to December 31, 1899.
Single	Sets the element to 0.0.
String (variable-length)	Frees the string, then sets the element to a zero-length string.
String (fixed-length)	Sets every character of each element to zero (Chr$(0)).
Object	Decrements the reference count and sets the element to Nothing.
Variant	Sets the element to Empty.
User-defined type	Sets each structure element as a separate variable.

Example: This example puts a value into an array and displays it. Then it erases the value and displays it again.

```
Sub Main()
    Dim a$(10)                          'Declare an array.
    a$(1) = Dir$("*")           'Fill element 1 with a filename.
    MsgBox a$(1)                        'Display element 1.
    Erase a                             'Erase all elements in the array.
    MsgBox a$(1)                        'Display element 1 again (should
be erased).
End Sub
```

See Also

[Redim](#)

Erl( )   **Function**

Description: This function always returns a 0. It is included for compatibility with other BASICs.

Syntax: Erl[( )]

See Also

[Err()](Err())
Error$()

Err   **Statement**

Description: This statement sets the value returned by the Err( ) function.

Syntax: Err = errorNum

Parameter: errorNum

                An integer representing an error number.

Example: The following example displays the messages for errors whose numbers range from 300 to 325.

```
...
For counter% = 300 To 325
    Err = counter
    MsgBox Error$
Next counter
```

See Also

[Error](#)

Err( )   **Function**

Description: This function returns the error number, an integer, for the most recently trapped error. It can be used only while an On Error statement is valid. The value of Err( ) is 0 when the script starts. It is reset to 0 by the Resume statement and when a subroutine or function ends.

Syntax: Err[( )]

Example: The following example sends all errors to the same label. The statements between the label and the Resume statement display the error numbers and messages that Symantec Basic normally displays when a run-time error terminates a script. The Error$( ) function returns the error message associated with the most recent error.

```
Sub Main ( )
On Error GoTo MessageDisplay
...
cmd$ = "[CreateGroup(" + quoted(Setup.GroupName) + ")]"
DDEExecute channel, cmd$
...
Exit Sub
'This routine can be used for all errors
'while you are debugging
'It may help you fix more than one error
' at a time
MessageDisplay:
    MsgBox Str$(Err( )) + Error$( )
    Resume Next
End Sub
```

This can be a handy debugging tool. However, the message you get when a run-time error stops the script's execution can have more information, such as the line number of the statement causing the error and the number of an array subscript that is causing a problem.

See Also

[Erl()](Erl())
Error$()

Error   **Statement**

Description: This statement simulates a Symantec Basic run-time error or a user-defined error. If no error handling routine exists, this statement generates an error message and stops the script's execution.

Syntax: Error errorNum

Parameter: errorNum

An error number that is built into the Symantec Basic language or assigned by the user with the Err statement.

Example: When errors are not being trapped, the following example stops the execution of the script and causes its error message to be displayed on the screen.

```
Error 6
'External function/subroutine does not exist
```

See Also

[Err](#)

Error$( )   Function

Description: This function returns a string containing the error message for the specified error number or for the most recently trapped error. It returns the empty string if no error has occurred. It returns "Unknown or user error code" if the error statement is user-defined.

Syntax: Error$[(errorNum)]

Parameter: errorNum

Any error number. The default is the most recently trapped error.

Example: The following example displays the error message associated with the most recent error.

```
MsgBox Error$( )
```

See Also

[Erl()](#)
[Err()](#)

Exclusive   Statement

Description: This statement turns exclusive mode on and off. When state is TRUE, the statement stops Windows from multitasking applications. Exclusive mode runs scripts faster than in multitasking mode but other applications cannot execute. Use DoEvents to interrupt exclusive mode for a round of multitasking.

This statement does nothing under Chicago and NT, since multitasking is preemptive on those systems.

Syntax: Exclusive state

Parameter: state

A numeric expression that can be TRUE or FALSE.

Example: The following example shows how to use exclusive mode while doing a time-consuming computation that you don't want interrupted by multi-tasking.

```
Exclusive TRUE                  'Enter exclusive mode
    ... 'A time-consuming computation
Exclusive FALSE            'Leave exclusive mode
```

Exit Do   **Statement**

Description: This statement terminates a Do loop. It transfers control to the statement immediately after the Loop statement.

Syntax: Exit Do

Example: The following example shows a Do loop with an Exit Do statement.

```
Do
    'sequence of statements
    If Err ( ) = BigProblem
        Exit Do
    End If
    'sequence of statements
Loop While JobNotDone
```

See Also

[Stop](#)
[Exit For](#)
[Exit Function](#)
[Exit Sub](#)
[End](#)
[Do . . .Loop](#)

Exit For

Statement

Description: This statement terminates a For...Next loop. It transfers control to the line immediately after the Next statement.

Syntax: Exit For

Example: The following example shows how a For loop is ended from a point within the loop.

```
For counter = 1 To 25
    ...
    If some_condition Then
       Exit For
    End If
    ...
Next
```

See Also

[Stop](#)
[Exit Do](#)
[Exit Function](#)
[Exit Sub](#)
[End](#)
[For . . .Next](#)

Exit Function   Statement

Description: This statement terminates the execution of the function in which the statement occurs. Control transfers to the statement that called the function.

Syntax: Exit Function

Example: The following example computes a factorial. If the parameter is negative, Exit Function is used to end the function.

```
Function Factorial(n%)
    If n < 0 Then
        Exit Function
    End If

    result% = 1

    For i = 1 To n
        result = result * i
    Next

    Factorial = result
End Function
```

See Also

[Stop](#)
[Exit For](#)
[Exit Do](#)
[Exit Sub](#)
[End](#)
[Function . . .End Function](#)

Exit Sub   Statement

Description: This statement terminates a subroutine's execution. Control transfers to the statement following the call to the subroutine.

Syntax: Exit Sub

Example: The following subroutine computes a factorial. If the first parameter is negative, Exit Sub is used to end the subroutine.

```
Sub Factorial(n%, result%)
    If n < 0 Then
        Exit Sub
    End If

    result = 1

    For i = 1 To n
        result = result * i
    Next
End Sub
```

See Also

[Stop](#)
[Exit For](#)
[Exit Do](#)
[Exit Function](#)
[End](#)
[Sub . . .End Sub](#)

Exp( )   **Function**

Description:	This function returns the value of e raised to the power of x. An overflow error occurs if x is out of the specified range. The value returned is a number of type double.
Syntax:	Exp(x)
Parameters:	x
	A numeric expression in the range from 0 to 709.782712893.
Example:	The following example calculates the value of the base e by raising it to the first power using the Exp( ) function.

```
e# = Exp(1)
```

See Also

[Log()](Log())

FALSE   **Constant**

Description: This numeric constant is used in logical expressions and as the value of some parameters. It can be assigned to numeric variables so that the variables can be used in logical expressions. Its value is 0.

Syntax: FALSE

Example: The following example returns the value FALSE if a specified integer is not odd. Otherwise, the function returns the value TRUE.

```
Function Odd(n As Integer)
    If (n MOD 2) = 1 Then
        Odd = TRUE
    Else
        Odd = FALSE
    End If
End Function
```

See Also

TRUE
Boolean (data type)

FileAttr( )   **Function**

Description: When the value of attr is 1, this function returns an integer indicating the mode in which the file was opened:

1   input mode,
2   output mode,
8   append mode.

When the value of attr is 2, the function returns the file handle (an integer) assigned by the operating system.

Syntax: FileAttr(fileNum, attr)

Parameters: fileNum

The integer assigned to a file when it is opened with the Open statement. This is the number Symantec Basic uses instead of a filename to refer to the file.

attr

A numeric expression whose value is 1 or 2, depending on what information is to be returned.

Example: The following example calls the FileAttr( ) function to determine the mode in which a file was opened, Then it calls FileAttr( ) to get the file handle given to the file by the operating system.

```
Open "testfile" as #1

'theMode contains the value 8 for append mode
theMode% = FileAttr(1,1)

'fileHandle now contains the file handle
'of the file
fileHandle% = FileAttr(1,2)

Close #1
```

See Also

[FileLen()](#)
[GetAttr()](#)
[FileType()](#)
[FileExists()](#)
[Open](#)
[SetAttr](#)

FileCopy

Statement

Description:	Copies a *source$* file to a *destination$* file.
Syntax:	FileCopy *source$, destination$*
Comments:	The FileCopy function takes the following parameters:

Parameter	Description
source$	String containing the name of a single file to copy.
	The *source$* parameter cannot contain wildcards (? or *) but may contain path information.
destination$	String containing a single, unique destination file, which may contain a drive and path specification.

The file will be copied and renamed if the *source$* and *destination$* filenames are not the same.

Some platforms do not support drive letters and may not support dots to indicate current and parent directories.

Example: This example copies the autoexec.bat file to "autoexec.sav", then opens the copied file and tries to copy it again--which generates an error.

```
Sub Main()
    On Error Goto ErrHandler
    FileCopy "C:\autoexec.bat", "C:\autoexec.sav"
    Open "C:\autoexec.sav" For Input As # 1
    FileCopy "C:\autoexec.sav", "C:\autoexec.sv2"
    Close
    Exit Sub

ErrHandler:
    If Err = 55 Then                    'File already open.
      MsgBox "Cannot copy an open file. Close it and try again."
    Else
      MsgBox "An unspecified file copy error has occurred."
    End If
    Resume Next
```

End Sub

See Also

Kill
[Name . . .As](Name)

FileDateTime( ) Function

Description: This function returns a double-precision serial number indicating the date and time of the first file matching the specification. It is the number of days since Dec. 30, 1899, which is the zero date. A run-time error results if the file does not exist. The value returned can be used with the date/time functions (Year( ), Month( ), Day( ), Weekday( ), Minute( ), Second( ), and Hour ( )) to extract individual elements.

Syntax: FileDateTime(filename)

Parameter: filename

A string expression containing the complete or relative pathname to a file. It can contain wildcards (* and ?).

Example: The following example retrieves the date and time of a file.

```
dateAndTime# = FileDateTime("TESTFILE")
fileHour% = Hour(dateAndTime)
fileMinute% = Minute(dateAndTime)
fileSecond% = Second(dateAndTime)
fileMonth% = Month(dateAndTime)
fileDay% = Day(dateAndTime)
fileYear% = Year(dateAndTime)
fileWeekday% = Weekday(dateAndTime)
```

See Also

FileLen()
GetAttr()
FileType()
FileAttr()
FileExists()

FileDirs

Statement

Description: This statement fills the specified string array with the directory names that match the directory specification.

Syntax: FileDirs arrayName [, dirSpec]

Parameters: arrayName

The name of a one-dimensional string array.
dirSpec

A string expression that specifies a path. It can contain wildcards (* and ?). The default specifies all the subdirectories in the current directory.

Example: The following example processes all the directory names in the current directory.

```
'Declare a dynamic string array
Dim buffer$( )

'Assume the default specification of *.*
FileDirs buffer

'Were any directories found?
If ArrayDims(buffer) = 1 Then
    numDirsFound% = UBound(buffer)-LBound(buffer)+1
    For i = LBound(buffer) To UBound(buffer)
        ... 'Do something with each name
    Next i
Else
    'Otherwise, no directories found
    numDirsFound% = 0
End If
```

See Also

[FileList](#)
[Dir$()](#)
[CurDir$()](#)
[ChDir](#)

FileExists( )   **Function**

Description: This function returns TRUE if the filename is a valid file, or FALSE if the filename does not exist.

Syntax: FileExists(filename)

Parameter: filename

A string expression containing a complete or relative pathname. Wildcards (* and ?) can be used.

Examples: The following example determines whether a file with the name MYFILE exists in the current directory.

```
exist% = FileExists("myfile")
```

The next example determines whether any files with the extension .SM exist in the root directory.

```
exist% = FileExists("\*.sm")
```

See Also

[FileLen()](#)
[GetAttr()](#)
[FileType()](#)
[FileAttr()](#)
[FileParse$()](#)

FileLen( ) **Function**

Description: This function returns the length of the specified file in bytes. It is used in place of the LOF( ) function to retrieve the length of a file without first opening the file. A run-time error results if the file does not exist. The value returned is a number of type long.

Syntax: FileLen(filename)

Parameter: filename

A string expression containing a filename. When wildcards (* and ?) are used, the length of the first matching file is returned.

Examples: The following example determines the length of the file named TESTFILE.

```
fileLength& = FileLen("TESTFILE")
    The next example determines the length of the first file in the
current directory.
fileLength& = FileLen("*.*")
```

See Also

[GetAttr()](#)
[FileType()](#)
[FileAttr()](#)
[FileParse$()](#)
[FileExists()](#)
[Loc()](#)

FileList   Statement

Description: This statement fills the specified string array with filenames that match the file specification and have the specified attributes.

Syntax: FileList arrayName [, fileSpec [, fileAttr]]

Parameters: arrayName

The name of a one-dimensional string array.

fileSpec

A string expression specifying filenames. It can contain wildcards (* and ?). The default is "*.*" (all files with extensions). To specify all files, use "*".

fileAttr

A numeric expression specifying file attributes.
The default is ebArchive + ebReadOnly + ebNormal (33). Any of the following constants can be summed:

0	ebNormal	Normal file,
1	eReadOnly	Read-only file,
2	ebHidden	Hidden file,
4	ebSystem	System file,
8	ebVolume	Volume label,
16	ebDirectory	Directory,
32	ebArchive	File has changed since last backup,
64	ebNone	File has no attributes.

Example: The following example returns the names of the files in a specific directory with specific attributes and a specific extension.

```
'Declare a one dimensional string array
Dim dgnNames$(1 To 100)

'Find all *.dgn files in c:\lvl,
' including read-only, hidden, and archive

attr% = ebNormal OR ebReadOnly OR ebHidden OR ebArchive
FileList dgnNames, "c:\lvl\*.dgn", attr

'Were any files found?
If ArrayDims(dgnNames) = 1 Then
    numDgnsFound% = UBound(dgnNames) - LBound(dgnNames) + 1
    For i = LBound(dgnNames) To UBound(dgnNames)
              ...              'Do something with *.DGN names
```

```
        Next i
Else
    'Otherwise, no filenames matching the spec
    'were found
    numDgnsFound% = 0

End If
```

See Also

[FileDirs](#)
[Dir$()](#)

FileParse$( )   Function

Description: This function returns a string containing the full or partial filename obtained by extracting characters from the specified filename.
What is returned depends on the value of the operation parameter.

Syntax: FileParse$(filename[, operation])

Parameters: filename

A string expression containing a filename. A file by the specified name does not need to exist. If no drive or directory is specified, the current drive or directory is assumed.

operation

A numeric expression with a value from 0 to 5 that specifies which portion of the filename to extract:

0   Complete filename including drive and path,
1   Drive letter,
2   Path from the root directory,
3   Filename,
4   Root of filename (up to 8 characters),
5   File extension (up to 3 characters).

The default is 0, the complete filename. A run-time error occurs if operation is not a value from 0 to 5.

Examples: The following example returns the complete name of the specified file in the current directory.

```
fullName$ = FileParse$("TESTFILE")
```

The next example returns the extension of the filename contained in the first ten elements of fileArray.

```
For i = 1 to 10
    fileExt$ = FileParse$(fileArray(i), 5)
Next i
```

See Also

[FileLen()](#)
[GetAttr()](#)
[FileType()](#)
[FileAttr()](#)
[FileExists()](#)

FileType( )   Function

Description: This function returns one of the file-type constants, ebDOS (which is 1) or ebWindows (which is 2), depending on whether the specified file executes in DOS or Windows. A value other than 1 or 2 is returned if the type is not known, but the value is always an integer.

Syntax: FileType(filename)

Parameter: filename

A string expression containing the complete or relative pathname to a file. It cannot contain wildcards (* and ?).

Example: The following example determines the type of the file named TESTFILE.

```
theType% = FileType("TESTFILE")
If theType = ebDOS Then
    ...'It is a DOS executable file.
    'Do something.
ElseIf theType = ebWindows Then
    ...'It is a Windows executable file.
    'Do something else.
Else
    ...'The file type is unknown.
    'Do something else.
End If
```

See Also

[FileLen()](#)
[GetAttr()](#)
[FileAttr()](#)
[FileExists()](#)

Fix( )   **Function**

Description: This function returns the integer part of the specified numeric expression.

Syntax: Fix(exprN)

Parameter: exprN

                  A numeric expression in the range for an integer.

Example: The following examples illustrate the Fix( ) function.

```
'x is assigned 13
x = Fix(13)
'x is assigned -13
x = Fix(-13)
'x is assigned 13
x = Fix(13.5)
'x is assigned -13
x = Fix(-13.5)
```

See Also

[Int()](#)
[CInt()](#)

For...Next   Construct

Description: This construct repeats a series of statements a specified number of times. The first time through the loop, the counter is equal to start. Each succeeding time through the loop, the counter is increased by the amount specified in increment. The For...Next loop continues executing until the counter exceeds its range or until an Exit For statement is executed. If end is greater than start the increment must be positive. If end is less than start, the increment must be negative.

Syntax: For counter = start To end [Step increment]
        [statements]
   Next [counter]

   counter   A numeric variable.

   start   A numeric expression containing the initial value to be assigned to a counter and, therefore, the beginning of the counter's range.

   end   A numeric expression containing the last value to be assigned to a counter and, therefore, the end of the counter's range.

   increment   A numeric expression containing the value that is added to the counter each time the statements in the loop are repeated. It can be negative or positive. Its default is +1. If it is 0, an infinite loop results.   **statement**

   s   Any series of executable statements.

Example: The following example uses a For loop to request and add scores.

```
Dim Counter As Integer     ' For loop counter.
Dim Score As Integer' Input number.
Dim Total As Integer' Total of scores.
Dim NumberOfScores As Integer     'Number of scores
...
NumberOfScores = Val(InputBox$ ("How many scores are there?"))
Total = 0
' beginning of loop
For Counter = 1 To NumberOfScores
    Score = Val(InputBox$ ("Please enter a score."))
    Total = Total + Score
Next
' end of loop
MsgBox "The total of the scores is " + Str$(Total)

...
```

See Also

[Do . . .Loop](#)
[While . . .Wend](#)

Format, Format$ functions

Description:	Returns a String formatted to user specification.
Syntax:	Format[$](*expression* [,*Userformat$*])
Comments:	Format$ returns a String, whereas Format returns a String variant.

The Format$/Format functions take the following parameters:

Parameter	Description
expression	String or numeric expression to be formatted.
Userformat$	Format expression that can be either one of the built-in Symantec Basic formats or a user-defined format consisting of characters that specify how the expression should be displayed.

String, numeric, and date/time formats cannot be mixed in a single *Userformat$* expression.

If *Userformat$* is omitted and the expression is numeric, then these functions perform the same function as the Str$ or Str statements, except that they do not preserve a leading space for positive values.

If *expression* is Null, then a zero-length string is returned.

Built-In Formats

To format numeric expressions, you can specify one of the built-in formats. There are two categories of built-in formats: one deals with numeric expressions and the other with date/time values. The following tables list the built-in numeric and date/time format strings, followed by an explanation of what each does.

Numeric Formats

Format	Description
General number	Display the numeric expression as is, with no additional formatting.
Currency	Displays the numeric expression as currency, with thousands separator if necessary.
Fixed	Displays at least one digit to the left of the decimal separator and two digits to

	the right.
Standard	Displays the numeric expression with thousands separator if necessary. Displays at least one digit to the left of the decimal separator and two digits to the right.
Percent	Displays the numeric expression multiplied by 100. A percent sign (%) will appear at the right of the formatted output. Two digits are displayed to the right of the decimal separator.
Scientific	Displays the number using scientific notation. One digit appears before the decimal separator and two after.
Yes/No	Displays No if the numeric expression is 0. Displays Yes for all other values.
True/False	Displays False if the numeric expression is 0. Displays True for all other values.
On/Off	Displays Off if the numeric expression is 0. Displays On for all other values.

Date/Time Formats

Format	Description
General date	Displays the date and time. If there is no fractional part in the numeric expression, then only the date is displayed. If there is no integral part in the numeric expression, then only the time is displayed. Output is in the following form: 1/1/95 01:00:00 AM.
Long date	Displays a long date.
Medium date	Displays a medium date-prints out only the abbreviated name of the month.
Short date	Displays a short date.
Long time	Displays the long time. The default is: h:mm:ss.
Medium time	Displays the time using a 12-hour clock. Hours and minutes are displayed, and the AM/PM designator is at the end.
Short time	Displays the time using a 24-hour clock. Hours and minutes are displayed.

User-Defined Formats

In addition to the built-in formats, you can specify a user-defined format by using characters that have special meaning when used in a format expression. The following tables list the characters you can use for numeric, string, and date/time formats and explain their functions.

Numeric Formats

Character	Meaning
Empty string	Displays the numeric expression as is, with no additional formatting.
0	This is a digit placeholder.

	Displays a number or a 0. If a number exists in the numeric expression in the position where the 0 appears, the number will be displayed. Otherwise, a 0 will be displayed. If there are more 0s in the format string than there are digits, the leading and trailing 0s are displayed without modification.
#	This is a digit placeholder.
	Displays a number or nothing. If a number exists in the numeric expression in the position where the number sign appears, the number will be displayed. Otherwise, nothing will be displayed. Leading and trailing 0s are not displayed.
.	This is the decimal placeholder.
	Designates the number of digits to the left of the decimal and the number of digits to the right. The character used in the formatted string depends on the decimal placeholder, as specified by your locale.
%	This is the percentage operator.
	The numeric expression is multiplied by 100, and the percent character is inserted in the same position as it appears in the user-defined format string.
,	This is the thousand separator.
	The common use for the thousands separator is to separate thousands from hundreds. To specify this use, the thousands separator must be surrounded by digit placeholders. Commas appearing before any digit placeholders are specified are just displayed. Adjacent commas with no digit placeholders specified between them and the decimal mean that the number should be divided by 1,000 for each adjacent comma in the format string. A comma immediately to the left of the decimal has the same function. The actual thousands separator character used depends on the character specified by your locale.
E-E+e-e+	These are the scientific notation operators, which display the number in scientific notation. At least one digit placeholder must exist to the left of E-, E+, e-, or e+. Any digit placeholders displayed to the left of E-, E+, e-, or e+ determine the number of digits displayed in the exponent. Using E+ or e+ places a + in front of positive exponents and a - in front of negative exponents. Using E- or e- places a - in front of negative exponents and nothing in front of positive exponents.

:	This is the time separator.
	Separates hours, minutes, and seconds when time values are being formatted. The actual character used depends on the character specified by your locale.
/	This is the date separator.
	Separates months, days, and years when date values are being formatted. The actual character used depends on the character specified by your locale.
-+$()space	These are the literal characters you can display.
	To display any other character, you should precede it with a backslash or enclose it in quotes.
\	This designates the next character as a displayed character.
	To display characters, precede them with a backslash. To display a backslash, use two backslashes. Double quotation marks can also be used to display characters. Numeric formatting characters, date/time formatting characters, and string formatting characters cannot be displayed without a preceding backslash.
"ABC"	Displays the text between the quotation marks, but not the quotation marks. To designate a double quotation mark within a format string, use two adjacent double quotation marks.
*	This will display the next character as the fill character.
	Any empty space in a field will be filled with the specified fill character.
	Numeric formats can contain one to three parts. Each part is separated by a semicolon. If you specify one format, it applies to all values. If you specify two formats, the first applies to positive values and the second to negative values. If you specify three formats, the first applies to positive values, the second to negative values, and the third to 0s. If you include semicolons with no format between them, the format for positive values is used.

String Format Character	Meaning
@	This is a character placeholder.
	Displays a character if one exists in the expression in the same position; otherwise, displays a space. Placeholders are filled from right to left unless the format string specifies left to right.

&	This is a character placeholder.
	Displays a character if one exists in the expression in the same position; otherwise, displays nothing. Placeholders are filled from right to left unless the format string specifies left to right.
<	This character forces lowercase.
	Displays all characters in the expression in lowercase.
>	This character forces uppercase.
	Displays all characters in the expression in uppercase.
!	This character forces placeholders to be filled from left to right. The default is right to left.

Date/Time Formats

Character	Meaning
c	Displays the date as ddddd and the time as ttttt. Only the date is displayed if no fractional part exists in the numeric expression. Only the time is displayed if no integral portion exists in the numeric expression.
d	Displays the day without a leading 0 (1-31).
dd	Displays the day with a leading 0 (01-31).
ddd	Displays the day of the week abbreviated (Sun-Sat).
dddd	Displays the day of the week (Sunday-Saturday).
ddddd	Displays the date as a short date.
dddddd	Displays the date as a long date.
w	Displays the number of the day of the week (1-7). Sunday is 1; Saturday is 7.
ww	Displays the week of the year (1-53).
m	Displays the month without a leading 0 (1-12). If m immediately follows h or hh, m is treated as minutes (0-59).
mm	Displays the month with a leading 0 (01-12). If mm immediately follows h or hh, mm is treated as minutes with a leading 0 (00-59).
mmm	Displays the month abbreviated (Jan-Dec).
mmmm	Displays the month (January-December).
q	Displays the quarter of the year (1-4).
y	Displays the day of the year (1-366).

yy	Displays the year, not the century (00-99).
yyyy	Displays the year (1000-9999).
h	Displays the hour without a leading 0 (0-24).
hh	Displays the hour with a leading 0 (00-24).
n	Displays the minute without a leading 0 (0-59).
nn	Displays the minute with a leading 0 (00-59).
s	Displays the second without a leading 0 (0-59).
ss	Displays the second with a leading 0 (00-59).
ttttt	Displays the time. A leading 0 is displayed if specified by your locale.
AM/PM	Displays the time using a 12-hour clock. Displays an uppercase AM for time values before 12 noon. Displays an uppercase PM for time values after 12 noon and before 12 midnight.
am/pm	Displays the time using a 12-hour clock. Displays a lowercase am or pm at the end.
A/P	Displays the time using a 12-hour clock. Displays an uppercase A or P at the end.
a/p	Displays the time using a 12-hour clock. Displays a lowercase a or p at the end.
AMPM	Displays the time using a 12-hour clock. Displays the string s1159 for values before 12 noon and s2359 for values after 12 noon and before 12 midnight.

Example:

```
Sub Main()
    a# = 1199.234
    MsgBox Format$(a,"General Number")
    MsgBox Format$(a,"Currency")
    MsgBox Format$(a,"Standard")
    MsgBox Format$(a,"Fixed")
    MsgBox Format$(a,"Percent")
    MsgBox Format$(a,"Scientific")
    MsgBox Format$(True,"Yes/No")
    MsgBox Format$(True,"True/False")
    MsgBox Format$(True,"On/Off")

    da$ = Date$
    MsgBox Format$(da,"General Date")
    MsgBox Format$(da,"Long Date")
    MsgBox Format$(da,"Medium Date")
    MsgBox Format$(da,"Short Date")

    ti$ = Time$
    Msgbox Format$(ti,"Long Time")
```

```
        MsgBox Format$(ti,"Medium Time")
        MsgBox Format$(ti,"Short Time")

        A# = 12445601.234
        MsgBox Format$(A,"0,0.00")
        MsgBox Format$(A,"##,###,###.###")
End Sub
```

See Also

[Str$()](#)
[CStr()](#)

FreeFile( ) **Function**

Description: This function returns an integer representing the next available file number that can be used in an Open statement for opening a new file.

Syntax: FreeFile[( )]

Example: The following example uses the function twice, once to find an available file number for an input file and later to find an available file number for an output file.

```
'Assign the file number to a variable
inFileNum% = FreeFile
Open "infile" For Input As inFileNum
'The parentheses are optional
outFileNum% = FreeFile( )
Open "outfile" For Output As outFileNum
'The file numbers are saved in variables
'so that they can be used to close
'the files
Close inFileNum
Close outFileNum
```

See Also

[FileAttr()](#)
[Open](#)

Function...End Function   Construct

Description: This construct declares a function. When executed, the function returns the value of the expression that is assigned to the name of the function. If no expression is assigned, the function returns zero for numeric functions and an empty string for string functions. End Function transfers control back to the statement that called the function. Recursion is allowed.

Syntax: name [(parameterList [As type]...)]

    [statements]
    name = expr
    [statements]
End Function

name   The name of this user-defined function. It can include a type declarator to indicate the type of value the function returns.

parameterList   A list of parameters for the function, separated by commas. The syntax is:

Parameter: [, parameter]...

and the syntax for each parameter is:

[ByVal] parameterName [( )] [As type]

where ByVal indicates that the parameter is to be passed by value. Either a type declarator is used at the end of parameterName or the As type phrase is used to tell the type of the parameter. The parentheses are for arrays, which must be passed by reference.

type The data type that the function returns. This is used when no type declarator is appended to the name.

name = expr   An assignment statement that assigns an expression of the specified type to the name of the function.

s   Zero or more executable statements.

Examples: The following example declares a string function with no parameters.

```
Function Test( ) As String
    Test = "Hello World"
End Function
    The next example declares an integer function with an integer
parameter.
Function Half%(x%)
    Half = x/2
End Function
```

See Also

[Sub . . .End Sub](#)

Fv   **Function**

Description:	Calculates the future value of an annuity based on periodic fixed payments and a constant rate of interest.
Syntax:	Fv(*Rate, Nper, Pmt,Pv,Due*)
Comments:	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The Fv function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. Make sure that annual rates are normalized for monthly periods (divided by 12).
NPer	Double representing the total number of payments (periods) in the annuity.
Pmt	Double representing the amount of each payment per period. Payments are entered as negative values, whereas receipts are entered as positive values.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, whereas in the case of a retirement annuity, the present value would be the amount of the fund.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.

Rate and *NPer* values must be expressed in the same units. If *Rate* is expressed as a percentage per month, then *NPer* must also be expressed in months. If *Rate* is an annual rate, then the *NPer* must also be given in years.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example:
This example calculates the future value of 100 dollars paid periodically for a period of 10 years (120 months) at a rate of 10% per year (or .10/12 per month) with payments made on the first of the month. The value is displayed in a dialog box. Note that payments are negative values.

```
Sub Main()
    a# = Fv((.10/12),120,-100.00,0,1)
    MsgBox "Future value is: " + Format$(a,"Currency")
End Sub
```

See Also

[IRR](#)
[MIRR](#)
[Npv](#)
[Pv](#)

Get

Statement

Description:	Retrieves data from a random or binary file and stores that data into the specified variable.
Syntax:	Get [#] *filenumber*, [*recordnumber*], *variable*
Comments:	The Get statement accepts the following parameters:

Parameter	Description
filenumber	Integer used by Symantec Basic to identify the file. This is the same number passed to the Open statement.
recordnumber	Long specifying which record is to be read from the file.
	For binary files, this number represents the first byte to be read starting with the beginning of the file (the first byte is 1). For random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647.
	If the *recordnumber* parameter is omitted, the next record is read from the file (if no records have been read yet, then the first record in the file is read). When this parameter is omitted, the commas must still appear, as in the following example:
	Get #1,,recvar
	If *recordnumber* is specified, it overrides any previous change in file position specified with the Seek statement.
variable	Variable into which data will be read. The type of the variable determines how the data is read from the file, as described below.

With random files, a runtime error will occur if the length of the data being read exceeds the

reclen parameter specified with the Open statement. If the length of the data being read is less than the record length, the file pointer is advanced to the start of the next record. With binary files, the data elements being read are contiguousthe file pointer is never advanced.

Variable Types

The type of the *variable* parameter determines how data will be read from the file. It can be any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are read from the file.
Long	4 bytes are read from the file.
String (variable)	In binary files, variable-length strings are read by first determining the specified string variable's length and then reading that many bytes from the file. For example, to read a string of eight characters:

```
s$ = String$(" ",8)
Get #1,,s$
```

In random files, variable-length strings are read by first reading a 2-byte length and then reading that many characters from the file.

String (fixed)	Fixed-length strings are read by reading a fixed number of characters from the file equal to the string's declared length.
Double	8 bytes are read from the file (IEEE format).
Single	4 bytes are read from the file (IEEE format).
Date	8 bytes are read from the file (IEEE double format).
Boolean	2 bytes are read from the file. Nonzero values are True, and zero values are False.
Variant	A 2-byte VarType is read from the file, which determines the format of the data that follows. Once the VarType is known, the data is read individually, as described above. With user-defined errors, after the 2-byte VarType, a 2-byte unsigned integer is read and assigned as the value of the user-defined error, followed by 2 additional bytes of information about the error. The exception is with strings, which are always preceded by a 2-byte string length.
User-defined	Each member of a user-defined data type is read individually. In binary files, variable-length strings within user-defined types are read by first reading a 2-byte length followed by the string's content. This storage is different from variable-length strings outside of user-defined types. When reading user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.
Arrays	Arrays cannot be read from a file using the Get statement.
Objects	Object variables cannot be read from a file using the Get statement.

Example: This example opens a file for random write, then writes ten records into the file with the values 10...50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a dialog box.

```
Const Crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "test2.dat" For Random Access Write As #1
    For X% = 1 to 10
       Y% = X * 10
       Put #1,X,Y
    Next X
```

```
        Close
        Pstr$ = ""
        Open "test2.dat" For Random Access Read As #1
        For Y = 1 to 5
            Get #1,y,X
            Pstr = Pstr + "Record " + Str$(Y) + ": " + Str$(X) + crlf
        Next Y
        MsgBox Pstr
        Close
End Sub
```

See Also

Open
Put
Input #
Line Input #
Input$()

GetAttr( ) Function

Description: This function returns an integer indicating the attributes of the first file matching the specification. The attribute value returned contains the sum of the following attributes that apply to the file:

0	ebNormal	Normal file,
1	eReadOnly	Read-only file,
2	ebHidden	Hidden file,
4	ebSystem	System file,
8	ebVolume	Volume label,
16	ebDirectory	Directory,
32	ebArchive	File has changed since last backup,
64	ebNone	File has no attributes.

Syntax: GetAttr(filename)

Parameter: filename

A string expression containing the complete or relative pathname to a file. It can contain wildcards (* and ?).

Example: The following example determines whether the AUTOEXEC.BAT file is read-only.

```
If GetAttr("C:\AUTOEXEC.BAT") AND ebReadOnly Then
    answer = TRUE            'AUTOEXEC.BAT is read-only
Else
    answer = FALSE           'AUTOEXEC.BAT is not read-only
End If
```

See Also

[SetAttr](#)
[FileAttr()](#)

GoSub   Statement

Description: This statement transfers control to the statement associated with the specified label. The label must appear somewhere inside the current function or subroutine. It begins with a letter and ends with a colon. If the label does not exist, a compile-time error occurs. If a Return statement is used to end the set of statements following the label, it transfers control to the statement immediately following the GoSub statement.

Syntax: GoSub label

label Any identifier.

Example: The following example uses the GoSub statement to repeat the same sequence of statements throughout a script.

```
Sub Main
...
'Write standard header to first file
GoSub PrepareHeader
...
'Write standard header to second file
GoSub PrepareHeader
...
'Write standard header to third file
GoSub PrepareHeader
...
'The Exit Sub keeps you from executing
'The PrepareHeader routine unless you are sent to it
Exit Sub

PrepareHeader:
    'sequence of statements that write
    'header lines to a file
Return
End Sub
```

See Also

[Goto](#)
[Return](#)

GoTo   **Statement**

Description: This statement transfers control to the statement associated with the label. The label (followed by a colon) appears on a line inside the current function or subroutine. If the label does not exist, a compile-time error occurs.

Syntax: GoTo label

label Any identifier.

Example: The following example uses a GoTo statement to skip the sequence of statements between the If statement and LabelOne.

```
...
If JobDone Then
    GoTo LabelOne
End If
... 'sequence of statements
LabelOne:
... rest of statements in subroutine
```

See Also

GoSub
Call

GroupBox   Statement

Description:	This statement can appear only within a Begin Dialog…End Dialog construct. It defines a group box for a dialog box template.
Syntax:	GroupBox x, y, width, height, name
Parameters:	x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the group box in dialog units.

width, height

The integers indicating the width and height of the group box in dialog units.

name

A string variable or literal that specifies the name of the group box.

Example: The following example displays a dialog box with two check boxes within a group box.

```
Dim checkMsg2$, chkMsg$(1)
chkMsg(0) = "unchecked!"
chkMsg(1) = "checked!"
checkMsg2 = "No, check me!"

'Declare the dialog
Begin Dialog userDialog 15,28,100,100, "Untitled"
    GroupBox 4,4,84,51, "Check Boxes"
    CheckBox 10,15,48,14, "Check me!", .CheckBox1
    CheckBox 10,35,68,14, checkMsg2, .CheckBox2
    OKButton 55,64,41,14
End Dialog

'Declare the name for the instance of the template
Dim myDialog As userDialog

'Make the first check box initially checked
myDialog.CheckBox1 = 1

'Display the instance of the template
Dialog myDialog

'What was the result?
MsgBox "Check Box 1 was " + chkMsg(myDialog.CheckBox1)
MsgBox "Check Box 2 was " + chkMsg(myDialog.CheckBox2)
```

See Also

[CancelButton](#)
[CheckBox](#)
[ComboBox](#)
[Dialog()](#)
[Dialog](#)
[DropListBox](#)
[ListBox](#)
[OKButton](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PushButton](#)
[Text](#)
[TextBox](#)
[Begin Dialog . . .End Dialog](#)
[PictureButton](#)

Hex$( )   **Function**

Description: This function returns a string expression containing the hexadecimal value of the specified numeric expression. The parameter is rounded to the nearest whole number before it is converted to hexadecimal format. The string contains up to four hex digits for an integer, and up to eight digits for a long.

Syntax: Hex$(exprN)

Parameter: exprN

A numeric expression in the range for longs.

Example: The following example converts the decimal number 16 to hexadecimal.

```
hexOf16$ = Hex$(16)
'Result is the string "10"
```

See Also

[Oct$()](#)

Hour( )   **Function**

Description: This function returns the hour of the day encoded in the specified serial parameter. The value returned is an integer ranging from 0 to 23.

Syntax: Hour(serial)

Parameter: serial

A double-precision expression containing a serial time.

Example: After calling the Now( ) function, you can extract the current hour from the serial date and time.

```
'Get the current date and time
serialDT# = Now( )

'Extract the hour
theHour% = Hour(serialDT)
```

See Also

[Day()](Day())
Minute()
Second()
Month()
Year()
Weekday()
DatePart

If...Then...Else statement

Syntax 1:

```
If condition Then statements [Else else_statements]
```

Syntax 2:

```
If condition Then
  [statements]
[ElseIf else_condition Then
  [elseif_statements]]
[Else
  [else_statements]]
End If
```

Description: Conditionally executes a statement or group of statements.

Comments: The single-line conditional statement (syntax 1) has the following parameters:

Parameter	Description
condition	Any expression evaluating to a Boolean value.
statements	One or more statements separated with colons. This group of statements is executed when *condition* is True.
else_statements	One or more statements separated with colons. This group of statements is executed when *condition* is False.

The multiline conditional statement (syntax 2) has the following parameters:

Parameter	Description
condition	Any expression evaluating to a Boolean value.
statements	One or more statements to be executed when condition is True.
else_condition	Any expression evaluating to a Boolean value. The *else_condition* is evaluated if *condition* is False.
elseif_statements	One or more statements to be exected when *condition* is False and

else_condition is True.

else_statements   One or more statements to be executed when both *condition* and *else_condition* are False.

There can be as many ElseIf conditions as required.

Example:

```
'This example inputs a name from the user and checks to see whether it
'is MICHAEL or MIKE using three forms of the If...Then...Else
'statement. It then branches to a statement that displays a welcome
'message depending on the user's name.

Sub Main()
    uname$ = UCase(InputBox("Enter your name:","Enter Name"))
    If uname$ = "MICHAEL" Then GoSub MikeName

    If uname$ = "MIKE" Then
       GoSub MikeName
       Exit Sub
    End If

    If uname$ = "" Then
       MsgBox "Since you don't have a name, I'll call you MIKE!"
       uname$ = "MIKE"
       GoSub MikeName
    ElseIf uname$ = "MICHAEL" Then
       GoSub MikeName
    Else
       GoSub OtherName
    End If
    Exit Sub

MikeName:
    MsgBox "Hello, MICHAEL!"
    Return

OtherName:
    MsgBox "Hello, " & uname$ & "!"
    Return
End Sub
```

Platform(s):   All.

See Also

[Choose (function)](#)
[Switch (function)](#)
[IIf (function)](#)
[Select Case...End Select](#)

IIf   **Function**

Syntax:	IIf(condition,TrueExpression,FalseExpression)
Description:	Returns *TrueExpression* if *condition* is True; otherwise, returns *FalseExpression*.
Comments:	Both expressions are calculated before IIf returns.

The IIf function is shorthand for the following construct:

```
If condition Then
    variable = TrueExpression
Else
    variable = FalseExpression
End If
```

Example:

```
Sub Main()
    s$ = "Car"
    MsgBox "You have a " & IIf(s$ = "Car","nice car.","nice non-car.")
End Sub
```

Platform(s):	All.

See Also

[Choose (function)](#)
[Switch (function)](#)
[If...Then...Else (statement)](#)
[Select Case...End Select](#)

If...End If Construct

Description: A conditional construct that executes a statement or a group of statements if the logical expression leading to those statements is TRUE.

Syntax1: If logicalExpr Then statement [Else statement]

Syntax2:
```
If logicalExpr1 Then
    [statements]
[ElseIf logicalExpr2 Then
    [statements]]
[Else [statements]]
End If
```

logicalExpr   An expression containing relational and/or logical operators.   statement

One executable statement. statement
s   One or more executable statements.

Examples: In the following example, the Total appears on the screen only when it is greater than zero.

```
Dim Total As Integer
...
If Total > 0 Then
    MsgBox "TOTAL: " + Str$(Total)
End If
...
```

In the next example, the series of ElseIf statements include all the possible values for Choice.

```
...
If (Choice = 1) and (Count < Total) Then
    Count = Count + 1
ElseIf (Choice = 2) Then
    Count = Count - 1
ElseIf (Choice = 3) Then
    Count = 0
    MsgBox "Starting over."
Else
    MsgBox "Invalid choice."
End If
...
```

Imp   **Operator**

Description:	Performs a logical or binary implication on two expressions.
Syntax:	*expression1* Imp *expression2*
Comments:	If both expressions are either Boolean, Boolean variants, or Null variants, then a logical implication is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Binary Implication

If the two expressions are Integer, then a binary implication is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary implication is then performed, returning a Long result.

Binary implication forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:

```
1     Imp    1    =    1              Example:
0     Imp    1    =    1              5        01101001
1     Imp    0    =    0              6        10101010
0     Imp    0    =    1              Imp      10111110
```

Example: This example compares the result of two expressions to determine whether one implies the other.

```
Sub Main()
    a=10
    b=20
    c=30
    d=40
```

```vba
    If (a < b) Imp (c < d) Then
        MsgBox "a is less than b implies that c is less than d."
    Else
        MsgBox "a is less than b does not imply that c is less than d."
    End If

    If (a < b) Imp (c > d) Then
        MsgBox "a is less than b implies that c is greater than d."
    Else
        MsgBox "a is less than b does not imply that c is greater than d."
    End If
End Sub
```

See Also

[Or](#)
[XOR](#)
[Eqv (operator)](#)
[And](#)

Inline   **Statement**

```
Inline name [parameters]
    anytext
End Inline
```

Description: Allows execution or interpretation of a block of text.

Comments: The Inline statement takes the following parameters:

Parameter	Description
name	Identifier specifying the type of inline statement.
parameters	Comma-separated list of parameters.
anytext	Text to be executed by the Inline statement. This text must be in a format appropriate for execution by the Inline statement.

The end of the text is assumed to be the first occurrence of the words End Inline appearing on a line.

Example:

```
Sub Main()
    Inline MacScript
      -- This is an AppleScript comment.
      Beep
      Display Dialog "AppleScript" buttons "OK" default button "OK"
      Display Dialog Current Date
    End Inline
End Sub
```

Platform(s):   All.

Input #   **Statement**

Description: This statement reads data from the specified file into the specified variables. The file must be open and in input mode.

Syntax: Input [#]fileNum, var[, var]...

Parameters: fileNum

The integer assigned to a file when it is   with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

var

A variable of the same data type as the data to be read from the file.

Example: TESTFILE contains the following two lines:

```
5,"Hello"
6,"World!"
    The following example reads the items into the corresponding
variables.
Open "testfile" For Input As #99

'Note that this also reads the 6 on the 2nd line
Input #99, five%, hello$, six%

'Now read in the last string
Input #99, world$
```

See Also

Open
Get
Line Input #
Input$()

Input$( )   **Function**

Description: This function returns a string containing the specified number of characters read from the specified file. The file must be open and in input mode. The function reads all characters including spaces and carriage returns.

Syntax: Input$(charNum, [#]fileNum)

Parameters: charNum

A numeric expression indicating the number of characters to read.

fileNum

The integer assigned to a file when it is opened with the Open statement. This is the number Symantec Basic uses instead of a filename to refer to the file.

Example: The following example uses Input$( ) to read the first ten characters of a file into the string variable BUFFER.

```
Open "testfile" For Input As #1
BUFFER$ = Input$(10,#1)
```

See Also

[Open](#)
[Get](#)
[Input #](#)
[Line Input #](#)

InputBox$( )   Function

Description:	This function displays a dialog box with a message, a text box, a name, and the OK and Cancel command buttons. You can position the dialog box in the current window, or use the default position, which centers the dialog box in the window. The dialog box does not resize itself to fit the message. It accommodates only 12 lines of text with about 20 characters each. This function returns a string containing the contents of the text box if the user clicks OK, or the empty string if the user cancels the dialog box.
Syntax:	InputBox$(message [, name [, contents [, x, y]]])
Parameters:	message
	A string expression for the user to respond to. This can be multiple lines separated by carriage return/linefeeds (Chr$(13) + Chr$(10)).
	name
	A string expression containing the name of the dialog box. By default, there is no name.
	contents
	A string expression to be used as the initial contents of the text box. The user can accept this or type a new string. The default is the empty string.
	x, y
	The integers indicating the horizontal and vertical distances from the upper-left corner of the active window to the upper-left corner of the dialog box in twips. (There are 1440 twips to an inch.) By default, the dialog box is centered in the window.
Example:	The following example fills the text box with a name of a company: ACME. If ACME is the most frequently used company name, making it the default saves the user time. The user can change the name when appropriate.

```
theMessage$ = "What company makes the request?"
theTitle$ = "Requesting Company"
Company$ = InputBox$(theMessage, theTitle, "ACME")
```

See Also

[MsgBox](#)
[AskBox$()](#)
[AskPassword$()](#)
[OpenFileName$()](#)
[SaveFileName$()](#)
[SelectBox()](#)
[AnswerBox()](#)

InStr( )   **Function**

Description: This function returns an integer indicating the starting position of a substring within a search string. The first character position in the string is 1. It returns 0 if the substring cannot be found; the search string is the empty string; or the starting point is greater than the length of the search string.

Syntax: InStr([start,] search, find)

Parameters: start

An integer indicating the starting place for the search. The default is 1, the first character of the search string.

search

A string expression containing the string to search.

find

A string expression containing the substring to find.

Example: The following example deletes trailing percent signs from LastName.

```
'Find the first occurrence of the
'trailing percent signs
Position% = InStr(LastName, "%")
If Position > 1 Then
    LastName = Left$(LastName, Position - 1)
End If
```

See Also

[Mid$()](#)
[Option Compare](#)
[Item$()](#)
[Word$()](#)
[Line$()](#)

Int( )   **Function**

Description: This function returns the   first integer less than the specified number. The sign is preserved.

Syntax: Int(exprN)

Parameter: exprN

A numeric expression in the range for integers.

Example: The following examples illustrate the Int( ) function.

```
'x is assigned 13 because 13 < 13.7
x = Int(13.7)
'x is assigned -14 because -14 < -13.2
x = Int(-13.2)
```

See Also

[Fix()](#)
[CInt()](#)

Integer   **Data type**

Syntax:	Integer
Description:	A data type used to declare whole numbers with up to four digits of precision.
Comments:	Integer variables are used to hold numbers within the following range:

$$-32768 <= \textit{integer} <= 32767$$

Internally, integers are 2-byte short values. Thus, when appearing within a structure, integers require 2 bytes of storage. When used with binary or random files, 2 bytes of storage are required.

When passed to external routines, Integer values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.

The type-declaration character for Integer is %.

Platform(s):	All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CInt( )](#)

IPmt   Function

Description: Returns the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax: IPmt(*Rate*, *Per*, *Nper*, *Pv*, *Fv*, *Due*)

Comments: An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages, monthly savings plans, and retirement plans.

The following table describes the different parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the payment periods are monthly, be sure to divide the annual interest rate by 12 to get the monthly rate.
Per	Double representing the payment period for which you are calculating the interest payment. If you want to know the interest paid or received during period 20 of an annuity, this value would be 20.
Nper	Double representing the total number of payments in the annuity. This is usually expressed in months, and you should be sure that the interest rate given above is for the same period that you enter here.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan because that is the amount of cash you have in the present. In the case of a retirement plan, this value would be the current value of the fund because you have a set amount of principal in the plan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero because you will have paid it off. In the case of a savings plan, the future value would be the balance of the account after all payments are made.

| *Due* | Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period (usually, the end of the month). If this value is 1, then payments are due at the start of each period (the beginning of the month). |

Rate and *Nper* must be in expressed in the same units. If *Rate* is expressed in percentage paid per month, then *Nper* must also be expressed in months. If *Rate* is an annual rate, then the period given in *Nper* should also be in years or the annual *Rate* should be divided by 12 to obtain a monthly rate.

If the function returns a negative value, it represents interest you are paying out, whereas a positive value represents interest paid to you.

Example: This example calculates the amount of interest paid on a $1,000.00 loan financed over 36 months with an annual interest rate of 10%.Payments are due at the beginning of the month. The interest paid during the first 10 months is displayed in a table.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Msg$ = ""
    For x = 1 to 10
        IPM#   =  IPmt((.10/12) , x, 36, 1000, 0, 1)
        Msg$ = Msg$ + Format$(X,"00") + " : " + Format$(IPM," 0,0.00") +
crlf
    Next x
    MsgBox Msg$
End Sub
```

See Also

[NPer](#)
[Pmt](#)
[PPmt](#)
[Rate](#)

IRR Function

Description: Returns the internal rate of return for a series of periodic payments and receipts.

Syntax: IRR(*ValueArray()*,*Guess*)

Comments: The internal rate of return is the equivalent rate of interest for an investment consisting of a series of positive and/or negative cash flows over a period of   regular intervals. It is usually used to project the rate of return on a business investment that requires a capital investment up front and a series of investments and returns on investment over time.

The IRR function requires the following parameters:

Parameter	Description
ValueArray()	Array of Double numbers that represent payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one negative value to indicate the initial investment (negative value) and the amount earned by the investment (positive value).
Guess	Double containing your guess as to the value that the IRR function will return. The most common guess is .1 (10 percent).

The value of IRR is found by iteration. It starts with the value of *Guess* and cycles through the calculation adjusting *Guess* until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, IRR fails, and the user must pick a better guess.

Example: This example illustrates the purchase of a lemonade stand for $800 and a series of incomes from the sale of lemonade over 12 months. The projected incomes for this example are generated in two For...Next Loops, and then the internal rate of return is calculated and displayed. (Not a bad investment!)

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim Valu#(12)
    Valu(1) = - 800                             'Initial investment
    PStr$ = Str$(Valu(1)) + ", "
    'Calculate the second through fifth months' sales.
    For X = 2 To 5
```

```
        Valu(X) = 100 + (X*2)
        PStr = PStr + Str$(Valu(X)) + ", "
    Next x
    'Calcluate the sixth through twelfth months' sales.
    For X = 6 To 12
        Valu(X) = 100 + (X*10)
        PStr = PStr + Str$(Valu(X)) + ", "
    Next x
    'Calcluate the equivalent investment return rate.
    Retrn# = IRR(Valu,.1)
    PStr = "The values: " + crlf + PStr + crlf + crlf
    MsgBox PStr + "Return rate: " + Format$(Retrn,"Percent")
End Sub
```

See Also

[Fv](#)
[MIRR](#)
[Npv](#)
[Pv](#)

Is   **Operator**

Description: Returns True if the two operands refer to the same object; returns False otherwise.

Syntax: *object* Is [*object* | Nothing]

Comments: This operator is used to determine whether two object variables refer to the same object. Both operands must be object variables of the same type (i.e., the same data object type or both of type Object).

The Nothing constant can be used to determine whether an object variable is uninitialized:

```
If MyObject Is Nothing Then MsgBox "MyObject is uninitialized."
```

Uninitialized object variables reference no object.

Example: This function inserts the date into a Microsoft Word document.

```
Sub InsertDate(ByVal WinWord As Object)
    If WinWord Is Nothing Then
       MsgBox "Object variant is not set."
    Else
       WinWord.Insert Date$
    End If
End Sub

Sub Main()
    Dim WinWord As Object
    On Error Resume Next
    WinWord = CreateObject("word.basic")
    InsertDate WinWord
End Sub
```

See Also

Like

IsDate   **Function**

Syntax:	IsDate(expression)
Description:	Returns True if *expression* can be legally converted to a date; returns False otherwise.
Example:	

```
Sub Main()
    Dim a As Variant
Retry:
    a = InputBox("Enter a date.","Enter Date")
    If IsDate(a) Then
       MsgBox Format(a,"long date")
    Else
       Msgbox "Not quite, please try again!"
       Goto Retry
    End If
End Sub
```

Platform(s):	All.

See Also

[Variant (data type)](#)
[IsEmpty (function)](#)
[IsError (function)](#)
[IsObject (function)](#)
[VarType (function)](#)
[IsNull (function)](#)

IsEmpty   **Function**

Syntax:	IsEmpty(expression)
Description:	Returns True if *expression* is a Variant variable that has never been initialized; returns False otherwise.
Comments:	The IsEmpty function is the same as the following:

```
(VarType(expression) = ebEmpty)
```

Example:

```
Sub Main()
    Dim a As Variant
    If IsEmpty(a) Then
        a = 1.0#                        'Give uninitialized data a Double value
0.0.
        MsgBox "The variable has been initialized to: " & a
    Else
        MsgBox "The variable was already initialized!"
    End If
End Sub
```

Platform(s):	All.

See Also

[Variant (data type)](#)
[IsDate (function)](#)
[IsError (function)](#)
[IsObject (function)](#)
[IsNull (function)](#)
[VarType (function)](#)

IsError   **Function**

Syntax: IsError(*expression*)

Description: Returns True if *expression* is a user-defined error value; returns False otherwise.

Example:

```
'This example creates a function that divides two numbers. If there
'is an error dividing the numbers, then a variant of type "error" is
'returned. Otherwise, the function returns the result of the division.
'The IsError function is used to determine whether the function
'encountered an error.

Function Div(ByVal a,ByVal b) As Variant
    If b = 0 Then
       Div = CVErr(2112)          'Return a special error value.
    Else
       Div = a / b                        'Return the division.
    End If
End Function

Sub Main()
    Dim a As Variant
    a = Div(10,12)
    If IsError(a) Then
       MsgBox "The following error occurred: " & CStr(a)
    Else
       MsgBox "The result of the division is: " & a
    End If
End Sub
```

Platform(s):   All.

See Also

[Variant (data type)](#)
[IsEmpty (function)](#)
[IsDate (function)](#)
[IsObject (function)](#)
[VarType (function)](#)
[IsNull (function)](#)

IsMissing   Function

Syntax: IsMissing(variable)

Description: Returns True if *variable* was passed to the current subroutine or function; returns False if omitted.

Comments: The IsMissing is used with variant variables passed as optional parameters (using the Optional keyword) to the current subroutine or function. For non-variant variables or variables that were not declared with the Optional keyword, IsMissing will always return True.

Example:

```
'The following function runs an application and optionally minimizes it.
If
'the optional isMinimize parameter is not specified by the caller, then
the
'application is not minimized.

Sub Test(AppName As String,Optional isMinimize As Variant)
    app = Shell(AppName)
    If Not IsMissing(isMinimize) Then
       AppMinimize app
    Else
       AppMaximize app
    End If
End Sub

Sub Main
    Test "notepad.exe"                          'Maximize this application
    Test "notepad.exe",True              'Mimimize this application
End Sub
```

Platform(s): All.

See Also

[Declare](#)
[Sub...End Sub](#)
[Function...End Function](#)

IsNull   **Function**

Syntax: IsNull(*expression*)

Description: Returns True if *expression* is a Variant variable that contains no valid data; returns False otherwise.

Comments: The IsNull function is the same as the following:

```
(VarType(expression) = ebNull)
```

Example:

```
Sub Main()
    Dim a As Variant        'Initialized as Empty
    If IsNull(a) Then MsgBox "The variable contains no valid data."
    a = Empty * Null
    If IsNull(a) Then MsgBox "Null propagated through the expression."
End Sub
```

Platform(s):   All.

See Also

[Empty (constant)](#)
[Variant (data type)](#)
[IsEmpty (function)](#)
[IsDate (function)](#)
[IsError (function)](#)
[IsObject (function)](#)
[VarType (function)](#)

IsNumeric   Function

Syntax: IsNumeric(expression)

Description: Returns True if expression can be converted to a number; returns False otherwise.

Comments: If passed a number or a variant containing a number, then IsNumeric always returns True.

If a String or String variant is passed, then IsNumeric will return True only if the string can be converted to a number. The following syntaxes are recognized as valid numbers:

```
&Hhexdigits[&|%|!|#|@]
```

```
&[O]octaldigits[&|%|!|#|@]
```

```
[-|+]digits[.[digits]][E[-|+]digits][!|%|&|#|@]
```

If an Object variant is passed, then the default property of that object is retrieved and one of the above rules is applied.

IsNumeric returns False if expression is a Date.

Example:

```
Sub Main()
    Dim s$ As String
    s$ = InputBox("Enter a number.","Enter Number")

    If IsNumeric(s$) Then
        MsgBox "You did good!"
    Else
        MsgBox "You didn't do so good!"
    End If
End Sub
```

Platform(s): All.

See Also

[Variant (data type)](#)
[VarType (function)](#)
[IsEmpty (function)](#)
[IsDate (function)](#)
[IsError (function)](#)
[IsObject (function)](#)
[IsNull (function)](#)

IsObject   **Function**

Syntax: IsObject(expression)

Description: Returns True if *expression* is a Variant variable containing an Object; returns False otherwise.

Example:

```
'This example will attempt to find a running copy of Excel and create 'a
Excel object that can be referenced as any other object in 'BasicScript.

Sub Main()
    Dim v As Variant
    On Error Resume Next
    Set v = GetObject(,"Excel.Application")

    If IsObject(v) Then
       MsgBox "The default object value is: " & v = v.Value
    'Access value property of the object.
    Else
       MsgBox "Excel not loaded."
    End If
End Sub
```

Platform(s):   All.

See Also

[Variant (data type)](#)
[IsEmpty (function)](#)
[IsDate (function)](#)
[IsError (function)](#)
[VarType (function)](#)
[IsNull (function)](#)

Item$( )   Function

Description: This function returns a string containing all the items in the specified text starting with the item specified by the first parameter and ending with the item specified by the last parameter. It returns the empty string if the first parameter specifies a number greater than the number of items in the text. It returns all the rest of the items if the last parameter is greater than the number of items in the text.

Syntax: Item$(text, first[, last [, delimiters]])

Parameters: text

A string expression that contains items delimited by commas and end-of-line characters or other special delimiters.

first

An integer representing the first item to be read.

The first item in the text is number 1.

last

An integer representing the last item to be read.

The default is the value of first, so at least one item is read.

delimiters

A string expression specifying the delimiters used to separate items. Each character of the string is a delimiter. The default delimiters are the comma and end-of-line characters.

Example: In the following example, a list of names, separated by commas, is stored in the string variable itemText. A For...Next loop places each name from the string into a string array. No delimiters are specified because the comma is one of the default delimiters.

```
Dim nameList$(10)

'The text with the names
itemText$ = "John,Mary Jane,Ken,T.S."

For i=1 To ItemCount(itemText)
    'Parse each name
    nameList(i) = Item$(itemText, i)
Next i
```

See Also

[ItemCount()](#)
[Line$()](#)
[LineCount()](#)
[Word$()](#)
[WordCount()](#)

ItemCount( )   **Function**

Description: This function returns an integer indicating the number of items in the specified text.

Syntax: ItemCount(text[, delimiters])

Parameters: text

A string expression containing items delimited by commas, end-of-line characters, or other special delimiters.

delimiters

A string expression specifying the delimiters used to separate items in the text. Each character of the string is a delimiter. The default delimiters are the comma and end-of-line characters.

Example: The following example loops once for each item.

```
For i=1 To ItemCount(itemText)
    'Parse each name
    nameList(i) = Item$(itemText, i)
Next i
```

See Also

[Item$()](#)
[Line$()](#)
[LineCount()](#)
[Word$()](#)
[WordCount()](#)

Kill

Statement

Description: This statement deletes all the files that match the file specification.   If you attempt to delete an open file, a run-time error occurs.

Syntax: Kill fileSpec

Parameter: fileSpec

A string expression containing the complete or relative pathname to a file. It can contain a path and wildcards (* and ?).

Examples: The following example deletes all files in the current directory.

```
Kill "*.*"     'Be careful when doing this
    The next example deletes the file TESTFILE at the root of the C:
drive.
Kill "C:\TESTFILE"
```

See Also

[Name . . .As](#)

LBound( )   **Function**

Description: This function returns an integer indicating the lower bound for subscripts in the specified dimension of the specified array. Use LBound( ) with UBound( ) to determine the range of elements in a dimension of an array.

Syntax: LBound(arrayName [, dimension])

Parameters: arrayName

The name of an array variable.

dimension

An integer indicating the dimension in the array.

The default is 1 for the first dimension.

Example: The following example finds the lower bound for subscripts in the first dimension of a two-dimensional array using the LBound( ) function.

```
Dim Array1(0 To 3, 0 To 2) As Integer
'Determine the lower bound
lowest_subscript = LBound(Array1)
```

See Also

UBound()
ArrayDims()

LCase$( )   **Function**

Description: This function returns the lowercase equivalent of the specified string.

Syntax: LCase$(exprS)

Parameter: exprS

A string expression.

Example: The following call to LCase$( ) should result in the string "this is only a test" being assigned to the variable newString.

```
newString$ = LCase$("This is Only a Test!")
```

See Also

[UCase$()](#)

Left$( )   **Function**

Description: This function returns the specified number of characters beginning with the leftmost character. It returns the empty string if charNum is zero.
It returns the entire string if charNum is greater than or equal to the number of characters in the specified string.

Syntax: Left$(exprS, charNum)

Parameters: exprS

Any string expression.

charNum

An integer indicating the number of characters to return.

Example: The following example deletes trailing percent signs from LastName.

```
'Find the first occurrence of the trailing percent signs
Position% = InStr(LastName, "%")
If Position > 1 Then
    'Retain only the leftmost characters
    LastName = Left$(LastName, Position - 1)
End If
```

See Also

Right$()

Len( )   **Function**

Description: This function returns an integer indicating the number of characters in the specified string. It returns 0 if the string expression is empty.

Syntax: Len(exprS)

Parameters: exprS

Any string expression.

Example: The following example uses the Len function to determine the length of a message:

```
Length = Len(Message)
```

See Also

InStr()

Let　　　　　　　　　　　　　　**Statement**

Description:　This statement assigns a value to a variable. The reserved word Let is optional. It is supported for compatibility with other versions of BASIC.

Syntax:　[Let] var = expr

　var　The name of the variable being assigned a value.

　expr　The value to be assigned to a variable.

Examples:　Both of the following examples assign the value 5 to x.

```
Let x = 5
x = 5
```

You can use a variable on both sides of the first assignment statement that uses it. For example, the following statement increases the value of the variable Counter by one.

```
Counter = Counter + 1
```

When this statement is executed, the value of the Counter on the right side is 0 (the initial value of Counter) and the value of the Counter on the left is the sum of 0+1, which is 1.

See Also

[= (equal To)](#)

Like   Operator

Description:	Compares two strings and returns True if the *expression* matches the given *pattern*; returns False otherwise.
Syntax:	*expression* Like *pattern*
Comments:	Case sensitivity is controlled by the Option Compare setting.

The *pattern* expression can contain special characters that allow more flexible matching:

Character	Evaluates To
?	Matches a single character
*	Matches one or more characters.
#	Matches any digit.
[*range*]	Matches if the character in question is within the specified range.
[!*range*]	Matches if the character in question is not within the specified range.

A *range* specifies a grouping of characters. To specify a match of any of a group of characters, use the syntax [ABCDE]. To specify a range of characters, use the syntax [A-Z]. Special characters must appear within brackets, such as []*?#.

If *expression* or *pattern* is not a string, then both *expression* and *pattern* are converted to String variants and compared, returning a Boolean variant. If either variant is Null, then Null is returned.

The following table shows some examples:

expression	True If *pattern* Is	False If *pattern* Is
"EBW"	"E*W", "E*"	"E*B"
"Symantec Basic"	"B*[r-t] icScript"	"B[r-t]ic"
"Version"	"V[e]?s*n"	"V[r]?s*N"
"2.0"	"#.#", "#?#"	"###", "#?[!0-9]"
"[ABC]"	"[[]*]"	"[ABC]", "[*]"

Example:	This example demonstrates various uses of the Like function.

```
Sub Main()
    A$ = "This is a string variable of 123456 characters"
```

```vba
    B$ = "123.45"
    If A Like "[A-Z][g-i]*" Then MsgBox "Comparison is True."
    If B Like "##3.##" Then MsgBox "Comparison is True."
    If A Like "*variable*" Then MsgBox "Comparison is True."
End Sub
```

See Also

[Is](#)
[Option Compare](#)

Line Input #   Statement

Description: This statement reads one line of a text file into a string variable. It positions the file pointer after the carriage return/linefeed of the last line read.

Syntax: Line Input [#]fileNum, text

Parameters: fileNum

The integer assigned to a file that is open for input. This is the number Symantec Basic uses instead of a filename to refer to the open file.

text

A string variable that will contain the line that is read.

Example: The following example can be used to skip past the first ten lines of a file.

```
Open "testfile" For Input As #7
For i = 1 To 10
    Line Input #7, dontcare$
Next i
```
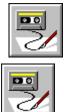
See Also

Line$( )   **Function**

Description: This function returns a string containing all the lines in the specified text starting with the line specified by the first parameter and ending with the line specified by the last parameter. It returns the empty string if the first parameter specifies a number greater than the number of lines in the text. It returns all the rest of the lines if the last parameter is greater than the number of lines in the text.

Syntax: Line$(text, first[, last])

Parameter: text

A string expression containing the lines of text delimited with carriage return/linefeed pairs.

first

An integer representing the first line to be read.
The first line in the text is number 1.

last

An integer representing the last line to be read.
The default is the value of first, so at least one line is read.

Example: In the following example, a string is assigned an address. Each component of the address is on a separate line. The Line$( ) function is then used to extract the name, street, city, and other information from it. LineCount( ) determines whether there is any other information besides the name, street, and city.

```
'Carriage-return/linefeed pair
crlf$ = Chr$(13) + Chr$(10)

'Make an address for the example
address$ = "John Doe" + crlf$
address = address + "123 Old Lane" + crlf$
address = address + "New City" + crlf$
address = address + "Other Information"

'Now parse the address
name$ = Line$(address, 1)
street$ = Line$(address, 2)
city$ = Line$(address, 3)

'Is there other information
If LineCount(address) > 3 Then
    other$ = Line$(address, 4, LineCount(address))
End If
```

See Also

[Item$( )](#)
[ItemCount( )](#)
[LineCount( )](#)
[Word$( )](#)
[WordCount( )](#)

LineCount( ) Function

Description: This function returns an integer indicating the number of lines in the specified text.

Syntax: LineCount(text)

Parameter: text

A string expression containing lines of text delimited with carriage return/linefeed pairs.

Example: The following example determines the number of lines in a string expression.

```
numLines = LineCount(textString)
```

See Also

[Item$( )](#)
[ItemCount( )](#)
[Line$( )](#)
[Word$( )](#)
[WordCount( )](#)

ListBox   Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a list box for a dialog box template.

Syntax: ListBox x, y, width, height, itemsArray, .field

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the list box in dialog units.

width, height

The integers indicating the width and height of the list box in dialog units.

itemsArray

The name of a one-dimensional string array that contains the elements to be placed into the list box.

field

An integer variable used to set and/or retrieve the subscript of the array element selected from the list box. Setting this field to a subscript from the array of items gives the list box an initial selection.

Example: The following example displays a dialog containing a list box.

```
Dim listOfItems$(9)

'Initialize the array of items
For i = 0 To 9
    listOfItems$(i) = "Item " + Str$(i)
Next

'Declares a dialog box template
Begin Dialog ListDialog 15,24,100,84, "Lists"
    ListBox 5,5,90,48, listOfItems, .ListBox1
    OKButton 55,64,41,14
End Dialog

'Declares an instance of the template
Dim dialog1 As ListDialog

'Displays the instance of the template
Dialog dialog1

'Display the result
MsgBox listOfItems(dialog1.ListBox1)
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog](#)

[Dialog( )](#)

[DropListBox](#)

[GroupBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[Picture](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog...End Dialog](#)

[PictureButton](#)

Loc( )   **Function**

Description: This function returns an integer indicating the position of the file pointer in the specified file.

Syntax: Loc(fileNum)

Parameter: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

Example: In the following example, two lines are read. Then the Loc( ) function determines the new position of the file pointer. The Seek( ) function could have been used instead of the Loc( ) function.

```
Open "testfile" For Input As #1

'Read in first line
Input #1, num34%, strCDE$

'Read in second line
Input #1, num45%, strDEF$

'Determine the new position of the file pointer
curPos& = Loc(1) 'equivalent to curPos& = Seek(1)
Close #1
```

See Also

[Seek( )](#)
[Seek](#)

Lock Statement

Description:	Locks a section of the specified file, preventing other processes from accessing that section of the file until the Unlock statement is issued.	
Syntax:	Lock [#] *filenumber* [,{*record*	[*start*] To *end*}]	
Comments:	The Lock statement requires the following parameters:	

Parameter	Description
filenumber	Integer used by Symantec Basic to refer to the open file-the number passed to the Open statement.
record	Long specifying which record to lock.
start	Long specifying the first record within a range to be locked.
end	Long specifying the last record within a range to be locked.

For sequential files, the *record*, *start*, and *end* parameters are ignored. The entire file is locked.

The section of the file is specified using one of the following:

Syntax	Description
No parameters	Locks the entire file (no record specification is given).
record	Locks the specified record number (for Random files) or byte (for Binary files).
to *end*	Locks from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
start to *end*	Locks the specified range of records (for Random files) or bytes (for Binary files).

The lock range must be the same as that used to subsequently unlock the file range, and all locked ranges must be unlocked before the file is closed. Ranges within files are not unlocked automatically by Symantec Basic when your script terminates, which can cause file access problems for other processes. It is a good idea to group the Lock and Unlock statements close together in the code, both for readability and so subsequent readers can see that the lock and

unlock are performed on the same range. This practice also reduces errors in file locks.

Example: This example creates test2.dat and fills it with ten string variable records. These are displayed in a dialog box. The file is then reopened for read/write, and each record is locked, modified, rewritten, and unlocked. The new records are then displayed in a dialog box.

```
Const Crlf = Chr$(13) + Chr$(10)

Sub Main()
    A$ = "This is record number: "
    B$ = "0"
    Rec$ = ""
    Msg$ = ""
    Open "test2.dat" For Random Access Write Shared As #1
    For x% = 1 To 10
       Rec = A + Str$(x)
       Lock #1,x
       Put #1,,Rec
       Unlock #1,x
       Msg = Msg + Rec + crlf
    Next x
    Close
    MsgBox "The records are: " + crlf + Msg
    Msg = ""
    Open "test2.dat" For Random Access Read Write Shared As #1
    For x = 1 To 10
       Rec = Mid$(Rec,1,23) + Str$(11-x)
       Lock #1,x
       Put #1,x,Rec
       Unlock #1,x
       Msg = Msg + Rec + crlf
    Next x
    MsgBox "The records are: " + crlf + Msg
    Close
End Sub
```

See Also

Unlock
Open

LOF( )   **Function**

Description: This function returns a long indicating the length (a number of bytes) of the specified open file.

Syntax: LOF(fileNum)

Parameter: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

Example: In the following example, the length of an open file is stored in a variable.

```
Open "testfile" As #1
fileLength = LOF(1)
Close #1
```

See Also

[Loc( )](#)
[Open](#)
[FileLen( )](#)

Log( )   **Function**

Description: This function returns the natural logarithm of a specified number. The value returned is a number of type double.

Syntax: Log(exprN)

Parameter: exprN

A numeric expression greater than 0.

Example: The following example shows a simple use of the Log( ) function.

```
'Definition of e
e# = 2.71828
'Natural logarithm of e equals 1 (approximately)
lne# = Log(e)
```

See Also

[Exp( )](#)

Long   **Data type**

Syntax:	Long
Description:	Long variables are used to hold numbers (with up to ten digits of precision) within the following range:

$$-2,147,483,648 \leq \textbf{\textit{Long}} \leq 2,147,483,647$$

Internally, longs are 4-byte values. Thus, when appearing within a structure, longs require 4 bytes of storage. When used with binary or random files, 4 bytes of storage are required.

The type-declaration character for Long is &.

Platform(s):	All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CLng( )](#)

LSet   **Statement**

Syntax 1:   LSet *dest* = *source*

Syntax 2:   LSet *dest_variable* = *source_variable*

Description: Left-aligns the source string in the destination string or copies one user-defined type to another.

Comments: **Syntax 1**

The LSet statement copies the source string *source* into the destination string *dest*. The *dest* parameter must be the name of either a String or Variant variable. The *source* parameter is any expression convertible to a string.

If *source* is shorter in length than *dest*, then the string is left-aligned within *dest,* and the remaining characters are padded with spaces. If *source$* is longer in length than *dest*, then *source* is truncated, copying only the leftmost number of characters that will fit in *dest*.

The *destvariable* parameter specifies a String or Variant variable. If *destvariable* is a Variant containing Empty, then no characters are copied. If *destvariable* is not convertible to a String, then a runtime error occurs. A runtime error results if *destvariable* is Null.

Syntax 2

The source structure is copied byte for byte into the destination structure. This is useful for copying structures of different types. Only the number of bytes of the smaller of the two structures is copied. Neither the source structure nor the destination structure can contain strings.

Example: This example replaces a 40-character string of asterisks (*) with an RSet and LSet string and then displays the result.

```
Const Crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim Msg$, TmpStr$
    TmpStr = String$(40, "*")
    Msg = "Here are two strings that have been right-" + crlf
    Msg = Msg + "and left-justified in a 40-character string."
    Msg = Msg + crlf + crlf
    RSet TmpStr = "Right->"
    Msg = Msg & TmpStr & crlf
    LSet TmpStr = "<-Left"
    Msg = Msg & TmpStr & crlf
    MsgBox Msg
End Sub
```

See Also

RSet (statement)

LTrim$( )   Function

Description: This function returns a string with its leading spaces removed.

Syntax: LTrim$(exprS)

Parameter: exprS

A string expression.

Example: The following example demonstrates the use of LTrim$( ).

```
aString$ = "    10 leading spaces"
'Now remove the leading spaces
aString = LTrim$(aString)
'aString should now be equal to
'the string "10 leading spaces"
```

See Also

[RTrim$( )](#)
[Trim$( )](#)

Main   **Statement**

Description: Main is the first subroutine to be executed in a script. It controls the script's execution.

Syntax: Sub Main( )

     ...
End Sub

Example: The following example shows a script with Main as the only subroutine.

```
Sub Main
    'Count to 100
    For i = 1 To 100
            ...
    Next i
End Sub
```

Mid$ **Statement**

Description: This statement replaces part of a string with another. The resulting string has the same name, but is never longer than, the original string.

Syntax: Mid$(originalStr, start[, length]) = newStr

Parameters: originalStr

A string expression into which the new string will be inserted. After the statement, originalStr contains newStr (or a portion of it) as a substring.

start

A numeric expression indicating the starting position of the substring in originalStr that will be replaced by characters from newStr. The first character of originalStr is at position 1.

length

A numeric expression indicating the number of characters to replace. The default is from the starting position to the end of the string.

newStr

A string expression containing the characters to replace part of the originalStr.

Example: The following example replaces the substring "dog" in "My dog has fleas." with the substring "cat" so that the string becomes "My cat has fleas." This example assumes that you want to keep copies of both the source and target strings, so it makes a copy of the source string before using the Mid$ statement.

```
DogString$ = "My dog has fleas."
CatString$ = DogString 'copies the source
Mid$(CatString, 4) = "cat" 'changes the copy
    The next example replaces the substring "dog" with the substring
"elephant". Because "elephant" is longer than "dog", you cannot use the
Mid$ statement (or you can only substitute "ele" for "dog").
DogString$ = "My dog has fleas."
Length% = Len(DogString)
LeftEnd% = InStr(DogString, "dog") - 1
RightStart% = Length - LeftEnd - Len("dog")
ElephantString$ = Left$(DogString, LeftEnd) + ×    "elephant" + Right$
(DogString, RightStart)
```

See Also

Option Compare
Mid$( )

Mid$( )   Function

Description: This function returns a substring from the specified string. The substring begins at the specified starting position and contains the specified number of characters. It returns an empty string if the starting position is greater than the length of the specified string.

Syntax: Mid$(exprS, start[, length])

Parameters: exprS

Any string expression.

start

A numeric expression indicating the starting position for the substring to be returned.

length

A numeric expression indicating the number of characters to be in the substring. The default is to return all the rest of the characters in the string.

Example: Assuming that RecordString contains a last name starting at position 1 of length 25 characters and a first name starting at position 26 of length 15 characters, the following example retrieves the first and last names from the string.

```
RecordString$ = × "Smith********************Jane**********"
LastName$ = Mid$(RecordString, 1, 25)
FirstName$ = Mid$(RecordString, 26, 15)
'At this point:
'   LastName = "Smith********************"
'and    FirstName = "Jane**********"
```

See Also

[InStr( )](#)
[Option Compare](#)
[Mid$](#)

Minute( )   Function

Description: This function returns the minute of the day encoded in the specified serial time. The returned value in an integer ranging from 0 to 59.

Syntax: Minute(serial)

Parameter: serial

A double-precision expression containing a serial time.

Example: After calling the Now( ) function, you can extract the current minute from the date and time.

```
'Get the current date and time
serialDT# = Now( )

'Now extract the value
theMinute% = Minute(serialDT)
```

See Also

[Day( )](#)
[Second( )](#)
[Month( )](#)
[Year( )](#)
[Hour( )](#)
[Weekday( )](#)
[DatePart](#)

MIRR   Function

Description: Returns a Double representing the modified internal rate of return for a series of periodic payments and receipts.

Syntax: MIRR(*ValueArray*(),*FinanceRate*,*ReinvestRate*)

Comments: The modified internal rate of return is the equivalent rate of return on an investment in which payments and receipts are financed at different rates. The interest cost of investment and the rate of interest received on the returns on investment are both factors in the calculations.

The MIRR function requires the following parameters:

Parameter	Description
ValueArray()	Array of Double numbers representing the payments and receipts. Positive values are payments (invested capital), and negative values are receipts (returns on investment). There must be at least one positive (investment) value and one negative (return) value.
FinanceRate	Double representing the interest rate paid on invested monies (paid out).
ReinvestRate	Double representing the rate of interest received on incomes from the investment (receipts).

FinanceRate and *ReinvestRate* should be expressed as percentages. For example, 11 percent should be expressed as 0.11.

To return the correct value, be sure to order your payments and receipts in the correct sequence.

Example: This example illustrates the purchase of a lemonade stand for $800 financed with money borrowed at 10%. The returns are estimated to accelerate as the stand gains popularity. The proceeds are placed in a bank at 9 percent interest. The incomes are estimated (generated) over 12 months. This program first generates the income stream array in two For...Next loops, and then the modified internal rate of return is calculated and displayed. Notice that the annual rates are normalized to monthly rates by dividing them by 12.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim Valu#(12)
    Valu(1) = -800
    'Initial investment
```

```
    PStr$ = Str$(Valu(1)) + ", "
    For X = 2 To 5
       Valu(X) = 100 + (X*2)
    'Incomes months 2-5
       PStr = PStr + Str$(Valu(X)) + ", "
    Next x
    For X = 6 To 12
       Valu(X) = 100 + (X*10)                            'Incomes
months 6-12
       PStr = PStr + Str$(Valu(X)) + ", "
    Next x
    Retrn# = MIRR (Valu,.1/12,.09/12)              'Note: normalized
annual rates

    PStr = "The values: " + crlf + PStr + crlf + crlf
    MsgBox PStr + "Modified rate: " + Format$(Retrn,"Percent")
End Sub
```

See Also

[Fv](#)
[IRR](#)
[Npv](#)
[Pv](#)

MkDir   **Statement**

Description: This statement creates a new directory.

Syntax: MkDir dir

Parameter: dir

A string expression containing the name of the new directory to create.

Examples: The following example creates a directory named ASDF in the current directory.

```
MkDir "asdf"
```

The next example creates a directory on the C drive named ASDF.

```
MkDir "c:asdf"
```

See Also

[ChDir](#)
[ChDrive](#)
[CurDir$( )](#)
[Dir$( )](#)
[RmDir](#)

MOD   **Numeric Operator**

Description: This operator finds the remainder of operand1 divided by operand2.

If the operands are not whole numbers, Symantec Basic rounds them to whole numbers before performing the modulo operation.   The resulting value is a number of type long.

Syntax: operand1 MOD operand2

operand1   A numeric expression in the range for longs for the dividend.

operand2   A numeric expression in the range for longs for the divisor.

Examples: In the following example, the result of is 1 because 3 divided by 2 leaves a remainder of 1.

```
z = 3 MOD 1.5
    The result in the next example is 0 because 3 divided by 1 leaves a
remainder of 0.
z = 3 MOD 1.4
```

See Also

[/ (division)](#)
[\ (integer division)](#)

Month( ) Function

Description: This function returns the month of the date encoded in the specified serial date. The value returned is an integer ranging from 1 to 12.

Syntax: Month(serial)

Parameter: serial

A double-precision expression containing a serial date.

Example: After calling the Now( ) function, you can extract the current month from the date and time.

```
'Get the current date and time
serialDT# = Now( )

'Now extract the value
theMonth% = Month(serialDT)
```

See Also

[Day( )](#)
[Minute( )](#)
[Second( )](#)
[Year( )](#)
[Hour( )](#)
[Weekday( )](#)
[DatePart](#)

MsgBox   Statement

Description: This statement displays a message in a dialog box.

Syntax: MsgBox message [, type [, name]]

Parameters: message

A string expression for the user to respond to.

The message box is sized to fit the message. When the message is long, it is word wrapped. You can specify explicit line breaks by using Chr$ (13) + Chr$(10) to include carriage return/linefeeds.

type

A numeric expression specifying the characteristics of the message box. It is the sum of the numbers that correspond to the characteristics: command buttons, icon, default button, and modality. The default is 0. When a number is out of range, the default is used. The follow are legal values:

Number	Command Button Combination
0	OK (the default)
1	OK, Cancel
2	Abort, Retry, Ignore
3	Yes, No, Cancel
4	Yes, No
5	Retry, Cancel

Number	Icon
16	Stop
32	Question Mark
48	Exclamation Point
64	Information

Number	Default Button
0	First Button (the default)
256	Second Button
512	Third Button

Number	Modality (who waits until user responds)
0	Symantec Basic (the default)
4096	All applications

name

A string expression containing the title of the message box. The default name is "BasicScript."

Example: The following example displays the message "Hello, world!" along with an OK button.

```
MsgBox "Hello, world!"
```

See Also

[AskBox$( )](#)
[AskPassword$( )](#)
[InputBox$( )](#)
[OpenFileName$( )](#)
[SaveFileName$( )](#)
[SelectBox( )](#)
[AnswerBox( )](#)

MsgBox( )   **Function**

Description:	This function displays a message in a dialog box and returns an integer indicating the button the user selected:

Number	Button Pressed by User
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Syntax: MsgBox(message [, type [, name]])

Parameters: message

A string expression for the user to respond to.

The message box is sized to fit the message. When the message is long, it is word wrapped. You can specify explicit line breaks by using Chr$ (13) + Chr$(10) to include carriage return/linefeeds.

type

A numeric expression specifying the characteristics of the message box. It is the sum of the numbers that correspond to the characteristics: command buttons, icon, default button, and modality. The default is 0. When a number is out of range, the default is used. The follow are legal values:

Number	Command Button Combination
0	OK (the default)
1	OK, Cancel
2	Abort, Retry, Ignore
3	Yes, No, Cancel
4	Yes, No
5	Retry, Cancel

Number	Icon
16	Stop
32	Question Mark
48	Exclamation Point
64	Information

Number	Default Button
0	First Button (the default)

256	Second Button
512	Third Button

Number	Modality (who waits until user responds)
0	Symantec Basic (the default)
4096	All applications

name

A string expression containing the title of the message box. The default name is "BasicScript."

Example: The following example displays a message, the Yes and No command buttons, and the Stop icon. The function returns 6 or 7 depending on which button the user selects.

```
buttonChoice = MsgBox("Get me off this ship!", 4+16, "THE TITANIC")
```

See Also

[AskBox$( )](#)
[AskPassword$( )](#)
[InputBox$( )](#)
[OpenFileName$( )](#)
[SaveFileName$( )](#)
[SelectBox( )](#)
[AnswerBox( )](#)

Name...As   **Statement**

Description: This statement renames a file.

Syntax: Name oldFile As newFile

Parameters: oldFile

A string expression containing the name of the file to rename.

newFile

A string expression containing the new name for oldFile.

Examples: The following example renames the file TESTFILE as NEWFILE.

```
Name "testfile" As "newfile"
    The next example renames the file ROOTFILE in the root directory of
the C: drive as GOODFILE.
Name "c:\rootfile" As "goodfile"
```

See Also

[Kill](#)
[FileCopy](#)

New   **Keyword**

Syntax 1:	Dim ObjectVariable As New ObjectType
Syntax 2:	Set ObjectVariable = New ObjectType
Description:	Creates a new instance of the specified object type, assigning it to the specified object variable.
Comments:	The New keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types.
	At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate context) and returning a reference to that object, which is immediately assigned to the variable being declared.
	When that variable goes out of scope (i.e., the Sub or Function procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.
Platform(s):	All.

See Also

[Dim](#)
[Set](#)

Not   **Operator**

Description:	Returns either a logical or binary negation of *expression*.
Syntax:	Not *expression*
Comments:	The result is determined as shown in the following table:

If the Expression Is	Then the Result Is
True	False
False	True
Null	Null
Any numeric type	A binary negation of the number. If the number is an Integer, then an Integer is returned. Otherwise, the *expression* is first converted to a Long, then a binary negation is performed, returning a Long.
Empty	Treated as a Long value 0.

Example: This example demonstrates the use of the Not operator in comparing logical expressions and for switching a True/False toggle variable.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    A = True
    B = False
    SWITCH = True
    If (A and Not B) And (Not (A = B)) Then
       Msg$ = "A And Not B = True" + crlf
    Else
       Msg = "A And Not B = False" + crlf
    End If
    Msg = Msg + "Switch is now " + Format$(Switch,"True/False") + crlf
    Switch   = Not Switch
    Msg = Msg + "Switch is now " + Format$(Switch,"True/False") + crlf
    Switch   = Not Switch
    Msg = Msg + "Switch is now " + Format$(Switch,"True/False")
    MsgBox Msg
End Sub
```

See Also

[Boolean (data type)](#)

Nothing   **Constant**

Description: A value indicating that an object variable no longer references a valid object.
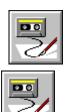
Example:

```
Sub Main()
    Dim a As Object
    If a Is Nothing Then
      MsgBox "The object variable references no object."
    Else
      MsgBox "The object variable references: " & a.Value
    End If
End Sub
```

Platform(s): All.

See Also

[Set](Set)

NOT   **Logical Operator**

Description: The logical expression containing this operator evaluates to TRUE if the expression is FALSE, or to FALSE if the expression is TRUE.

If the expression is numeric, the result is a bitwise NOT of the expression. If either of the expressions is a floating-point number, it is converted to a long before the bitwise NOT.

Syntax: NOT expr

expr   Numeric, relational, or logical expression.

Example: In the following example, the statement in the If construct is executed when a condition is not true.

```
'Do not give free admission to anyone
'not named Darlene
If NOT (personName = "Darlene") Then
    freeAdmission = FALSE
End If
```

Now( )   Function

Description: This function returns a double-precision number that is the serial representation of the current date and time. The integer part of the number is the number of days since Dec. 20, 1899 (the zero date).

The fraction of the number represents the time.

Syntax: Now( )

Example: The following statement stores the current serial date and time in a variable.

```
serialDT# = Now( )
```

See Also

Date$( )
Time$( )

NPer   **Function**

Description: Returns the number of periods for an annuity based on periodic fixed payments and a constant rate of interest.

Syntax: NPer(*Rate,Pmt,Pv,Fv,Due*)

Comments: An annuity is a series of fixed payments paid to or received from an investment over a period of time. Examples of annuities are mortgages, retirement plans, monthly savings plans, and term loans.

The NPer function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the periods are monthly, be sure to normalize annual rates by dividing them by 12.
Pmt	Double representing the amount of each payment or income. Income is represented by positive values, whereas payments are represented by negative values.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, and the future value (see below) would be zero.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero, and the present value would be the amount of the loan.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example: This example calculates the number of $100.00 monthly payments necessary to accumulate $10,000.00 at an annual rate of 10%. Payments are made at the beginning of the month.

```
Sub Main()
    ag# = NPer((.10/12),100,0,10000,1)
    MsgBox "The number of monthly periods is: " + Format$(ag,"Standard")
End Sub
```

See Also

[Imp](#)
[Pmt](#)
[Rate](#)

Npv   **Function**

Description: Returns the net present value of an annuity based on periodic payments and receipts, and a discount rate.

Syntax: Npv(*Rate*,*ValueArray*())

Comments: The Npv function requires the following parameters:

Parameter	Description
Rate	Double that represents the interest rate over the length of the period. If the values are monthly, annual rates must be divided by 12 to normalize them to monthly rates.
ValueArray()	Array of Double numbers representing the payments and receipts. Positive values are payments, and negative values are receipts.

There must be at least one positive and one negative value.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

For accurate results, be sure to enter your payments and receipts in the correct order because Npv uses the order of the array values to interpret the order of the payments and receipts.

If your first cash flow occurs at the beginning of the first period, that value must be added to the return value of the Npv function. It should not be included in the array of cash flows.

Npv differs from the Pv function in that the payments are due at the end of the period and the cash flows are variable. Pv's cash flows are constant, and payment may be made at either the beginning or end of the period.

Example: This example illustrates the purchase of a lemonade stand for $800 financed with money borrowed at 10%. The returns are estimated to accelerate as the stand gains popularity. The incomes are estimated (generated) over 12 months. This program first generates the income stream array in two For...Next loops, and then the net present value (Npv) is calculated and displayed. Note normalization of the annual 10% rate.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim Valu#(12)
    Valu(1) = -800
            'Initial investment
```

```
    PStr$ = Str$(Valu(1)) + ", "
    For X = 2 To 5
            'Months 2-5
       Valu(X) = 100 + (X*2)
       PStr = PStr + Str$(Valu(X)) + ", "
    Next x
    For X = 6 To 12
       'Months 6-12
       Valu(X) = 100 + (X*10)
    'Accelerated income
       Pstr = PStr + Str$(Valu(X)) + ", "
    Next x
    NetVal# = NPV ((.10/12),Valu)
    PStr = "The values: " + crlf + PStr + crlf + crlf
    MsgBox PStr + "Net present value: " + Format$(NetVal,"Currency")
End Sub
```

See Also

[Fv](#)
[IRR](#)
[MIRR](#)
[Pv](#)

Null( )   **Function**

Description: This function returns a null string that contains no characters and requires no memory. An empty string ("") also has no characters, but it requires some memory for storage.

Syntax: Null[( )]

Example: The following example shows the use of the Null function to free the space that once stored a string.
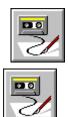
```
'Assign a string to a string variable
aString$ = "We take up space!"

'Now make the string variable a null string
aString = Null
```

Null   **Constant**

Description: Represents a variant of VarType 1.

Comments: The Null value has special meaning indicating that a variable contains no data.

Most numeric operators return Null when either of the arguments is Null. This "propagation" of Null makes it especially useful for returning error values through a complex expression. For example, you can write functions that return Null when an error occurs, then call this function within an expression. You can then use the IsNull function to test the final result to see whether an error occurred during calculation.

Since variants are Empty by default, the only way for Null to appear within a variant is for you to explicitly place it there. Only a few BasicScript functions return this value.

Example:

```
Sub Main()
    Dim a As Variant
    a = Null
    If IsNull(a) Then MsgBox "The variable is Null."
    MsgBox "The VarType of a is: " & VarType(a)        'Should display 1.
End Sub
```

Platform(s): **All.**

Oct$( )   Function

Description: This function converts a number to its octal equivalent. It returns a string containing only the number of octal digits necessary to represent the specified number.

Syntax: Oct$(exprN)

Parameter: exprN

A numeric expression in the range for longs that is rounded to the nearest whole number before it is converted to octal format.

Example: The following example converts the decimal number 16 to octal.

```
octOf16$ = Oct$(16)
'The result is the string "20"
```

See Also

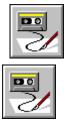[Hex$( )](#)

OKButton Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines an OK button for a dialog box template.

Syntax: OKButton x, y, width, height

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the OK button in dialog units.

width, height

The integers indicating the width and height of the OK button in dialog units.

Example: The following example displays an instance of a dialog template with an OK and a Cancel button. Selecting either button causes the dialog function that displays the template to end. If OK is selected, the Dialog( ) function returns TRUE. If Cancel is selected, the function returns FALSE. The result is displayed in a message box.

```
'Define the dialog box template
Begin Dialog userDialog 15, 28, 100, 100, "OK and Cancel"
    Text 40,14,48,8 "Do you want to continue?"
    OKButton 55, 64, 41, 14
    CancelButton 55, 82, 41, 14
End Dialog

'Declare the name of the instance
' of the template
Dim OKCancelDialog As userDialog

'Display the instance of the template
result = Dialog(OKCancelDialog)

'What was the result?
If result = TRUE Then
    MsgBox "OK"
Else
    MsgBox "Cancel"
End If
```

See Also

[CancelButton](#)
[CheckBox](#)
[ComboBox](#)
[Dialog( )](#)
[Dialog](#)
[DropListBox](#)
[GroupBox](#)
[ListBox](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PictureButton](#)

On Error

Statement

Description: This statement defines the action taken when a run-time error (that can be trapped) occurs. The On Error statement is valid only within the subroutine or function in which it appears. If you are using GoTo label, the error-handling routine is a set of statements that start with the label and end with a Resume statement. If an error occurs within the label...Resume error handling routine, a run-time error stops script execution. See the Resume statement.

Syntax: On Error {GoTo label | Resume Next | GoTo 0}

GoTo label   Transfers script execution to the specified label when a run-time error occurs.

label Any identifier.

Resume Next   Transfers script execution to the line following the line that caused the run-time error.

GoTo 0   Turns off the previously set method of error handling.

Example: The following example sends all errors to the same label. The statements between the label and the Resume statement display the error numbers and messages that Symantec Basic normally displays when a run-time error terminates a script. Err( ) is a predefined function that returns the value of the most recent error. Similarly, the Error$( ) function returns the error message associated with the most recent error.

```
On Error GoTo MessageDisplay
...
cmd$ = "[CreateGroup(" + quoted(Setup.GroupName) + ")]"
DDEExecute channel, cmd$
...
Exit Sub
'This routine can be used for all errors while
'you are debugging
'It may help you fix more than one error at a time
MessageDisplay:
   MsgBox Str$(Err( )) + Error$( )
Resume Next
```

See Also

Error
Resume

Open   **Statement**

Description: This statement opens a file in input, output, or append mode, assigns an integer to it, and enables the script to read or write to the file (depending on its mode). You can use the FreeFile( ) function to determine an available file number.

Syntax: Open filename [For {Input|Output|Append}]

  As [#]fileNum

Parameters: filename

A string expression containing the complete or relative pathname to a file. It cannot contain wildcards (* and ?).

For Input

Opens an existing file to read from it.
For Output

Opens an existing file and truncates its length to 0, or creates a new 0-length file to write to.
For Append

Opens a file to append text to its current contents or creates a new file. This is the default.
fileNum An integer between 1 and 255 used by Symantec Basic to identify the file.

Examples: The following example opens the file TESTFILE in append mode as file number 250.

```
'No mode is specified, so the default is append
Open "TESTFILE" As #250
    The next example opens the file TESTFILE in input mode as file number
0.
Open "TESTFILE" For Input As #0
```

See Also

[Close](#)
[Reset](#)
[FreeFile( )](#)

OpenFileName$( )   Function

Description: This function displays a standard "File Open" dialog box used to select a file. It returns a string containing the pathname for the file the user selects. It returns an empty string if the user cancels the dialog box.

Syntax: OpenFileName$(name[, extensions])

Parameters: name

A string expression containing the name for the dialog box.

extensions

A string expression that specifies the available file types in the following format:
type:ext[, ext][;type:ext[, ext]]...
Type is a string that identifies the file type, such as "Documents".
Ext is a valid file extension like *.BAT or *.?F?
The default is "All Files:*.*".

Example: The following example uses the OpenFileName( ) function to list all the scripts with the extension .SM. in any directory selected by the user.

```
FileTypes$ = "All Scripts:*.SM"
SelectedFile$ = OpenFileName$("Open Symantec Basic Script", FileTypes)
If SelectedFile = "" Then
    MsgBox "No file was selected!"
Else
    MsgBox "The file " + SelectedFile + " was selected."
End If
```

See Also

[MsgBox](#)
[AskBox$( )](#)
[AskPassword$( )](#)
[InputBox$( )](#)
[SaveFileName$( )](#)
[SelectBox( )](#)
[AnswerBox( )](#)

Option Base

Statement

Description: This statement sets the lower bound for array declarations that do not explicitly specify a lower bound. It must appear outside of any functions or subroutines.

Syntax: Option Base {0 | 1}

Example: The following example uses Option Base to set 1 as the default lower bound for array declarations.

```
Option Base 1
Sub Main( )
    'contains elements 1 to 12
    Dim MonthArray (12)
End Sub
```

See Also

[Dim](#)
[Public](#)
[Private](#)

OptionButton   Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines an option button for a dialog box template.

Syntax: OptionButton x, y, width, height, name

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the option button in dialog units.

width, height

The integers indicating the width and height of the option button in dialog units.

name

A string variable or literal that specifies the name of the option button.

Example: The following example displays a dialog box named "Flavors," containing three option buttons and an OK push button.

```
Dim flavors$(2)

flavors(0) = "Chocolate"
flavors(1) = "Vanilla"
flavors(2) = "Strawberry"

Begin Dialog OptionDialog 15,24,100,81, "Flavors"
    OptionGroup .Flavor
        OptionButton 5,5,90,14, flavors(0)
        OptionButton 5,25,90,14, flavors(1)
        OptionButton 5,45,90,14, flavors(2)
    OKButton 55,64,41,14
End Dialog

Dim FlavorDialog As OptionDialog
Dialog FlavorDialog

'What flavor option was selected
MsgBox flavors(FlavorDialog.Flavor)
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog( )](#)

[Dialog](#)

[DropListBox](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionGroup](#)

[Picture](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog...End Dialog](#)

[PictureButton](#)

Option Compare   Statement

Description: Controls how strings are compared.

Syntax: Option Compare [Binary | Text]

Comments: When Option Compare is set to Binary, then string comparisons are case-sensitive (e.g., "A" does not equal "a"). When it is set to Text, string comparisons are case-insensitive (e.g., "A" is equal to "a").

The default value for Option Compare is Binary.

The Option Compare statement affects all string comparisons in any statements that follow the Option Compare statement. Additionally, the setting affects the default behavior of Instr, StrComp, and the Like operator. The following table shows the types of string comparisons affected by this setting:

>	<	<>
<=	>=	Instr
StrComp	Like	

The Option Compare statement must appear outside the scope of all subroutines and functions. In other words, it cannot appear within a Sub or Function block.

Example: This example shows the use of Option Compare.

```
Option Compare Binary
Sub CompareBinary
    A$ = "This String Contains UPPERCASE."
    B$ = "this string contains uppercase."
    If A = B Then
      MsgBox "The two strings were compared case-insensitive."
    Else
      MsgBox "The two strings were compared case-sensitive."
    End If
End Sub

Option Compare Text
Sub CompareText
    A$ = "This String Contains UPPERCASE."
    B$ = "this string contains uppercase."
    If A = B Then
      MsgBox "The two strings were compared case-insensitive."
    Else
      MsgBox "The two strings were compared case-sensitive."
    End If
End Sub
```

```vb
Sub Main()
    CompareBinary                          'Calls subroutine above.
    CompareText                            'Calls subroutine above.
End Sub
```

See Also

Option CStrings Statement

Syntax:	Option CStrings {On | Off}
Description:	Turns on or off the ability to use C-style escape sequences within strings.
Comments:	When Option CStrings On is in effect, the compiler treats the backslash character as an escape character when it appears within strings. An escape character is simply a special character that cannot otherwise be ordinarily typed by the computer keyboard.

Escape	Description	Equivalent Expression
\r	Carriage return	Chr$(13)
\n	Line feed	Chr$(10)
\a	Bell	Chr$(7)
\b	Backspace	Chr$(8)
\f	Form feed	Chr$(12)
\t	Tab	Chr$(9)
\v	Vertical tab	Chr$(11)
\0	Null	Chr$(0)
\"	Double quotation mark	"" or Chr$(34)
\\	Backslash	Chr$(92)
\?	Question mark	?
\'	Single quotation mark	'
\xhh	Hexadecimal number	Chr$ (Val("&H*hh*))
\ooo	Octal number	Chr$ (Val("&O*ooo*"))
\ anycharacter	Any character	*anycharacter*

With hexadecimal values, BasicScript stops scanning for digits when it encounters a nonhexadecimal digit or two digits, whichever comes first. Similarly, with octal values, BasicScript stops scanning when it encounters a nonoctal digit or three digits, whichever comes first.

When Option CStrings Off is in effect, then the backslash character has no special meaning. This is the default.

Example:

```
Option CStrings On

Sub Main()
    MsgBox "They said, \"Watch out for that clump of grass!\""
    MsgBox "First line.\r\nSecond line."
    MsgBox "Char A: \x41 \r\n Char B: \x42"
End Sub
```

Platform(s):   All.

OptionGroup   Statement

Description: This statement defines the beginning of a group of option buttons,
and the variable that indicates which option button is selected when the dialog box closes. It can
appear only within a Begin Dialog...End Dialog construct. Option buttons are numbered beginning
with 0.

Syntax: OptionGroup .field

Parameter: .field

An integer variable used to select an option button and/or retrieve the selected option button's
value.

Example: The following example displays a dialog titled "Flavors," containing three option buttons and an OK
push button.

```
Dim flavors$(2)

flavors(0) = "Chocolate"
flavors(1) = "Vanilla"
flavors(2) = "Strawberry"

Begin Dialog OptionDialog 15,24,100,81, "Flavors"
    OptionGroup .Flavor
        OptionButton 5,5,90,14, flavors(0)
        OptionButton 5,25,90,14, flavors(1)
        OptionButton 5,45,90,14, flavors(2)
    OKButton 55,64,41,14
End Dialog
```

See Also

CancelButton

CheckBox

ComboBox

Dialog( )

Dialog

DropListBox

GroupBox

ListBox

OKButton

OptionButton

Picture

PictureButton

PushButton

Text

TextBox

Begin Dialog...End Dialog

OR   **Logical Operator**

Description: This logical operator usually joins two logical or relational expressions into another logical expression. The result is TRUE if either or both of the expressions are TRUE; otherwise it is FALSE.

If the expressions are numeric, the result is a bitwise OR of the two numbers. If either of the expressions is a floating-point number, the two expressions are converted to longs before the bitwise OR.

Syntax: expr1 OR expr2

expr1, expr2

Numeric, relational, or logical expressions.

Example: In the following example, the OR operator ensures that at least one condition is satisfied.

```
'Give free admission to children,
'the elderly, and the handicapped
If age < 18 OR age > 65 OR handicapped = TRUE Then
    freeAdmission = TRUE
End If
```

See Also

[XOR](#)
[Eqv (operator)](#)
[Imp](#)
[AND](#)

Pi   **Constant**

Description: The Double value 3.14159265358979323846264383279.

Syntax: Pi

Comments: Pi can also be determined using the following formula:

```
4 * Atn(1)
```

Example: This example illustrates the use of the Pi constant.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dia# = 5
    Circ# = Pi * Dia
    Area# = Pi * ((Dia / 2) ^ 2)
    Msg$ = "Diameter: 5" + crlf
    Msg = Msg + "Circumference: " + Format$(Circ,"Standard") + crlf
    Msg = Msg + "Area: " + Format$(Area,"Standard")
    MsgBox Msg
End Sub
```

See Also

[Tan( )](#)
[Atn( )](#)
[Cos( )](#)
[Sin( )](#)

Picture   **Statement**

Description:	Creates a picture control in a dialog box template.
Syntax:	Picture *X*,*Y*,*width*,*height*,*PictureName$*,*PictureType* [,[.*Identifier*] [,*style*]]
Comments:	Picture controls are used for the display of graphics images only. The user cannot interact with these controls.

The Picture statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
PictureName$	String containing the name of the picture. If *PictureType* is 0, then this name specifies the name of the file containing the image. If *PictureType* is 10, then *PictureName$* specifies the name of the image within the resource of the picture library.
PictureType	Integer specifying the source for the image. The following sources are supported:
0	The image is contained in a file on disk.
10	The image is contained in the resouce of a

	picture library. When this type is used, the *PictureName$* parameter must be specified with the Begin Dialog statement.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words of *PictureName$* are used.
style	Specifies whether the picture is drawn within a 3D frame. It can be any of the following values:
0	Draw the picture control with a normal frame.
1	Draw the picture control with a 3D frame.

If omitted, then the picture control is drawn with a normal frame.

The picture control extracts the actual image from either a disk file or a picture library. In the case of bitmaps, both 2- and 16-color bitmaps are supported. In the case of WMFs, Symantec Basic supports the Placeable Windows Metafile.

If *PictureName$* is a zero-length string, then the picture is removed from the picture control, freeing any memory associated with that picture.

Examples: This first example shows how to use a picture from a file.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        Picture 8,4,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog

    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

This second example shows how to use a picture from a picture library with a 3D frame.

```
Sub Main()
    Begin Dialog LogoDialogTemplate
16,31,288,76,"Introduction",,"pictures.dll"
        OKButton 240,8,40,14
        Picture 8,4,224,64,"CompanyLogo",10,.Logo,1
    End Dialog

    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog( )](#)

[Dialog](#)

[DropListBox](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog...End Dialog](#)

[PictureButton](#)

[DlgSetPicture](#)

PictureButton   Statement

Description: Creates a picture button control in a dialog box template.

Syntax: PictureButton *X*,*Y*,*width*,*height*,*PictureName$*,*PictureType* [,.*Identifier*]

Comments: Picture button controls behave very much like a push button controls. Visually, picture buttons are different than push buttons in that they contain a graphic image imported either from a file or from a picture library.

The PictureButton statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, *height*	Integer coordinates specifying the dimensions of the control in dialog units.
PictureName$	String containing the name of the picture. If *PictureType* is 0, then this name specifies the name of the file containing the image. If *PictureType* is 10, then *PictureName$* specifies the name of the image within the resource of the picture library.
PictureType	Integer specifying the source for the image. The following sources are supported:
0	The image is contained in a file on disk.
10	The image is contained in the resouce of a picture library. When this type is used, the *PictureName$* parameter must be specified with the Begin Dialog statement.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

The picture button control extracts the actual image from either a disk file or a picture library, depending on the value of *PictureType*. The supported picture formats vary from platform to platform.

If *PictureName$* is a zero-length string, then the picture is removed from the picture button control, freeing any memory associated with that picture.

Examples:

This first example shows how to use a picture from a file.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog

    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

This second example shows how to use a picture from a picture library.
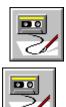
```
Sub Main()
    Begin Dialog LogoDialogTemplate
16,31,288,76,"Introduction",,"pictures.dll"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"CompanyLogo",10,.Logo
    End Dialog

    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog( )](#)

[Dialog](#)

[DropListBox](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[PushButton](#)

[Text](#)

[TextBox](#)

[Begin Dialog...End Dialog](#)

[Picture](#)

[DlgSetPicture](#)

PI   **Constant**

Description: This numeric constant represents the ratio of the circumference to the diameter of a circle. Its value is: 3.14159265358979323846264383279. PI can also be determined using the following formula: 4 * Atn(1).

Syntax: PI

Example: The following example determines the circumference of a circle given the radius.

```
Function Circumference(radius As Integer) As Double
    Circumference = 2 * PI * radius
End Function
```

Pmt   **Function**

Description:	Returns the payment for an annuity based on periodic fixed payments and a constant rate of interest.
Syntax:	Pmt(*Rate,NPer,Pv,Fv,Due*)
Comments:	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The Pmt function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the periods are given in months, be sure to normalize annual rates by dividing them by 12.
NPer	Double representing the total number of payments in the annuity.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

Rate and *NPer* must be expressed in the same units. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example: This example calculates the payment necessary to repay a $1,000.00 loan over 36 months at an annual rate of 10%. Payments are due at the beginning of the period.

```
Sub Main()
    X = Pmt((.1/12),36,1000.00,0,1)
    Msg$ = "The payment to amortize $1,000 over 36 months @ 10% is: "
```

```
        MsgBox Msg + Format$(X,"Currency")
End Sub
```

See Also

[IPmt](#)
[NPer](#)
[PPmt](#)
[Rate](#)

PopupMenu( )   Function

Description: This function creates a popup menu at the current mouse position using the elements from an array as menu items. An empty string in the array results in a separator bar in the menu. Only one popup menu can be displayed at a time. A run-time error occurs if another script executes this function while a popup menu is visible. The function returns an integer indicating the subscript of the selected item. It returns an integer one less than the lower-bound for the subscripts if the user selects the Cancel button.

Syntax: PopupMenu(menuItems)

Parameter: menuItems

The name of a one-dimensional string array of menu items.

Example: The following use of PopupMenu( ) displays a list of applications. The fourth array element is empty, so a bar separates the utilities from the word processors.

```
Dim MyMenu$(1 To 6)
MyMenu(1) = "Norton Disk Doctor"
MyMenu(2) = "Norton Speed Disk"
MyMenu(3) = "Norton Diagnostics"
MyMenu(5) = "Microsoft Word"
MyMenu(6) = "WordPerfect"
Users_Choice = PopupMenu(MyMenu)
```

See Also

[SelectBox( )](#)

PPmt

Function

Description: Calculates the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax: PPmt(*Rate*,*Per*,*NPer*,*Pv*,*Fv*,*Due*)

Comments: An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The PPmt function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period.
Per	Double representing the number of payment periods. *Per* can be no less than 1 and no greater than *NPer*.
NPer	Double representing the total number of payments in your annuity.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
Due	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period; if it is 1, then payments are due at the start of each period.

Rate and *NPer* must be in the same units to calculate correctly. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Negative values represent payments paid out, whereas positive values represent payments received.

Example: This example calculates the principal paid during each year on a loan of $1,000.00 with an annual rate of 10% for a period of 10 years. The result is displayed as a table containing the following information: payment, principal payment, principal balance.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Pay = Pmt(.1,10,1000.00,0,1)
    Msg$ = "Amortization table for 1,000" + crlf + "at 10% annually for"
    Msg = Msg + " 10 years: " + crlf + crlf
    Bal = 1000.00
    For Per = 1 to 10
       Prn = PPmt(.1,Per,10,1000,0,0)
       Bal = Bal + Prn
       Msg = Msg + Format$(Pay,"Currency") + "  " + Format$
(Prn,"Currency")
       Msg = Msg + "  " + Format$(Bal,"Currency") + crlf
    Next Per
    MsgBox Msg
End Sub
```

See Also

[IPmt](#)
[NPer](#)
[Pmt](#)
[Rate](#)

Print#   Statement

Description:	Writes data to a sequential disk file.
Syntax:	Print [#]*filenumber*, [[{Spc(*n*) | Tab(*n*)}][*expressionlist*][{;|,}]]
Comments:	The *filenumber* parameter is a number that is used by Symantec Basic to refer to the open file-the number passed to the Open statement.

The following table describes how data of different types is written:

Data Type	Description
String	Printed in its literal form, with no enclosing quotes.
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
Boolean	Printed as "True" or "False".
Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is concistent with the "general date" format understood by the Format/Format$ functions).
Empty	Nothing is printed.
Null	Prints "Null".
User-defined errors	Printed to files as "Error *code*", where *code* is the value of the user-defined error. The word "Error" is not translated.

Each expression in *expressionlist* is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.

If the last expression in the list is not followed by a comma or a semicolon, then an end-of-line is printed to the file. If the last expression ends with a semicolon, no end-of-line is printedthe next Print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.

The Write statement always outputs information ending with an end-of-line. Thus, if a Print

statement is followed by a Write statement, the file pointer is positioned on a new line.

The Print statement can only be used with files that are opened in Output or Append mode.

The Tab and Spc functions provide additional control over the file position. The Tab function moves the file position to the specified column, whereas the Spc function outputs the specified number of spaces.

In order to correctly read the data using the Input# statement, you should write the data using the Write statement.
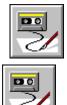
Examples:

```
Sub Main()
    'This example opens a file and prints some data.
    Open "test.dat" For Output As #1
    i% = 10
    s$ = "This is a test."
    Print #1,"The value of i=";i%,"the value of s=";s$

    'This example prints the value of i% in print zone 1 and s$ in
    'print zone 3.
    Print #1,i%,,s$

    'This example prints the value of i% and s$ separated by ten spaces.
    Print #1,i%;Spc(10);s$

    'This example prints the value of i in column 1 and s$ in column 30.
    Print #1,i%;Tab(30);s$

    'This example prints the value of i% and s$.
    Print #1,i%;s$,
    Print #1,67
    Close #1
End Sub
```

See Also

[Open](#)
[Put](#)
[Write #](#)

Print

Statement

Description: This statement writes data to a file or to a viewport window. Printing information to a viewport window is a convenient way to output debugging information. Strings are written in their literal form, with no enclosing quotation marks. Numbers are written with an initial space reserved for a negative sign. An empty space means the number is positive. There is also a space after each number. Single-precision numbers are printed with 7 significant digits; double-precision numbers are printed with 15 or 16.

Syntax: Print [#fileNum], expr {,|;} [expr{,|;}]...

Expressions are separated by commas (,) or semicolons (;). The last expression can be followed by either or neither.

Parameters: fileNum

The integer assigned to the file with the Open statement when it was opened in output or append mode. If omitted, the output is sent to an open viewport window. If there is no open viewport window, the Print statement is ignored.

expr

String or numeric expression usually delimited with a comma or semicolon.

,

Indicates that the next expression is to be written into the next print zone. A new print zone is defined every 14 spaces. A comma moves the file pointer to the next print zone, so the first character of the next expression is written in the next print zone. Using the empty string as the expression for a print zone bypasses that zone.

;

Indicates that the next expression is to be written immediately after the current expression. A semicolon does not move the file pointer; so the next expression should follow immediately after the current expression.

If neither a comma nor a semicolon follows the last expression, a carriage-return/linefeed is written to the file. This positions the file pointer at the beginning of the next line.

Examples: The following example prints the squares of the first ten positive numbers all on the same line of the open file.

```
Open "testfile" For Output As #1
For i = 1 To 10
    'The semicolon forces the next print to Print
    'immediately after
    Print #1, i * i;
Next i
    The file contains "1 4 9 16 25 36 49 64 81 100."
```

The next example opens a viewport window in the upper-left corner of the screen and continuously displays the current time. When the time is updated, the window is cleared so that the new time always appears at the beginning of the window.

```
Dim lastTime$

ViewportOpen "Time", 0, 0, 100, 70

Do
    If lastTime$ <> Time$( ) Then
            ViewportClear
            lastTime$ = Time$( )
            Print Time$( )
    End If
Loop
```

PrinterGetOrientation( ) Function

Description: This function returns the numeric constant ebPortrait if the page orientation for the printer is set to portrait, or ebLandscape if the page orientation is set to landscape. The default printer is the printer specified in the device= line in the [windows] section of the WIN.INI file.

Syntax: PrinterGetOrientation( )

Example: The following example determines whether the current page orientation is landscape or portrait.

```
If PrinterGetOrientation( ) = ebLandscape Then
    MsgBox "Landscape"
Else
    MsgBox "Portrait"
End If
```

See Also

PrinterSetOrientation

PrinterSetOrientation Statement

Description: This statement sets the page orientation of the default printer,
the printer specified in the device= line in the [windows] section of the WIN.INI file.

Syntax: PrinterSetOrientation setting

Parameters: setting

A numeric expression that determines the page orientation for printed documents. It can be either of the following constants: ebPortrait or ebLandscape.

Example: The following example asks the user for a page orientation and adjusts the printer accordingly.

```
If AnswerBox("Orientation?", "Portrait", "Landscape") = 1 Then
    PrinterSetOrientation ebPortrait
Else
    PrinterSetOrientation ebLandscape
End If
```

See Also

[PrinterGetOrientation( )](#)

PrintFile( )   Function

Description:	This function uses the Windows 3.1 shell functions that cause an application to print a file. The application must be registered with Windows, and the file's extension must be associated with the application.
	This function is only available under Windows 3.1.
Syntax:	PrintFile(filename)
Parameter:	filename
	A string expression containing the complete or relative pathname to the file you want to print. An error occurs if the file does not exist.
Example:	The following line prints the file FOODMENU.DOC.

```
result% = PrintFile("C:\FOODMENU.DOC")
```

See Also

[Shell( )](#)

Private Statement

Description: Declares a list of private variables and their corresponding types and sizes.

Syntax: Private *name* [(*subscripts*)] [As *type*] [,*name* [(*subscripts*)] [As *type*]]...

Comments: Private variables are global to every Sub and Function within the currently executing script.

If a type-declaration character is used when specifying *name* (such as %, @, &, $, or !), the optional [As *type*] expression is not allowed. For example, the following are allowed:

```
Private foo As Integer
Private foo%
```

The *subscripts* parameter allows the declaration of arrays. This parameter uses the following syntax:

```
[lower To] upper [,[lower To] upper]...
```

The *lower* and *upper* parameters are integers specifying the lower and upper bounds of the array. If *lower* is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.

The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

```
Private a()
```

The *type* parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.

If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Private name As String * length
```

where *length* is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0

Long	0
Double	0.0
Single	0.0
Currency	0.0
Object	Nothing
Date	December 31, 1899 00:00:00
Boolean	False
Variant	Empty
String	"" (zero-length string)
User-defined type	Each element of the structure is given a default value, as described above.
Arrays	Each element of the array is given a default value, as described above.

Example: See Public (statement).

See Also

[Dim](#)
[ReDim](#)
[Public](#)
[Option Base](#)

Public

Statement

Description:	Declares a list of public variables and their corresponding types and sizes.
Syntax:	Public *name* [(*subscripts*)] [As *type*] [,*name* [(*subscripts*)] [As *type*]]...
Comments:	Public variables are global to all Subs and Functions in all scripts.

If a type-declaration character is used when specifying *name* (such as %, @, &, $, or !), the optional [As *type*] expression is not allowed. For example, the following are allowed:

```
Public foo As integer
Public foo%
```

The *subscripts* parameter allows the declaration of arrays. This parameter uses the following syntax:

> [*lower* To] *upper* [,[*lower* To] *upper*]...

The *lower* and *upper* parameters are integers specifying the lower and upper bounds of the array. If *lower* is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.

The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

```
Public a()
```

The *type* parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.

If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.

For compatibility, the keyword Global is also supported. It has the same meaning as Public.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

> **Public** *name* As String * *length*

where *length* is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value

Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Date	December 31, 1899 00:00:00
Object	Nothing
Boolean	False
Variant	Empty
String	"" (zero-length string)
User-defined type	Each element of the structure is given a default value, as described above.
Arrays	Each element of the array is given a default value, as described above.

Sharing Variables

When sharing variables, you must ensure that the declarations of the shared variables are the same in each script that uses those variables. If the public variable being shared is a user-defined structure, then the structure definitions must be exactly the same.

Example: This example uses a subroutine to calculate the area of ten circles and displays the result in a dialog box. The variables R and Ar are declared as Public variables so that they can be used in both Main and Area.

```
Const crlf$ = Chr$(13) + Chr$(10)

Public R#,Ar#

Sub Area()
    Ar = (R ^ 2) * Pi
End Sub

Sub Main()
    Msg$ = Null
    For X = 1 To 10
       R = X
       Area
       Msg = Msg + Str$(R) + " : " + Str$(Ar) + crlf
    Next x
    MsgBox Msg
End Sub
```

See Also

[Dim](#)
[ReDim](#)
[Private](#)
[Option Base](#)

PushButton   Statement

Description:	This statement can appear only within a Begin Dialog...End Dialog construct. It defines a command button for a dialog box template. Each user-defined command button has a positive integer associated with it. The first button declared in the construct is 1, the second is 2, and so forth.
Syntax:	PushButton x, y, width, height, name
Parameters:	x, y
	The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the command button in dialog units.
	width, height
	The integers indicating the width and height of the command button in dialog units.
	name
	A string variable or literal which specifies the name of the command button.
Example:	The following example displays a dialog box containing four buttons labeled with the compass directions and arranged in a circle.

```
The buttons return 1, 2, 3, and 4 respectively.
Begin Dialog DirectionsDialog 16,32,122,119, "Directions"
    PushButton 50,6,21,21, "north"
    PushButton 93,48,21,21, "east"
    PushButton 50,91,21,21, "south"
    PushButton 8,48,21,21, "west"
End Dialog
```

See Also

Put   **Statement**

Description:	Writes data from the specified variable to a Random or Binary file.
Syntax:	Put [#] *filenumber*, [*recordnumber*], *variable*
Comments:	The Put statement accepts the following parameters:

Parameter	Description
filenumber	Integer representing the file to be written to. This is the same value as returned by the Open statement.
recordnumber	Long specifying which record is to be written to the file.

For Binary files, this number represents the first byte to be written starting with the beginning of the file (the first byte is 1). For Random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647.

If the *recordnumber* parameter is omitted, the next record is written to the file (if no records have been written yet, then the first record in the file is written). When *recordnumber* is omitted, the commas must still appear, as in the following example:

```
Put #1,,recvar
```

If *recordlength* is specified, it overrides any previous change in file position specified with the Seek statement.

The *variable* parameter is the name of any variable of any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are written to the file.
Long	4 bytes are written to the file.
String (variable-length)	In Binary files, variable-length strings are written by first determining the specified string variable's length, then writing that many bytes to the file.
	In Random files, variable-length strings are written by first writing a 2-byte length, then writing that many characters to the file.

String (fixed-length)	Fixed-length strings are written to Random and Binary files in the same way: the number of characters equal to the string's declared length are written.
Double	8 bytes are written to the file (IEEE format).
Single	4 bytes are written to the file (IEEE format).
Date	8 bytes are written to the file (IEEE double format).
Boolean	2 bytes are written to the file (either -1 for True or 0 for False).
Variant	A 2-byte VarType is written to the file followed by the data as described above. With variants of type 10 (user-defined errors), the 2-byte VarType is followed by a 2-byte unsigned integer (the error value), which is then followed by 2 additional bytes of information
	The exception is with strings, which are always preceded by a 2-byte string length.
User-defined types	Each member of a user-defined data type is written individually.
	In Binary files, variable-length strings within user-defined types are written by first writing a 2-byte length followed by the string's content. This storage is different than variable-length strings outside of user-defined types.
	When writing user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.
Arrays	Arrays cannot be written to a file using the Put statement.
Objects	Object variables cannot be written to a file using the Put statement.

With Random files, a runtime error will occur if the length of the data being written exceeds the record length (specified as the *reclen* parameter with the Open statement). If the length of the data being written is less than the record length, the entire record is written along with padding (whatever data happens to be in the I/O buffer at that time). With Binary files, the data elements are written contiguously: they are never separated with padding.

Example: This example opens a file for random write, then writes ten records into the file with the values 10-50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "test2.dat" For Random Access Write As #1
    For X% = 1 To 10
```

```
        Y% = X * 10
        Put #1,X,Y
    Next X
    Close
    Pstr$ = ""
    Open "test2.dat" For Random Access Read As #1
    For Y = 1 To 5
        Get #1,y,X
        Pstr = Pstr + "Record " + Str$(Y) + ": " + Str$(X) + crlf
    Next Y
    MsgBox Pstr
    Close
End Sub
```

See Also

[Open](#)
[Put](#)
[Write #](#)
[Print#](#)

Pv   **Function**

Description: Calculates the present value of an annuity based on future periodic fixed payments and a constant rate of interest.

Syntax: Pv(*Rate*,*NPer*,*Pmt*,*Fv*,*Due*)

Comments: The Pv function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. When used with monthly payments, be sure to normalize annual percentage rates by dividing them by 12.
NPer	Double representing the total number of payments in the annuity.
Pmt	Double representing the amount of each payment per period.
Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be 0.
Due	Integer indicating when the payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

Rate and *NPer* must be expressed in the same units. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example: This example demonstrates the present value (the amount you'd have to pay now) for a $100,000 annuity that pays an annual income of $5,000 over 20 years at an annual interest rate of 10%.

```
Sub Main()
    PVal = Pv(.1,20,-5000,100000,1)
    MsgBox "The present value is: " & Format$(PVal,"Currency")
End Sub
```

See Also

[Fv](#)
[IRR](#)
[MIRR](#)
[Npv](#)

Random( ) **Function**

Description: This function returns a random number within the range specified by the parameters.

Syntax: Random(min, max)

Parameters: min, max

The numeric expressions indicating the lowest and highest possible values for the random number.

Example: The following call to Random( ) could be used to simulate the roll of a die.

```
'Generate a random number between 1 and 6
rollOfDie = Random(1,6)
```

See Also

Randomize

Randomize

Statement

Description: This statement initializes the random number generator with a new seed. Repeating the seed allows you to repeat a sequence of random numbers.

Syntax: Randomize [(seed)]

Parameters: seed

A long number. The default is the value of the system clock.

Example: In this example, the seed for the random number generator is set to the current clock value and then a number is requested from 1 to 100.

```
Randomize
aRandomNumber = Random(1,100)
     In this example, the seed for the random number generator is set to
123 and then a number is requested from 1 to 100.
Randomize 123
aRandomNumber = Random(1,100)
```

See Also

[Random( )](#)
[Rnd( )](#)

Rate   **Function**

Description:	Returns the rate of interest for each period of an annuity.
Syntax:	Rate(*NPer*,*Pmt*,*Pv*,*Fv*,*Due*,*Guess*)
Comments:	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The Rate function requires the following parameters:

Parameter	Description
NPer	Double representing the total number of payments in the annuity.
Pmt	Double representing the amount of each payment per period.
Pv	Double representing the present value of your annuity. In a loan situation, the present value would be the amount of the loan.
Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be zero.
Due	Integer specifying when the payments are due for each payment period. A 0 indicates payment at the end of each period, whereas a 1 indicates payment at the start of each period.
Guess	Double specifying a guess as to the value the Rate function will return. The most common guess is .1 (10 percent).

Positive numbers represent cash received, whereas negative values represent cash paid out.

The value of *Rate* is found by iteration. It starts with the value of *Guess* and cycles through the calculation adjusting *Guess* until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, *Rate* fails, and the user must pick a better guess.

Example: This example calculates the rate of interest necessary to save $10,000 by paying $550 each year for 10 years. The guess rate is 10%.

```
Sub Main()
```

```
        R# = Rate(10,-550,000,10000,1,.1)
        MsgBox "The rate required is: " + Format$(R,"Percent")
End Sub
```

See Also

[IPmt](#)
[NPer](#)
[Pmt](#)
[PPmt](#)

ReadINI$( )　　　　　　　　Function

Description:	This function returns a string containing the value of the specified entry.
Syntax:	ReadINI$(section, entry[, filename])
Parameters:	section
	A string expression containing the name of a section (without the brackets) in the specified .INI file.
	entry
	A string expression containing the name of the specific entry to read from the section.
	filename
	A string expression containing the complete or relative pathname of an .INI file. If no path is specified, the Windows directory is searched. The default is WIN.INI.
Example:	The following example reads the value from the load= entry in the [windows] section of the WIN.INI file.

```
value$ = ReadINI$("windows", "load")
```

See Also

[WriteINI](#)
[ReadINISection](#)

ReadINISection   Statement

Description: This statement reads all items from a section of the specified .INI file. Use the LBound( ) and UBound( ) functions to determine the size of the string array when it is filled.

Syntax: ReadINISection section, entries[, filename]

Parameters: section

A string expression containing the name of a section (without the brackets) in the specified .INI file.

entries

The name of the one-dimensional string array that will contain the entries read from the specified section along with their values.

filename

A string expression containing the complete or relative pathname of an .INI file. If no path is specified, the Windows directory is searched. The default is WIN.INI.

Example: The following example reads all the entry names in the "windows" section of the WIN.INI file.

```
'Declare array to hold entries
Dim entries$( )

'Read the entries from the "windows"
'section of WIN.INI
ReadINISection "windows", entries
```

See Also

[ReadINI$( )](#)
[WriteINI](#)

ReDim   **Statement**

Description: This statement redimensions an array by specifying new upper and lower bounds for it. The default lower bound is 0 (or the value set using the Option Base statement). Each element of the array is re-initialized to zero or the empty string.

Syntax: Redim [Preserve] variablename (subscriptRange) [As Type].....

Parameters: Preserve

The Preserve attribute lets you reallocate an dynamic array while not re-initializing it.

subscriptRange

Numeric expressions for the new upper and lower bounds for each dimension of the array using the following syntax:
          [lower To] upper [,[lower To] upper]...

Type

Specifies the type of the array. This can be used when a type declarator is not appended to the name.
If neither is used, the type is as determined by a Deftype statement or, by default, the type is an integer.

Example: Suppose the Main subroutine can call either of two subroutines to use an array. One subroutine needs three dimensions in the array and the other needs only two. You can redimension the array during the script to fit either subroutine. For example, you can use the following:

```
ReDim Array2(0 To 8,6 To 10)
```

See Also

[Dim](#)
[Public](#)
[Private](#)
[ArrayDims( )](#)
[LBound( )](#)
[UBound( )](#)

Rem   **Statement**

Description:	This reserved word indicates that the entire line is a comment to be ignored by the compiler.
Syntax:	Rem
Example:	The following example shows how to use Rem.

```
REM This script performs...
```

See Also

Reset   **Statement**

Description: This statement closes all open files and empties all I/O buffers.

Syntax: Reset

Example: In the following example, the single Reset statement closes both open files.

```
Open "testfil1" As #1
Open "testfil2" As #2
Reset
```

See Also

[Close](#)
[Open](#)

Resume   Statement

Description: This statement ends an error handling routine and continues script execution. It also resets the error value to 0.

Syntax: Resume {[0] | Next | label}

0   Continues execution with the statement that caused the error condition.

Next Continues execution with the statement following the statement that caused the error condition.

label Continues execution at the specified label.

Example: The following example sends all errors to the same label. The statements between the label and the Resume statement display the error numbers and messages that Symantec Basic normally displays when a run-time error terminates a script. Err( ) is a predefined function that returns the value of the most recent error. Similarly, the Error$( ) function returns the error message associated with the most recent error.

```
On Error GoTo MessageDisplay
...
cmd$ = "[CreateGroup(" + quoted(Setup.GroupName) + ")]"
DDEExecute channel, cmd$
...
'The Exit Sub keeps you from executing
'the error routine when no error occurs
Exit Sub
'This routine can be used for all
'errors while you are
'debugging
'It may help you fix more than one
'error at a time
MessageDisplay:
MsgBox Str$(Err( )) + Error$( )
Resume Next
```

See Also

[On Error](#)

Return

Statement

Description:	This statement transfers execution control to the statement following the most recent GoSub statement. A run-time error occurs if there is no GoSub statement in the subroutine.
Syntax:	Return
Example:	The following example uses the GoSub statement with a Return statement to repeat the same sequence of statements throughout a script.

```
Sub Main
...
'Write standard header to first file
GoSub PrepareHeader
...
'Write standard header to second file
GoSub PrepareHeader
...
'Write standard header to third file
GoSub PrepareHeader
...
'The Exit Sub keeps you from executing
'The PrepareHeader routine unless you are sent to it
Exit Sub

PrepareHeader:
    'sequence of statements that write
    'header lines to a file
Return
End Sub
```

See Also

GoSub

Right$( )   Function

Description: This function returns the specified number of characters from a string starting with the rightmost character. If charNum is greater than or equal to the length of the string, the entire string is returned.
If charNum is set to 0, an empty string is returned.

Syntax: Right$(exprS, charNum)

Parameters: exprS

A string expression.

charNum

An integer representing the number of characters to return. The default is 1.

Example: In the following example, assume that a percent sign (%) separates the first name from the last name in the string Name.

```
'Find the percent sign
Position% = InStr(Name, "%")
'Retain only the rightmost characters
LastName = Right$(Name, Position + 1)
```

See Also

Left$( )

RmDir   **Statement**

Description: This statement deletes the specified directory. Only empty directories can be deleted. You can use the Kill statement to delete files.

Syntax: RmDir dir

Parameter: dir

A string expression containing the complete or relative pathname of a directory.

Examples: The following example removes the directory named ASDF from the current drive.

```
RmDir "asdf"
    The next example removes the directory named ASDF from the
C drive.
RmDir "c:asdf"
```

See Also

[ChDir](#)
[ChDrive](#)
[CurDir$( )](#)
[Dir$( )](#)
[MkDir](#)

Rnd( )   **Function**

Description: This function returns a single-precision random number between 0 and 1 based on the value of exprN. If exprN is less than 0, the same number is always returned. If exprN is equal to 0, the last number generated is returned. Otherwise, if exprN is greater than 0 or omitted, the next new random number is returned.

Syntax: Rnd[(exprN)]

Parameter: exprN

A numeric expression used in calculating a random number. The default is a number greater than 0.

Example: The following example uses the Rnd( ) function to randomly pick a whole number from a specified range.

```
'Range is from 0 to n
randomNumber% = Rnd * n
```

See Also

[Random( )](#)
[Randomize](#)

RSet

Statement

Syntax: RSet destvariable = source

Description: Copies the source string source into the destination string destvariable.

Comments: If source is shorter in length than destvariable, then the string is right-aligned within destvariable and the remaining characters are padded with spaces. If source is longer in length than destvariable, then source is truncated, copying only the leftmost number of characters that will fit in destvariable. A runtime error is generated if source is Null.

The destvariable parameter specifies a String or Variant variable. If destvariable is a Variant containing Empty, then no characters are copied. If destvariable is not convertible to a String, then a runtime error occurs. A runtime error results if destvariable is Null.

Example:

```
'This example replaces a 40-character string of asterisks (*) with
'an RSet and LSet string and then displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim msg,tmpstr$
    tmpstr$ = String(40,"*")
    msg = "Here are two strings that have been right-" + crlf
    msg = msg & "and left-justified in a 40-character string."
    msg = msg & crlf & crlf
    RSet tmpstr$ = "Right|"
    msg = msg & tmpstr$ & crlf
    LSet tmpstr$ = "|Left"
    msg = msg & tmpstr$ & crlf
    MsgBox msg
End Sub
```

Platform(s):   All.

See Also

LSet

RTrim$( )   **Function**

Description: This function returns the specified string with any trailing spaces removed.

Syntax: RTrim$(exprS)

Parameter: exprS

A string expression.

Example: The following example demonstrates the use of RTrim$( ).

```
aString$ = "10 trailing spaces     "
'Now remove the leading spaces
aString = RTrim$(aString)
'aString should now be equal to the string
'"10 trailing spaces"
```

See Also

[LTrim$( )](#)
[Trim$( )](#)

SaveFileName$( )   Function

Description: This function displays the standard dialog box for saving files. It returns a string expression containing the complete pathname for the file that the user selected or an empty string if the user cancels the dialog box.

Syntax: SaveFileName$(name [, extensions])

Parameters: name

A string expression containing the name for the dialog box.

extensions

A string expression that specifies the available file types in the following format:
type:ext[, ext][; type:ext[, ext]]...
Type is a string that identifies the file type, such as "Documents".
Ext is a valid file extension like *.BAT or *.?F?
The default is "All Files:*.*".

Example: The following example uses the SaveFileName$( ) function to locate pictures.

```
FileTypes$ = "All Files:*.*;Bitmaps:*.BMP;Metafiles:*.WMF"
SelectedFile$ = SaveFileName$("Save Picture", FileTypes)
If SelectedFile = "" Then
    MsgBox "No file was selected!"
Else
    MsgBox "The file " + SelectedFile + " was selected."
End If
```

Initially, all the files in the current directory are displayed in the file list of the dialog box. The user can manually type in a filename or select one from the file list. After the user exits the dialog box, the function returns a string value. If the user clicked Cancel, SelectedFile contains an empty string and a message box displays "No file was selected!" Otherwise, SelectedFile contains the complete pathname for the selected file and a message box displays that name.

See Also

[MsgBox](#)
[AskBox$( )](#)
[AskPassword$( )](#)
[InputBox$( )](#)
[OpenFileName$( )](#)
[SelectBox( )](#)
[AnswerBox( )](#)

Second( )　　　　　　　　Function

Description:　This function returns the second of the day encoded in the specified serial parameter. The value returned is an integer ranging from 0 to 59.

Syntax:　Second(serial)

Parameters:　serial

　　　A double-precision number containing the serial time.

Example:　After calling the Now( ) function, you can extract the current second from the date and time.

```
'Get the current date and time
serialDT# = Now( )

'Now extract the value
theSecond% = Second(serialDT)
```

See Also

[Day( )](#)
[Minute( )](#)
[Month( )](#)
[Year( )](#)
[Hour( )](#)
[Weekday( )](#)
[DatePart](#)

Seek

Statement

Description: Sets the position of the file pointer in an open file. Use the Loc( ) function or Seek( ) function to get the current position of the file pointer before moving it.

Syntax: Seek [#]fileNum, position

Parameters: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

position

A numeric expression indicating the new position for the file pointer in the specified file. The first character in the file is at position number 1.

Example: The following example uses the Seek statement to go to the beginning of the third line (position 21 in a file that has 10 characters per line).

```
Open "testfile" For Input As #1
...
'Seek to the third line (position 21)
Seek #1, 21
...
Close #1
```

See Also

[Seek( )](Seek)
[Loc( )](Loc)

Seek( )   **Function**

Description: This function returns a numeric expression indicating the position of the file pointer in the specified file. The first character in the file is at position number 1.

Syntax: Seek(fileNum)

Parameter: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

Example: In the following example, two lines are read. Then the Seek( ) function determines the new position of the file pointer. The Loc( ) function could have been used instead of the Seek( ) function.

```
Open "testfile" For Input As #1

'Read in first line (two items)
Input #1, num34%, strCDE$

'Read in second line (two items)
Input #1, num45%, strDEF$

'Determine the new position of the file pointer
curPos& = Seek(1) 'equivalent to curPos& = Loc(1)
Close #1
```

See Also

[Seek](#)
[Loc( )](#)

Select Case...End Select   Construct

Description: This construct executes a series of statements depending on the value of a specified expression. The statements associated with the first match between testExpr and any of the expressions contained in exprList are executed. The data type of each expression in exprList must be the same as that of testExpr. Multiple expression ranges can be used within a single Case clause, such as Case 1 To 10,12,15, Is > 40.

Syntax: Select Case testExpr
[Case exprList [statements]...
...
[Case Else [statements]...]
End Select

testExpr A numeric or string expression.

exprList Any of the following:
expr [, expr]...
expr To expr
Is relational_operator expr statement

s   A series of executable statements.

Example: In the following example, a Select Case statement decides which sequence of statements to execute based on the value of a one-character string.

```
...
Select Case Grade
    Case "A"
            ...     'sequence of statements
    Case "B" To "D"
            ...     'sequence of statements
    Case Is > "D"
            ...     'sequence of statements
    Case Else
            ...     'sequence of statements
End Select
```

See Also

[Choose (function)](#)
[Switch (function)](#)
[IIf (function)](#)
[If...Then...Else (statement)](#)

SelectBox( )   **Function**

Description: This function displays a dialog box containing a list box of items. The dialog box also has a name, a message for the user, and OK, Cancel, and Help push buttons. It returns an integer indicating the subscript for the item that the user selects. It returns a number that is one less than the lower bound for subscripts if the user selects the Cancel button. The Help button is not functional. It is included for compatibility with other BASICs.

Syntax: SelectBox(name , message, items)

Parameters: name

A string expression containing the name of the dialog box.

message

A string expression for the user to respond to.

items

A one-dimensional string array containing the items to list in the dialog box's list box.

Example: The following call to SelectBox( ) displays a list of applications.

```
Dim Title$
Dim Message$
Dim MyMenu$(1 To 5)
Title = "Applications"
Message = "Select an application."
MyMenu(1) = "Norton Disk Doctor"
MyMenu(2) = "Norton Speed Disk"
MyMenu(3) = "Norton Diagnostics"
MyMenu(4) = "Microsoft Word"
MyMenu(5) = "WordPerfect"
Users_Choice = SelectBox(Title, Message, MyMenu)
```

See Also

[MsgBox( )](#)
[AskBox$( )](#)
[AskPassword$( )](#)
[InputBox$( )](#)
[OpenFileName$( )](#)
[SaveFileName$( )](#)
[AnswerBox( )](#)

SendKeys   Statement

Description:	This statement sends the specified keys to the active application. must be inserted manually into a script.
Syntax:	SendKeys keyStr [, wait] [, timeout]
Parameters:	keyStr
	A string expression containing full keystrokes to be sent.
	wait
	A numeric expression that can be TRUE or FALSE, included for compatibility with other BASICs. Symantec Basic always acts as if wait were TRUE and waits for all the keystrokes to be performed before continuing script execution.
	timeout
	The time in milliseconds to wait for the keystroke event. The default is zero (one attempt).
Examples:	All three examples have the same functionality. Only after all the keys are sent is the next statement executed.

```
SendKeys "{PRTSC}", FALSE
SendKeys "{PRTSC}"
SendKeys "{PRTSC}", TRUE
```

See Also

[DoKeys](#)

Set Statement

Syntax 1:	Set *object_var* = *object_expression*
Syntax 2:	Set *object_var* = New *object_type*
Syntax 3:	Set *object_var* = Nothing
Description:	Assigns a value to an object variable.
Comments:	**Syntax 1**

The first syntax assigns the result of an expression to an object variable. This statement does not duplicate the object being assigned but rather copies a reference of an existing object to an object variable.

The *object_expression* is any expression that evaluates to an object of the same type as the *object_var*.

With data objects, Set performs additional processing. When the Set is performed, the object is notified that a reference to it is being made and destroyed. For example, the following statement deletes a reference to object A, then adds a new reference to B.

```
Set A = B
```

In this way, an object that is no longer being referenced can be destroyed.

Syntax 2

In the second syntax, the object variable is being assigned to a new instance of an existing object type. This syntax is valid only for data objects.

When an object created using the New keyword goes out of scope (i.e., the Sub or Function in which the variable is declared ends), the object is destroyed.

Syntax 3

The reserved keyword Nothing is used to make an object variable reference no object. At a later time, the object variable can be compared to Nothing to test whether the object variable has been instantiated:

```
Set A = Nothing
    :
If A Is Nothing Then Beep
```

Example: This example creates 2 objects and sets their values.

```
Sub Main()
    dim Document as object
    dim Page as object
    set Document = GetObject("c:\resume.doc")
```

```vba
    set page = Document.ActivePage
    MsgBox page.name
End Sub
```

See Also

SetAttr

Statement

Description: This statement changes the attributes of the specified file to the specified attributes. A run-time error occurs if the file cannot be found.

Syntax: SetAttr filename, fileAttr

Parameters: filename

A string expression containing the complete or relative pathname to a file. It cannot contain wildcards (* and ?).

fileAttr

A numeric expression indicating the file attributes by providing the sum of a subset of the following constants:

0	ebNormal	Normal file,
1	eReadOnly	Read-only file,
2	ebHidden	Hidden file,
4	ebSystem	System file,
8	ebVolume	Volume label,
16	ebDirectory	Directory,
32	ebArchive	File has changed since last backup,
64	ebNone	File has no attributes.

Examples: The following example makes the AUTOEXEC.BAT file read-only and hidden.

```
SetAttr "C:\AUTOEXEC.BAT", ebReadOnly+ebHidden
    The next example makes the AUTOEXEC.BAT file a normal file.
SetAttr "C:\AUTOEXEC.BAT", ebNormal
```

See Also

[GetAttr( )](#)
[FileAttr( )](#)

Sgn( )   **Function**

Description:	This function determines the sign of a number. It returns 1 if the number is greater than zero, 0 if the number is equal to zero, or -1 if the number is less than zero.
Syntax:	Sgn(exprN)
Parameter:	exprN
	A numeric expression.
Example:	In the following example, the integer indicating the sign of the specified expression is stored in the variable sign.

```
sign% = Sgn(2*3/-1)
```

See Also

[Abs( )](#)

Shell( ) **Function**

Description: The function launches any executable file. If the function is unsuccessful, a run-time error occurs.

This function is equivalent to choosing Run…from the Program Manager or Norton Desktop File menu. The function launches an application and returns the integer that is the application's task ID. The script and the application execute concurrently.

Syntax: Shell(command [, style])

Parameters: command

A string expression that contains either a complete or relative pathname, to the executable file along with any command-line options you want to use.

style

The number specifying the state of the main window after the application is launched.

An application window can be in any of the following states:
1   Normal active window (the default),
2   Minimized active window,
3   Maximized active window,
4   Normal inactive window,
7   Minimized inactive window.

Examples: Assuming that Notepad (NOTEPAD.EXE) is in one of the directories contained in the PATH environment variable, the following example launches Notepad in a maximized active window.

```
taskID = Shell("NOTEPAD.EXE", 3)
```

See Also

[PrintFile( )](#)
[SendKeys](#)

Sin( )   **Function**

Description: This function returns the sine of a specified angle. The value returned is a number of type double.

Syntax: Sin(angle)

Parameters: angle

                   A numeric expression specifying an angle in radians.

Example: The y coordinate of a point on a circle of radius 1 centered at the origin can be found by computing the sine of the angle at which the point lies on the circle.

```
'Calculate the y coordinate of the point
'at 30 degrees
y = Sin(30*PI/180)
```

See Also

[Tan( )](#)
[Cos( )](#)
[Atn( )](#)

Single

Data type

Syntax:	Single
Description:	A data type used to declare variables capable of holding real numbers with up to seven digits of precision.
Comments:	Single variables are used to hold numbers within the following ranges:

Sign	Range
Negative	-3.402823E38 $<=$ *single* $<=$ -1.401298E-45
Positive	1.401298E-45 $<=$ *single* $<=$ 3.402823E38

The type-declaration character for Single is !.

Storage

Internally, singles are stored as 4-byte (32-bit) IEEE values. Thus, when appearing within a structure, singles require 4 bytes of storage. When used with binary or random files, 4 bytes of storage is required.

Each single consists of the following
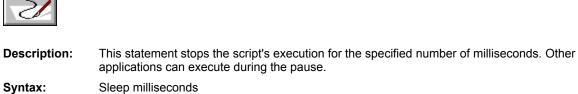
A 1-bit sign

An 8-bit exponent

A 24-bit mantissa

Platform(s):	All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[String (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CSng( )](#)

Sleep **Statement**

Description:	This statement stops the script's execution for the specified number of milliseconds. Other applications can execute during the pause.
Syntax:	Sleep milliseconds
Parameter:	milliseconds
	A numeric expression specifying a time interval in milliseconds.
Example:	In the following example, the script pauses (using a Sleep statement) while an application paints a new window.

```
'Send the keys that cause the window to appear
'Wait 2 seconds to be sure the window is painted
Sleep 2000
'Send more keys...
```
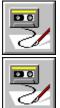
Sln   **Function**

Syntax:	Sln(*Cost*,*Salvage*,*Life*)
Description:	Returns the straight-line depreciation of an asset assuming constant benefit from the asset.
Comments:	The Sln of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.

The formula used to find the Sln of an asset is as follows:

```
(Cost - Salvage Value) / Useful Life
```

The Sln function requires the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset.
Salvage	Double representing the estimated value of the asset at the end of its useful life.
Life	Double representing the length of the asset's useful life.

The unit of time used to express the useful life of the asset is the same as the unit of time used to express the period for which the depreciation is returned.

Example:

```
'This example calculates the straight-line depreciation of an asset
'that cost $10,000.00 and has a salvage value of $500.00 as scrap
'after 10 years of service life.

Sub Main()
    dep# = Sln(10000.00,500.00,10)
    MsgBox "The annual depreciation is: " & Format(dep#,"Currency")
End Sub
```

Platform(s):   All.

See Also

[SYD](#)
[DDB](#)

Space$( )   Function

Description: This function returns a string containing the specified number of spaces.

Syntax: Space$(numSpaces)

Parameter: numSpaces

A positive integer expression.

Examples: The following call to Space$( ) generates a string containing 100 spaces.

```
space100$ = Space$(100)
```

If you are writing records to a file, you can use the Space$( ) function to pad a field with spaces so that your string exactly fits the field.
This example finds the length of LastName and adds the proper number of blank spaces to it before writing it to a file as a field
of 25 characters. For example, the last name Johnson would receive 18 spaces.

```
Length = Len(LastName)
If Length < 25 Then
    LastName = LastName + Space$(25 - Length)
End If
```

See Also

String$( )
Spc

Spc   **Function**

Description: Prints out the specified number of spaces. This function can only be used with the Print and Print# statements.

Syntax: Spc(*numspaces*)

Comments: The *numspaces* parameter is an Integer specifying the number of spaces to be printed. It can be any value between 0 and 32767.

If a line width has been specified (using the Width statement), then the number of spaces is adjusted as follows:

```
numspaces = numspaces Mod width
```

If the resultant number of spaces is greater than width - print_position, then the number of spaces is recalculated as follows:

```
numspaces = numspaces - (width - print_position)
```

These calculations have the effect of never allowing the spaces to overflow the line length. Furthermore, with a large value for column and a small line width, the file pointer will never advance more than one line.

Example: This example displays 20 spaces between the arrows.

```
Sub Main()
ViewportOpen
Print "20 spaces:-->"; Spc(20); "<--"
viewportclose
End Sub
```

See Also

[Tab](#)
[Print](#)
[Print#](#)

Sqr( )   **Function**

Description: This function returns the square root of the specified expression. The value returned is a number of type double.

Syntax: Sqr(exprN)

Parameter: exprN

A numeric expression greater than or equal to 0.

Example: The Pythagorean theorem says that the length of the hypotenuse of a right triangle is equal to the square root of the sum of the squares of the lengths of the other two sides. The following example calculates the length of the hypotenuse given the length of the other two sides.

```
's1 and s2 are the lengths of the other two sides
Function LengthOfHypotenuse#(s1#, s2#)
    LengthOfHypotenuse = Sqr(s1*s1 + s2*s2)
End Function
```

Stop   **Statement**

Description: This statement stops execution of the script and displays the message: "Stopped at line Linenumber", where Linenumber is the line number of the Stop statement. All open files and DDE channels are closed.

Syntax: Stop

Example: The following example gives the user three chances to enter a password correctly. If the password has not been entered correctly, the script is terminated using the Stop statement.

```
i% = 0
Do
    s$ = AskPassword$("Type in the password:")
    If s$ = "password" Then
        Exit Do
    End If
    i = i + 1
    If i = 3 Then
        Stop
    End If
Loop
```

See Also

Exit For
Exit Do
Exit Function
Exit Sub
End

Str$( )   **Function**

Description: This function converts the specified numeric expression to a string. The first character of the string is a space if the number is positive or a minus if the number is negative.

Syntax: Str$(exprN)

Parameter: exprN

A numeric expression.

Example: The following example converts the number 16 to its string equivalent.

```
strOf16$ = Str$(16)
'Result is the string " 16"
```

See Also

Format, Format$
CStr( )

StrComp( )   Function

Description: This function returns an integer indicating whether the string expressions are equal or not:
0
Indicates that the string expressions are equal.
1
Indicates that exprS1 is greater than exprS2.
-1
Indicates that exprS1 is less than exprS2.

Syntax: StrComp(exprS1, exprS2 [, caseSensitive])

Parameters: exprS1, exprS2

The string expressions to be compared.

caseSensitive

The integer 0 or 1, respectively indicating whether the comparison is case sensitive or not. The default is 0 (case sensitive).

Example: The following example compares "apples" and "oranges". The result is -1 because the string "apples" comes before the string "oranges" in ASCII order and is, therefore, less than "oranges".

```
String1$ = "apples"
String2$ = "oranges"
Result = StrComp(String1, String2)
```

See Also

[Like](#)
[Option Compare](#)

String$( )   Function

Description: This function returns a string containing the specified filler character the specified number of times.

Syntax: String$(exprN, {charCode|exprS})

Parameters: exprN

A positive integer expression.

charCode

An integer specifying the ASCII value of a character to fill the string.

exprS

A string expression whose first character will fill the string.

Examples: The ASCII code for the letter "'A" is 65. Both of the following generate a string containing 13 "A" characters.

```
stringOf13A = String$(13,65)     'Using ASCII code
stringOf13A = String$(13,"A")    'Using string
```

If you are writing records to a file, you can use String$( ) to pad a field with filler characters so that your string exactly fits the field. The next example finds the length of LastName and adds the proper number of percent signs to it. For example, the last name Johnson receives 18 characters before it is written to a file as a field of 25 characters. If the length of Last Name is longer than the field, the Left$ function truncates the string.

```
Length = Len(LastName)
If Length < 25 Then
    LastName = LastName + String$(25 - Length, "%")
Else
    LastName = Left$(LastName, 25)
End If
```

See Also

[Space$( )](#)

String

Data type

Syntax:	String
Description:	A data type capable of holding a number of characters.
Comments:	Strings are used to hold sequences of characters, each character having a value between 0 and 255. Strings can be any length up to a maximum length of 32767 characters.

Strings can contain embedded nulls, as shown in the following example:

```
s$ = "Hello" + Chr$(0) + "there"          'String with embedded null
```

The length of a string can be determined using the Len function. This function returns the number of characters that have been stored in the string, including unprintable characters.

The type-declaration character for String is $.

String variables that have not yet been assigned are set to zero-length by default.

Strings are normally declared as variable-length, meaning that the memory required for storage of the string depends on the size of its content. The following BasicScript statements declare a variable-length string and assign it a value of length 5:

```
Dim s As String
s = "Hello"              'String has length 5.
```

Fixed-length strings are given a length in their declaration:

```
Dim s As String * 20
s = "Hello"              'String has length 20 (internally pads with
spaces).
```

When a string expression is assigned to a fixed-length string, the following rules apply:

> If the string expression is less than the length of the fixed-length string, then the fixed-length string is padded with spaces up to its declared length.

> If the string expression is greater than the length of the fixed-length string, then the string expression is truncated to the length of the fixed-length string.

Fixed-length strings are useful within structures when a fixed size is required, such as when passing structures to external routines.

The storage for a fixed-length string depends on where the string is declared, as described in the following table:

Strings Declared	Are Stored
In structures	In the same data area as that of the structure. Local structures are on the

		stack; public structures are stored in the public data space; and private structures are stored in the private data space. Local structures should be used sparingly as stack space is limited.
	In arrays	In the global string space along with all the other array elements.
	Local routines	On the stack. The stack is limited in size, so local fixed-length strings should be used sparingly.
Platform(s):	All.	

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[Variant (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CStr( )](#)

Sub...End Sub Construct

Description: This construct declares a subroutine. When executed, the End Sub statement transfers control back to the calling routine. Subroutines can be recursive.

Syntax: Sub name [parameterList]

 ...
End Sub

name   The name of the subroutine. It must follow BASIC naming conventions, and cannot include a type declarator.

parameterList   List of parameters for the subroutine separated by commas. The syntax is:

parameter [, parameter]...

and the syntax for each parameter is:
[ByVal] parameterName [( )] [As type]

Type is the data type of the parameter being specified. Use As type or a type declarator at the end of parameterName. Use the ByVal keyword to pass a parameter by value. The empty parentheses are for arrays, which must be passed by reference.

Example: The next example declares a subroutine with one parameter.

```
Sub StringPlay (LongString$)
    'A variety of statements that use LongString
End Sub
```

See Also

[Main](#)
[Function...End Function](#)

Switch

Function

Syntax:	Switch(condition1,expression1 [,condition2,expression2 ... [,condition7,expression7]])
Description:	Returns the expression corresponding to the first True condition.
Comments:	The Switch function evaluates each condition and expression, returning the expression that corresponds to the first condition (starting from the left) that evaluates to True. Up to seven condition/expression pairs can be specified.
	A runtime error is generated it there is an odd number of parameters (i.e., there is a condition without a corresponding expression).
	The Switch function returns Null if no condition evaluates to True.
Example:	The following code fragment displays the current operating platform. If the platform is unknown, then the word "Unknown" is displayed.

```
Sub Main()
    Dim a As Variant
    a = Switch(Basic.OS = 0,"Windows 3.1",Basic.OS = 2,"Win32",Basic.OS =
11,"OS/2")
    MsgBox "The current platform is: " & IIf(IsNull(a),"Unknown",a)
End Sub
```

Platform(s):	All.

See Also

[Choose (function)](#)
[IIf (function)](#)
[If...Then...Else (statement)](#)
[Select Case...End Select](#)

SYD   Function

Description: Returns the sum of years' digits depreciation of an asset over a specific period of time.

Syntax: SYD(*Cost*,*Salvage*,*Life*,*Period*)

Comments: The SYD of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.

The formula used to find the SYD of an asset is as follows:

```
(Cost - Salvage_Value) * Remaining_Useful_Life / SYD
```

The SYD function requires the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset.
Salvage	Double representing the estimated value of the asset at the end of its useful life.
Life	Double representing the length of the asset's useful life.
Period	Double representing the period for which the depreciation is to be calculated. It cannot exceed the life of the asset.

To receive accurate results, the parameters *Life* and *Period* must be expressed in the same units. If *Life* is expressed in terms of months, for example, then *Period* must also be expressed in terms of months.

Example: In this example, an asset that cost $1,000.00 is depreciated over ten years. The salvage value is $100.00, and the sum of the years' digits depreciation is shown for each year.

```
Sub Main()
    For X = 1 To 10
        Dep# = SYD(1000,100,10,X)
        Msg$ = Msg + "Year" + Str$(X) +" Dep: " + Format$(Dep,"Currency")
+ Chr$(13)
    Next x
    MsgBox Msg
End Sub
```

See Also

[Sln (function)](#)
[DDB](#)

Tab   **Function**

Description: Prints the number of spaces necessary to reach a given column position.

Syntax: Tab(*column*)

Comments: This function can only be used with the Print and Print# statements.

The *column* parameter is an Integer specifying the desired column position to which to advance. It can be any value between 0 and 32767 inclusive.

Rule 1: If the current print position is less than or equal to *column*, then the number of spaces is calculated as:

```
column - print_position
```

Rule 2: If the current print position is greater than *column*, then *column* - 1 spaces are printed on the next line.

If a line width is specified (using the Width statement), then the column position is adjusted as follows before applying the above two rules:

```
column = column Mod width
```

The Tab function is useful for making sure that output begins at a given column position, regardless of the length of the data already printed on that line.

Example: This example prints three column headers and three numbers alligned below the column headers.

```
Sub Main()
    ViewportOpen
    Print "Column1"; Tab(10); "Column2"; Tab(20); "Column3"
    Print Tab(3); "1"; Tab(14); "2"; Tab(24); "3"
End Sub
```

See Also

[Spc](Spc)
Print
Print#

Tan( )   **Function**

Description: This function returns the tangent of the specified angle. The value returned is a number of type double.

Syntax: Tan(angle)

Parameters: angle

A numeric expression containing the number of radians in an angle.

Example: The tangent of an angle is equal to the angle's sine divided by its cosine. The following example confirms this.

```
'Calculate the tangent of 30 degrees
tan30 = Tan(30*PI/180)
'The result of the previous calculation
'equals the result of the following calculation
sin30_cos30 = Sin(30*PI/180)/Cos(30*PI/180)
```

See Also

[Sin( )](#)
[Cos( )](#)
[Atn( )](#)

Text   **Statement**

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a text control for a dialog box template.

Syntax: Text x, y, width, height, name

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the text control in dialog units.

width, height

The integers indicating the width and height of the static text control in dialog units.

name

A string variable or literal which specifies the text.

Example: The following example displays a dialog box containing a text control and a text box for entering a serial number. When the Dialog statement ends, a message box displays the serial number that was entered.

```
Begin Dialog SerialNumDialog 16,32,110,33, "Serial Number"
    Text 5,6,57,8, "Serial Number:"
    TextBox 5,15,51,12, .SerialNumber
    OKButton 64,13,41,14
End Dialog

Dim dialog1 As SerialNumDialog
Dialog dialog1

'Display the entered serial number
MsgBox dialog1.SerialNumber
```

See Also

[CancelButton](#)
[CheckBox](#)
[ComboBox](#)
[Dialog( )](#)
[Dialog](#)
[DropListBox](#)
[GroupBox](#)
[ListBox](#)
[OKButton](#)
[OptionButton](#)
[OptionGroup](#)
[Picture](#)
[PushButton](#)
[TextBox](#)
[Begin Dialog...End Dialog](#)
[PictureButton](#)

TextBox

Statement

Description: This statement can appear only within a Begin Dialog…End Dialog construct. It defines a text box for a dialog box template.

Syntax: TextBox x, y, width, height, .field

Parameters: x, y

The integers indicating the horizontal and vertical distances from the upper-left corner of the dialog box to the upper-left corner of the text box in dialog units.

width, height

The integers indicating the width and height of the text box in dialog units.

.field

A string variable used to set and/or retrieve the contents of the text box.

Example: The following example displays a dialog box containing a text control and a text box for entering a serial number. When the Dialog statement ends, a message box displays the serial number that was entered.

```
Begin Dialog SerialNumDialog 16,32,110,33, "Serial Number"
    Text 5,6,57,8, "Serial Number:"
    TextBox 5,15,51,12, .SerialNumber
    OKButton 64,13,41,14
End Dialog

Dim dialog1 As SerialNumDialog
Dialog dialog1

'Display the entered serial number
MsgBox dialog1.SerialNumber
```

See Also

[CancelButton](#)

[CheckBox](#)

[ComboBox](#)

[Dialog( )](#)

[Dialog](#)

[DropListBox](#)

[GroupBox](#)

[ListBox](#)

[OKButton](#)

[OptionButton](#)

[OptionGroup](#)

[Picture](#)

[PushButton](#)

[Text](#)

[Begin Dialog...End Dialog](#)

[PictureButton](#)

Time$ **Statement**

Description: This statement sets the system time using the 24-hour clock.

Syntax: Time$ = newTime

Parameter: newTime

String formatted as one of the following: HH, HH:MM, or HH:MM:SS.

Example: The following statements illustrate the use of Time$ to set the time.

```
Time$ = "16:05"      'Set the time to 16:05:00 or 4:05p.m.
Time$ = "8:5" 'Set the time to 8:05:00a.m.
```

See Also

Time$( )
Date$( )
Date$

Time$( )   Function

Description: This function returns the system time as a string using the format HH:MM:SS.

Syntax: Time$( )

Example: The following statement saves the current system time as the variable currentTime.

```
currentTime$ = Time$( )
```

See Also

[Time$](#)
[Date$( )](#)
[Date$](#)
[Now( )](#)

Timer( )   Function

Description: This function returns the number of seconds since midnight.
The value returned is a number of type long.

Syntax: Timer( )

Example: The following statement stores the seconds since midnight in a variable of type long.

```
secSinceMidnight& = Timer( )
```

See Also

Time$( )
[Now( )](Now( ))

TimeSerial( )   Function

Description: This function returns a double-precision number representing the specified time as a serial time with a date of zero (Dec. 30, 1899).

Syntax: TimeSerial(hour, minute, second)

Parameters: hour

A numeric expression indicating the hour with a number from 1 to 24.

minute

A numeric expression indicating the minute with a number from 1 to 60.

second

A numeric expression indicating the second with a number from 1 to 60.

Example: The following example obtains the serial time for 3:30 p.m.

```
serialDT# = TimeSerial(15,30,0)
```

See Also

[DateValue( )](#)
[TimeValue( )](#)
[DateSerial( )](#)

TimeValue( )   Function

Description: This function returns a double-precision number that is the serial representation of the specified time or specified date and time.

Syntax: TimeValue(timeStr)

Parameter: timeStr

A string expression containing the time or date and time. The order of the date items depends on the settings contained in the [intl] section of the WIN.INI file. Check the International dialog box from the Control Panel to review the settings. Valid time separators are the colon (:) and period (.). If a particular date or time item is missing, the missing items are set to zero. For example, the string "10 pm" would be interpreted as "22:00:00".

Example: The following example obtains the serial time for 3:30 p.m.

```
serialDT# = TimeValue("3:30 PM")
```

See Also

[DateValue( )](#)
[TimeSerial( )](#)
[DateSerial( )](#)

Trim$( )   **Function**

Description:	This function returns the specified string with any leading and/or trailing spaces removed.
Syntax:	Trim$(exprS)
Parameter:	exprS
	A string expression.
Example:	The following example demonstrates the use of Trim$( ).

```
aString$ = "  3 leading and 3 trailing spaces  "
'Now remove the leading and trailing spaces
aString = Trim$(aString)
'aString should now be equal to the string
'"3 leading and 3 trailing spaces"
```

See Also

[LTrim$( )](#)
[RTrim$( )](#)

TRUE   **Constant**

Description: This numeric constant can be used in logical expressions. It can be assigned to variables of type integer or long so that the variables can be used in logical expressions. Its value is -1.

Syntax: TRUE

Example: The following example returns the value TRUE if a specified integer is even. Otherwise, the function returns the value FALSE.

```
Function Even(n As Integer)

    If (n MOD 2) = 0 Then
            Even = TRUE
    Else
            Even = FALSE
    End If
End Function
```

See Also

[False](#)
[Boolean (data type)](#)

Type

Statement

Description: The Type statement creates a structure definition that can then be used with the Dim statement to declare variables of that type. The *username* field specifies the name of the structure that is used later with the Dim statement.

Syntax: Type *username*
    variable As *type*
    variable As *type*
    variable As *type*
    :
End Type

Comments: Within a structure definition appear field descriptions in the format:

    variable As *type*

Where *variable* is the name of a field of the structure, and *type* is the data type for that variable. Any fundamental data type or previously declared user-defined data type can be used within the structure definition (structures within structures are allowed). Only fixed arrays can appear within structure definitions.

The Type statement can only appear outside of subroutine and function declarations.

When declaring strings within fixed-size types, it is useful to declare the strings as fixed-length. Fixed-length strings are stored within the structure itself rather than in the string space. For example, the following structure will always require 62 bytes of storage:

```
Type Person
   FirstName As String * 20
   LastName As String * 40
   Age As Integer
End Type
```

Note: Fixed-length strings within structures are size-adjusted upward to an even byte boundary. Thus, a fixed-length string of length 5 will occupy 6 bytes of storage within the structure.

Example: This example displays the use of the Type statement to create a structure representing the parts of a circle and assign values to them.

```
Type Circ
    Msg As String
    Rad As Integer
    Dia As Integer
    Are As Double
    Cir As Double
End Type
```

```
Sub Main()
    Dim Circle As Circ
    Circle.Rad = 5
    Circle.Dia = Circle.Rad * 2
    Circle.Are = Circle.Rad ^ 2 * Pi
    Circle.Cir = Circle.Dia * Pi
    Circle.Msg = "The area of the circle is: " + Str$(Circle.Are)
    MsgBox Circle.Msg
End Sub
```

See Also

[Dim](#)
[Public](#)
[Private](#)

UBound( )   Function

Description: This function returns an integer indicating the upper bound for subscripts in the specified dimension of the specified array.

Syntax: UBound(arrayName [,dimension])

Parameters: arrayName

The name of an array.

dimension

A numeric expression indicating the dimension of an array. The default is 1 for the first dimension.

Example: The following example finds the upper bound for subscripts in the first dimension of a two-dimensional array.

```
Dim Array1(0 To 3, 0 To 2) As Integer
'Determine the upper bound
highest_subscript = UBound(Array1)
```

See Also

LBound( )
ArrayDims( )

UCase$( ) **Function**

Description: This function converts a string to uppercase.

Syntax: UCase$(exprS)

Parameter: exprS

A string expression.

Example: The following example results in the string "THIS IS ONLY A TEST" being assigned to the variable newString.

```
newString$ = UCase$("This is Only a Test!")
```

See Also

[LCase$( )](#)

Unlock   Statement

Description:	Unlocks a section of the specified file, allowing other processes access to that section of the file.
Syntax:	Lock [#] *filenumber* [,{*record*	[*start*] To *end*}]
Comments:	The Unlock statement requires the following parameters:

Parameter	Description
filenumber	Integer used by Symantec Basic to refer to the open file-the number passed to the Open statement.
record	Long specifying which record to unlock.
start	Long specifying the first record within a range to be unlocked.
end	Long specifying the last record within a range to be unlocked.

For sequential files, the *record*, *start*, and *end* parameters are ignored: the entire file is unlocked.

The section of the file is specified using one of the following:

Syntax	Description
No record specification	Unlock the entire file.
record	Unlock the specified record number (for Random files) or byte (for Binary files).
to *end*	Unlock from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
start to *end*	Unlock the specified range of records (for Random files) or bytes (for Binary files).

The unlock range must be the same as that used by the Lock statement.

Example: This example creates Test2.Dat and fills it with ten string variable records. These are displayed in a dialog box. The file is then reopened for read/write, and each record is locked, modified, rewritten, and unlocked. The new records are then displayed in a dialog box.

```
Const crlf$ = Chr$(13) + Chr$(10)

Sub Main()
    A$ = "This is record number: "
```

```
       B$ = "0"
       Rec$ = ""
       Msg$ = ""
       Open "Test2.Dat" for random-access write shared as #1
       For x% = 1 To 10
          Rec = A + Str$(x)
          Lock #1,x
          Put #1,,Rec
          Unlock #1,x
          Msg = Msg + Rec + crlf
       Next x
       Close
       MsgBox "The records are: " + crlf + Msg
       Msg = ""
       Open "Test2.Dat" for random-access read write shared as #1
       For x = 1 to 10
          Rec = Mid$(Rec,1,23) + Str$(11-x)
          Lock #1,x                          'Lock it for our use.
          Put #1,x,Rec                     'Nobody's changed it.
          UnLock #1,x
          Msg = Msg + Rec + crlf
       Next x
       MsgBox "The records are: " + crlf + Msg
       Close
End Sub
```

See Also

[Lock](#)
[Open](#)

Val( ) **Function**

Description:	This function converts the specified string expression to a number of type double and returns that double-precision number. It returns 0 if the string does not contain a number.
Syntax:	Val(exprS)
Parameter:	exprS

A string representation of a number. It can contain any of the following:

- Leading minus sign (for non hex or octal numbers only)

- Hexadecimal number in the format: &Hhex_digits

- Octal number in the format: &Ooctal_digits

- Floating-point number, which can contain a decimal point and optional exponent.

- Spaces, tabs, and linefeeds, because they are ignored by the function Val.

Example: The following three lines convert hexadecimal, octal, and decimal string representations of the number 16 into their numeric equivalents.

```
hexConv% = Val("&H10")      'hexConv equals 16
octConv% = Val("&O20")      'octConv equals 16
decConv% = Val("16")'decConv equals 16
```

See Also

[CDbl( )](#)
[Str$( )](#)

Variant   **Data type**

Syntax:	Variant
Description:	A data type used to declare variables that can hold one of many different types of data.
Comments:	During a variant's existence, the type of data contained within it can change. Variants can contain any of the following types of data:

Type of Data	BasicScript Data Types
Numeric	Integer, Long, Single, Double, Boolean, Date, Currency.
Logical	Boolean.
Dates and times	Date.
String	String.
Object	Object.
No valid data	A variant with no valid data is considered Null.
Uninitialized	An uninitialized variant is considered Empty.

There is no type-declaration character for variants.

The number of significant digits representable by a variant depends on the type of data contained within the variant.

Variant is the default data type for BasicScript. If a variable is not explicitly declared with Dim, Public, or Private, and there is no type-declaration character (i.e., #, @, !, %, or &), then the variable is assumed to be Variant.

Determining the Subtype of a Variant

The following functions are used to query the type of data contained within a variant:

Function	Description
VarType	Returns a number representing the type of data contained within the variant.
IsNumeric	Returns True if a variant contains numeric data. The following are considered numeric:

Integer, Long, Single, Double, Date, Boolean, Currency

If a variant contains a string, this function returns True if the string can be converted to a number.

If a variant contains an Object whose default property is numeric, then IsNumeric returns True.

IsObject	Returns True if a variant contains an object.
IsNull	Returns True if a variant contains no valid data.
IsEmpty	Returns True if a variant is uninitialized.
IsDate	Returns True if a variant contains a date. If the variant contains a string, then this function returns True if the string can be converted to a date. If the variant contains an Object, then this function returns True if the default property of that object can be converted to a date.

Assigning to Variants

Before a Variant has been assigned a value, it is considered empty. Thus, immediately after declaration, the VarType function will return ebEmpty. An uninitialized variant is 0 when used in numeric expressions and is a zero-length string when used within string expressions.

A Variant is Empty only after declaration and before assigning it a value. The only way for a Variant to become Empty after having received a value is for that variant to be assigned to another Variant containing Empty, for it to be assigned explicitly to the constant Empty, or for it to be erased using the Erase statement.

When a variant is assigned a value, it is also assigned that value's type. Thus, in all subsequent operations involving that variant, the variant will behave like the type of data it contains.

Operations on Variants

Normally, a Variant behaves just like the data it contains. One exception to this rule is that, in arithmetic operations, variants are automatically promoted when an overflow occurs. Consider the following statements:

```
Dim a As Integer,b As Integer,c As Integer
Dim x As Variant,y As Variant,z As Variant

a% = 32767
b% = 1
c% = a% + b%                    'This will overflow.

x = 32767
y = 1
z = x + y                       'z becomes a Long because of Integer
overflow.
```

In the above example, the addition involving Integer variables overflows because the result (32768) overflows the legal range for integers. With Variant variables, on the other hand, the addition operator recognizes the overflow and automatically promotes the result to a Long.

Adding Variants

The + operator is defined as performing two functions: when passed strings, it concatenates them; when passed numbers, it adds the numbers.

With variants, the rules are complicated because the types of the variants are not known until

execution time. If you use +, you may unintentionally perform the wrong operation.

It is recommended that you use the & operator if you intend to concatenate two String variants. This guarantees that string concatenation will be performed and not addition.

Variants That Contain No Data

A Variant can be set to a special value indicating that it contains no valid data by assigning the Variant to Null:

```
Dim a As Variant
a = Null
```

The only way that a Variant becomes Null is if you assign it as shown above.

The Null value can be useful for catching errors since its value propagates through an expression.

Variant Storage

Variants require 16 bytes of storage internally:

A 2-byte type

A 2-byte extended type for data objects

4 bytes of padding for alignment

An 8-byte value

Unlike other data types, writing variants to Binary or Random files does not write 16 bytes. With variants, a 2-byte type is written, followed by the data (2 bytes for Integer and so on).

Disadvantages of Variants

The following list describes some disadvantages of variants:

Using variants is slower than using the other fundamental data types (i.e., Integer, Long, Single, Double, Date, Object, String, Currency, and Boolean). Each operation involving a Variant requires examination of the variant's type.

Variants require more storage than other data types (16 bytes as opposed to 8 bytes for a Double, 2 bytes for an Integer, and so on).

Unpredictable behavior. You may write code to expect an Integer variant. At runtime, the variant may be automatically promoted to a Long variant, causing your code to break.

Passing Nonvariant Data to Routines Taking Variants

Passing nonvariant data to a routine that is declared to receive a variant by reference prevents that variant from changing type within that routine. For example:

```
Sub Foo(v As Variant)
   v = 50                 'OK.
   v = "Hello, world."    'Get a type-mismatch error here!
End Sub

Sub Main()
   Dim i As Integer
   Foo I                  'Pass an integer by reference.
End Sub
```

In the above example, since an Integer is passed by reference (meaning that the caller can change the original value of the Integer), the caller must ensure that no attempt is made to change the variant's type.

Passing Variants to Routines Taking Nonvariants

Variant variables cannot be passed to routines that accept nonvariant data by reference, as demonstrated in the following example:

```
Sub Foo(i As Integer)
End Sub

Sub Main()
   Dim a As Variant
```

```
        Foo a                    'Compiler gives type-mismatch error here.
    End Sub
```

Platform(s):   All.

See Also

[Currency (data type)](#)
[Date (data type)](#)
[Double (data type)](#)
[Integer (data type)](#)
[Long (data type)](#)
[Single (data type)](#)
[String (data type)](#)
[Boolean (data type)](#)
[Deftype](#)
[CVar (function)](#)
[Empty (constant)](#)
[Null (constant)](#)
[VarType (function)](#)

VarType   Function

Syntax: VarType(variable)

Description: Returns an Integer representing the type of data in variable.

Comments: The *variable* parameter is the name of any Variant.

The following table shows the different values that can be returned by VarType:

Value	Constant	Data Type
0	ebEmpty	Uninitialized
1	ebNull	No valid data
2	ebInteger	Integer
3	ebLong	Long
4	ebSingle	Single
5	ebDouble	Double
6	ebCurrency	Currency
7	ebDate	Date
8	ebString	String
9	ebObject	Object (OLE automation object)
10	ebError	User-defined error
11	ebBoolean	Boolean
12	ebVariant	Variant (not returned by this function)
13	ebDataObject	Non-OLE automation object

Comments: When passed an cbject, the VarType function returns the type of the default property of that object. If the object has no default property, then either ebObject or ebDataObject is returned, depending on the type of *variable*.

Example:

```
Sub Main()
    Dim v As Variant
    v = 5&                                  'Set v to a Long.
```

```
        If VarType(v) = ebInteger Then
            Msgbox "v is an Integer."
        ElseIf VarType(v) = ebLong Then
            Msgbox "v is a Long."
        End If
End Sub
```

Platform(s):   All.

See Also

[Empty (constant)](#)
[Null (constant)](#)
[Variant (data type)](#)

Weekday( )   Function

Description: This function returns an integer indicating the day of the week for the specified serial date. The value returned ranges from 1 to 7, where 1 is Sunday.

Syntax: Weekday(serialDate)

Parameters: serialDate

A double-precision expression containing a serial date.

Example: After calling the Now( ) function, you can extract the current day of the week from the date and time.

```
'Get the current date and time
serialDT# = Now( )

'Now extract the value
theWeekday% = Weekday(serialDT)
```

See Also

[Day( )](#)
[Minute( )](#)
[Second( )](#)
[Month( )](#)
[Year( )](#)
[Hour( )](#)
[DatePart](#)

While...Wend   Construct

Description:	This construct repeats a statement or group of statements while a logical expression is TRUE.
Syntax:	While logicalExpr [statements] Wend
	logicalExpr An expression containing relational and/or logical operators.
Statement:	s A series of executable statements.
Example:	The following example calculates the factorial of a positive integer. The While...Wend loop terminates when the counter is greater than FactNum. The value of Counter changes during every iteration of the loop.

```
Sub FactCal
'Loop counter.
Dim Counter As Integer
'Stores the result of factorial.
Dim Factorial As Integer
'Number for calculation.
Dim FactNum As Integer

'FactNum is input by user.

Factorial    = 1
Counter      = 1

'Calculate factorial. When Counter is greater
'than FactNum, the While loop terminates.
While Counter <= FactNum
    Factorial = Factorial * Counter
    Counter = Counter + 1
Wend
MsgBox "The factorial is: " + Str$(Factorial) + "."
End Sub
```

See Also

[Do...Loop](#)
[For...Next](#)

Width#　　　　　　　　　　**Statement**

Description: Specifies the line width for sequential files opened in either Output or Append mode.

Syntax: Width# *filenumber,newwidth*

Comments: The Width# statement requires the following parameters:

Parameter	Description
filenumber	Integer used by Symantec Basic to refer to the open file-the number passed to the Open statement.
newwidth	Integer between 0 to 255 inclusive specifying the new width. If *newwidth* is 0, then no maximum line length is used.

When a file is initially opened, there is no limit to line length. This command forces all subsequent output to the specified file to use the specified value as the maximum line length.

The Width statement affects output in the following manner: if the column position is greater than 1 and the length of the text to be written to the file causes the column position to exceed the current line width, then the data is written on the next line.

The Width statement also affects output of the Print command when used with the Tab and Spc functions.

Example: This statement sets the maximum line width for file number 1 to 80 columns.

```
Sub Main()
    Width #1,80
End Sub
```

See Also

Print
Print#
[Tab](Tab)
[Spc](Spc)

Word$( )   Function

Description: This function returns a string containing all the words in the specified text starting with the word specified by the first parameter and ending with the word specified by the last parameter. It returns the empty string if the first parameter specifies a number greater than the number of words in the text. It returns all the rest of the words if the last parameter is greater than the number of words in the text.

Syntax: Word$(text, first[, last])

Parameters: text

A string expression containing words delimited with spaces, tabs, or carriage returns/linefeeds.

first

An integer representing the first word to be read.
The first word in the text is number 1.

last

An integer representing the last word to be read.
The default is the value of first, so at least one word is read.

Example: In the following example, the string variable whoIsIt is parsed for a first, middle, and last name. If the string has more than two words, the first word is the first name, the second word is the middle name, and the third word is the last name. If the string has two words, the first word is the first name and the second word is the last name. If neither of the previous two conditions holds, the first word is the first name.

```
Dim first$, middle$, last$, whoIsIt$

whoIsIt = "Joe Roe Doe"

'Check If first, middle, last name are all present
If WordCount(whoIsIt) > 2 Then
    first = Word$(whoIsIt, 1)
    middle = Word$(whoIsIt, 2)
    last = Word$(whoIsIt, 3)

'Check for presence of only first and last name
ElseIf WordCount(whoIsIt) > 1 Then
    first = Word$(whoIsIt, 1)
    middle = ""
    last = Word$(whoIsIt, 2)

'Assume first name only
Else
```

```
        first = Word$(whoIsIt, 1)
        middle = ""
        last = ""
    End If
```

See Also

[Item$( )](#)
[ItemCount( )](#)
[Line$( )](#)
[LineCount( )](#)
[WordCount( )](#)

WordCount( )　　　　　Function

Description: This function returns an integer indicating the number of words in the specified text.

Syntax: WordCount(text)

Parameter: text

A string expression containing words delimited with spaces, tabs, or carriage returns/linefeeds.

Example: In the following example, the string variable whoIsIt is parsed for a first, middle, and last name. If the string has more than two words, the first word is the first name, the second word is the middle name, and the third word is the last name. If the string has two words, the first word is the first name and the second word is the last name. If neither of the previous two conditions holds, the first word is the first name.

```
Dim first$, middle$, last$, whoIsIt$

whoIsIt = "Joe Roe Doe"

'Check If first, middle, last name are all present
If WordCount(whoIsIt) > 2 Then
    first = Word$(whoIsIt, 1)
    middle = Word$(whoIsIt, 2)
    last = Word$(whoIsIt, 3)

'Check for presence of only first and last name
ElseIf WordCount(whoIsIt) > 1 Then
    first = Word$(whoIsIt, 1)
    middle = ""
    last = Word$(whoIsIt, 2)

'Assume first name only
Else
    first = Word$(whoIsIt, 1)
    middle = ""
    last = ""
End If
```

See Also

[Item$( )](#)
[ItemCount( )](#)
[Line$( )](#)
[LineCount( )](#)
[Word$( )](#)

Write # Statement

Description: This statement writes a list of expressions to the specified file.
Numbers written to the file have no leading or trailing spaces added. Strings are enclosed in quotation marks. A comma is written to the file after each expression except the last expression. A carriage return is written after the last expression.

Syntax: Write [#]fileNum [, expr]…

Parameters: fileNum

The integer that is assigned to a file when it is opened with the Open statement. This is the number that Symantec Basic uses instead of a filename to refer to the file.

expr

A string or numeric expression.

Examples: The following example writes the first four positive numbers along with their squares to the open file. The items on each line are separated by commas and the numbers have neither leading nor trailing spaces.

```
Open "testfile" For Output As #1
For i = 1 To 4
    'Each line has the number and its square
    Write #1, i, i * i
Next i
    The output resulting from the above statements appears in the file as
follows:
1,1
2,4
3,9
4,16
    The next example writes the strings "asdf" and "qwer" to two
consecutive lines.
Open "testfile" For Output As #1
Write #1, "asdf"
Write #1, "qwer"
```

See Also

[Open](#)
[Put](#)
[Print#](#)

WriteINI   **Statement**

Description:	This statement can add, modify, or delete entries in an .INI file; add or delete a section; or create a new file. Whenever the specified file, section, or entry does not exist, it is created.
Syntax:	WriteINI section, entryName, value[, filename]
Parameters:	section

A string expression containing the name of the section. Do not use the square brackets that appear in the .INI file around the section name.

entryName

A string expression containing the name of the entry. If you specify an empty string, the entire section is deleted. Do not use the = operator.

value

The new value to write to the .INI file for the specified entry. If an empty string is used, the entry is deleted from the file.

filename

A string expression containing the complete or relative pathname for the .INI file to edit.   If no path is specified, the Windows directory is searched. The default is the WIN.INI file.

Example: The following example makes CLOCK.EXE the value of the load= entry in the [windows] section of the WIN.INI file.

```
WriteINI "windows", "load","clock.exe"
```

See Also

[ReadINI$( )](#)
[ReadINISection](#)

XOR   **Logical Operator**

Description: Exclusive OR is a logical operator and usually joins two logical or relational expressions into another logical expression. The result is TRUE if one and only one of the relational or logical expressions is TRUE. Otherwise it is FALSE.

If the expressions are numeric, the result is a bitwise XOR of the two numbers. If either of the expressions is a floating-point number,
the two expressions are converted to longs before the bitwise XOR.

Syntax: expr1 XOR expr2

expr1, expr2

Numeric, relational, or logical expressions.

Example: In the following example, the XOR operator is used to test that one and only one condition holds.

```
/* Give free admission
    to anyone named Hercules who is 2 years
    or older and to anyone not named Hercules
    who is less than 2 years old */
If personName = "Hercules" XOR age < 2 Then
    freeAdmission = TRUE
End If
```

See Also

[Or](#)
[Eqv (operator)](#)
[Imp](#)
[AND](#)

Year( ) **Function**

Description: This function returns an integer indicating the year of the date encoded in the specified serial date. The value returned ranges from 100 to 9999.

Syntax: Year(serial)

Parameter: serial

A double-precision expression containing a serial date.

Example: After calling the Now( ) function, you can extract the current year from the date and time.

```
'Get zthe current date and time
serialDT# = Now( )

'Now extract the value
theYear% = Year(serialDT)
```

See Also

[Day( )](#)
[Minute( )](#)
[Second( )](#)
[Month( )](#)
[Hour( )](#)
[Weekday( )](#)
[DatePart](#)

Commands and Queries Available to Use with Symantec BASIC

You can use global commands for scripting while working in any part of Visual Café. If you intend to use menu selections instead of command codes, however, you must be in the appropriate window to make the menu selections.

Conventions

In this guide the parameter names may have characters in the end of the parameter name indicating the parameter type:

? (Question Mark) indicates a Boolean parameter

% (Percent Sign) indicates an integer value

$ (Dollar Sign) indicates a string parameter

BufferCloseAllFiles

Note on Commands for Brief(tm) Compatibility

1. **You may choose the key bindings for Brief compatibility as follows:** From the main menu choose **File** then **Environment Options...**. Press the **Keyboard** tab. Choose **brief** from the **File:** drop-down list.

2. **When you select the Brief key bindings, the menu accelerators are disabled.** Changing this setting is possible but is not advised as it will affect the permanent key map for Brief.

3. **Also in Brief, the Typing Replaces Selection option is disabled.**

4. **Virtual cursoring is activated.**

5. **Cursors are used to select text without using the Shift key** To cancel a selection, press ESC, or toggle out using the appropriate Brief command, or click the text.

6. **Macros written in Brief mode will not run as Brief macros when Brief mode is turned off.** For example, in Brief mode, cursor_right is automatically converted to select_character_right during marking. When a macro is run in non-Brief mode, cursor_right is not converted.

Command Reference Alphabetic Index

TextBackspace

TextBacktab

TextBeginningOfBuffer

TextBeginningOfLine

TextBottomOfWindow

TextBufferOptions

TextChangeCase

TextClearAllBreakpoints

TextClearSelect

TextCopy

TextCopyBlock

TextCopyLine

TextCursorDown

TextCursorLeft

TextCursorRight

TextCursorUp

TextCut

TextCutBlock

TextCutLine

TextDebugContinueToCursor

TextDebugJumpToCursor

TextDelete

TextDeleteBlock

TextDeleteLine

TextDeleteToEol

TextDeleteWordLeft

TextDeleteWordRight

TextDisableBreakpoint

TextDropBookmark(1-10)

TextDupLine

TextEndOfBuffer

TextEndOfLine

TextEnter

TextFind

TextFindAgain

TextFindNext

TextFindPrev

TextFindSelection

TextGotoFunction

TextGotoLine

TextGotoVariables

TextIndentBlock

TextInsertFile

TextInsertTab

TextLoadFile

TextLowercase

TextMatchDelimiter

TextNextPane

TextPageDown

TextPageUp

TextPaste

TextPrevPane

TextPrint

TextQueryValue

TextQuickWatch

TextReplace

TextRevert

TextSave

TextSaveAs

TextSelectAll

TextSelectCharLeft

TextSelectCharRight

TextSelectLine

TextSelectLineDown

TextSelectLineUp

TextSelectPageDown

TextSelectPageUp

TextSelectToBol

TextSelectToEnd

TextSelectToEol

TextSelectToTop

TextSelectWord

TextSelectWordLeft

TextSelectWordRight

TextSetConditionalBreakpoint

TextShowTabs

TextSpacesToTabs

TextSplitLine

TextStamp

TextSwapMark

TextTab

TextTabLeft

TextTabsToSpaces

TextToBottom

TextToCenter

TextToggleBreakpoint

TextToggleColumnSelect

TextToggleExclusiveSelect

TextToggleInsert

TextToggleLineSelect

TextToggleMarkSelect

TextToggleWordwrap

TextTopOfWindow

TextToTop

TextUndo

TextUnindentBlock

TextUnmarkBlock

TextUppercase

Command Reference Categories

Each of the categories listed below contains a set of ScriptMaker commands that are related by task. Click a category to see the list of related commands.

Buffer Commands (Global)	**Class Commands (Local)**
Debugger Commands	**General Editing Commands**
Hierarchy Commands (Local)	**Macro Commands (Global)**
Member Commands (Local)	**Miscellaneous Commands (Global)**
Object Commands	**Output Commands**
Search Commands (Local)	**Settings Commands (Global)**
Text Editor Commands (Local)	**Text Editor Movement Commands (Local)**
Text Editor Selection Commands (Local)	**Window Commands (Global)**

Buffer Commands (Global)

BufferCloseAllFiles
BufferGotoPane(1-9)
BufferIndexClose
BufferIndexSave
BufferSaveAllFiles
BufferSwitchToIndex

Debugger Commands

DebugPause
DebugRestart
DebugStepInto
DebugStepOut
DebugStepOver
ProjectDebug
TextDebugContinueToCursor
TextDebugJumpToCursor

General Editing Commands

EditCopy
EditCut
EditDelete
EditPaste
EditSelectAll
EditUndo

Output Commands (Global)

MessagesClearAll
MessagesCopyAll
MessagesCurrentError
MessagesFirstError
MessagesNextError
MessagesPreviousError

Macro Commands (Global)

MacroPlay
MacroRecordToggle
MacroScriptMaker

Settings Commands (Global)

ClassesOptions
ClassOptions
FileEnvironmentOptions
FilePageSetup
FilePrintSetup
MemberOptions
ProjectOptions
ToggleBackup

Window Commands (Global)

BufferGotoPane(1-9)
FileClose
FileOpenRecent(1-4)
FileSave
FileSaveAs
ProjectMinimizeAll
WindowBreakpoints
WindowCallStack
WindowClassBrowser
WindowComponentLibrary
WindowHierarchyEditor
WindowMessages
WindowNew
WindowNext
WindowPropertyList
WindowThreads
WindowVariables
WindowWatch

Miscellaneous Commands (Global)

dbAddTableWizard
dbNavigator
FileExit
HelpAbout
LiveUpdate
ProjectAddItem
ProjectCompile
SearchBookmarks
SearchCompareFiles
SearchFindInFiles
SearchGotoDefinition
ProjectParseAll

Class Commands (Local)

ClassAttributes
ClassesClassAttributes
ClassesGotoSource
ClassGoto
ClassInsert
ClassRemoveInheritance
InsertClass

Object Commands

ObjectAddInteraction
ObjectAddToLibrary
ObjectEditSourceObjectEditObject
ObjectGotoDefinition

Member Commands (Local)

ClassesDeleteMember
ClassesMemberAttributes
InsertMember
MemberAttributes
MemberDelete
MemberGoto
MemberInsert

Hierarchy Commands (Local)

HierarchyPrint
HierarchyViewImports

Search Commands (Local)

FindInfFilesAddAllToProject
FindInFilesAddSelectedToProject
FindInFilesChangeFind
FindInFilesGotoSource
SearchClearBookmark(1-10)
SearchFind
SearchFindAgain
SearchGotoBuffer
TextFind
TextFindAgain
TextFindNext
TextFindPrev
TextFindSelection

Text Editor Commands (Local)

[TextAlignComment](#)

[TextBufferOptions](#)

[TextChangeCase](#)

[TextClearAllBreakpoints](#)

[TextCopy](#)

[TextCopyBlock](#)

[TextCopyLine](#)

[TextCut](#)

[TextCutBlock](#)

[TextCutLine](#)

[TextDisableBreakpoint](#)

[TextDebugContinueToCursor](#)

[TextDebugJumpToCursor](#)

[TextDelete](#)

[TextDeleteBlock](#)

[TextDeleteLine](#)

[TextDeleteToEol](#)

[TextDeleteWordLeft](#)

[TextDeleteWordRight](#)

[TextDropBookmark(1-10)](#)

[TextDupLine](#)

[TextEnter](#)

[TextFind](#)

[TextFindAgain](#)

[TextFindNext](#)

[TextFindPrev](#)

[TextFindSelection](#)

[TextGotoVariables](#)

[TextIndentBlock](#)

[TextInsertFile](#)

[TextInsertTab](#)

[TextLoadFile](#)

[TextLowercase](#)

[TextMatchDelimiter](#)

[TextPaste](#)

[TextPrint](#)

[TextQueryValue](#)

[TextQuickWatch](#)

[TextReplace](#)

[TextRevert](#)

[TextSave](#)

[TextSaveAs](#)

[TextSetConditionalBreakpoint](#)

[TextShowTabs](#)

[TextSpacesToTabs](#)

[TextSplitLine](#)

[TextStamp](#)

[TextSwapMark](#)

[TextTabsToSpaces](#)

[TextToggleBreakpoint](#)

[TextToggleColumnSelect](#)

[TextToggleExclusiveSelect](#)

[TextToggleInsert](#)

[TextToggleLineSelect](#)

[TextToggleMarkSelect](#)

[TextToggleWordwrap](#)

[TextUndo](#)

[TextUnindentBlock](#)

[TextUppercase](#)

[TextViewEvents](#)

[TextWrapPara](#)

[TextWriteBlock](#)

[TextZoomPane](#)

Text Editor Selection Commands

TextClearSelect	**TextFindSelection**
TextSelectAll	**TextSelectCharLeft**
TextSelectCharRight	**TextSelectLine**
TextSelectLineDown	**TextSelectLineUp**
TextSelectPageDown	**TextSelectPageUp**
TextSelectToBol	**TextSelectToEnd**
TextSelectToEol	**TextSelectToTop**
TextSelectWord	**TextSelectWordLeft**
TextSelectWordRight	**TextUnmarkBlock**

Text Editor Movement Commands (Local)

TextBackspace	**TextBacktab**
TextBeginningOfBuffer	**TextBeginningOfLine**
TextBottomOfWindow	**TextCursorDown**
TextCursorLeft	**TextCursorRight**
TextCursorUp	**TextEndOfBuffer**
TextEndOfLine	**TextGotoFunction**
TextGotoLine	**TextNextPane**
TextPageDown	**TextPageUp**
TextPrevPane	**TextTab**
TextTabLeft	**TextToBottom**
TextToCenter	**TextTopOfWindow**
TextToTop	**TextWindowDown**
TextWindowUp	**TextWordLeft**
TextWordRight	

BufferCloseAllFiles

Parameters: `BufferConstant%`
Constant that specifies a buffer type. The following are the acceptable values and their meanings:

  1=file buffers,

  2=untitled buffers,

  3=file or untitled buffers,

  4=member buffers.

`QueryForSave?`
Boolean. If true, before closing any unsaved files, Visual Café will prompt you to save them.

`ViewsOnly?`
Boolean. If true, only the window is closed; the buffer is left in memory (unsaved). The next time the file is opened, it is opened from memory.

Description: Use this command to close all open buffers.

BufferIndexClose

Parameters: `Index%`
A 0-based integer that specifies which buffer to close.

`QueryForSave?`
Boolean. If true, before closing any unsaved files, Visual Café will prompt you to save them.

`ViewsOnly?`
Boolean. If true, only the window is closed, but the buffer is left in memory (unsaved). Next time the file is opened it is opened from memory.

Menu selection: From the main menu select **Search** then **Goto Buffer...**. Choose a buffer then click the **Close** button.

Description: Use this command to close an open buffer.

BufferIndexSave

Parameters:	Index% A 0-based integer that specifies which buffer to save. SaveUntitled? Boolean. If true, Visual Café will prompt you to save untitled files, otherwise untitled files will be ignored.
Menu selection:	From the main menu select **Search** then **Goto Buffer...**. Choose a buffer then click the **Save** button.
Description:	Use this command to save the specified buffer to a file on disk.

SearchGotoBuffer

Menu selection: From the main menu select **Search** then **Go to Buffer...**.
Description: Use this command to bring up the Go to Buffer dialog.

BufferSaveAllFiles

Parameters: `BufferConstant%`
Constant that specifies a buffer type. The following are the acceptable values and their meanings:

> 1=file buffers,

> 2=untitled buffers,

> 3=file or untitled buffers.

`PromptIfUntitled?`
Boolean. If true, Visual Café prompts to name each untitled buffer before it is saved.

`QueryFirst?`
Boolean. If true, Visual Café prompts before saving each file.

Menu selection: From the main menu select **File** then **Save All**.

Description: Use this command to save all current buffers.

BufferSwitchToIndex

Parameters: `Index%`
A 0-based integer that specifies which buffer to switch to.

Menu selection: From the main menu select **Search** then **Goto Buffer...**. Then double-click the name of buffer you want to switch to, or click on the name then press the Go To button.

Description: Use this command to switch to a buffer specified by the `Index%` parameter.

MacroPlay

Menu selection: From the main menu select **File** then **Macro** then **Play**.
Description: Use this command to play the default macro.

MacroRecordToggle

Menu selection: From the main menu select **File** then **Macro** then **Record Macro** (or **Stop Recording)**.

Description: Use this command to toggle the recorder on and off. This toggle state is reflected in the associated menu selection(s), which alternate between " **Record Macro**" and "**Stop Recording**." When you are not recording, the menu item will read **Record Macro**. When you are recording the menu item will read **Stop Recording**. Use this command to record the default macro. The default macro is the macro that plays when you select **Macro**, then **Play**.

MacroScriptMaker

Menu selection: From the main menu select **File** then **Macro** then **ScriptMaker...**.

Description: Use this command to bring up the ScriptMaker dialog which lets you create and edit macros.

FilePageSetup

Description: Use this command to bring up the **Page Setup** dialog, which lets you define how your printed output will look.

FileOpenRecent(1-4)

Menu selection: From the main menu select **File** then choose a file name from the recent files section (not recent projects).

Description: Use these commands to re-open one of the four most recently opened files (not projects).

SearchBookmarks

Menu selection: From the main menu select **Search** then **Bookmarks...**.

Description: Use this command to bring up the Bookmarks dialog, which lets you view, set, and go to different bookmarks.

ProjectParseAll

Menu selection: From the main menu select **Project** then **Parse All**.
Description: Use this command to force the re-parsing of all files even if they're marked as up-to-date.

ToggleBackup

Menu selection: From the main menu select **File** then **Environment Options...**. Select the Backup page in the Environment Options dialog and toggle the Backup files on Save checkbox.

Description: Using this command to toggles on and off the automatic backup of opened files when saved.

ClassesClassAttributes (ClassAttributes)

Parameters: `OriginalClassName$`
String. Original name of the class.

`NewClassName$`
String. New name for the class.

`Reserved1$`
String. Reserved for future use.

`NewBaseClassName$`
String. The name of the base class of the class specified in ClassName.

Menu selection: With a class selected in the Class Browser, choose **Classes** from the main menu then select **Class Attributes...**.

Right-click on a class in Class Browser then select **Class Attributes...** from the pop-up menu.

With a class selected in the Hierarchy Editor, choose **Hierarchy** from the main menu then select **Class Attributes...**.

Right-click on a class in Hierarchy Editor then select **Class Attributes...** from the pop-up menu.

Description: Opens the Class Attributes dialog for the currently selected class.

ClassesDeleteMember (MemberDelete)

Menu selection: With a member selected in the Class Browser, choose **Classes** from the main menu then select **Delete Member**.

Right-click on a member in the Class Browser, then select **Delete Member** from the pop-up menu.

Description: Use this command to delete the current member from the current class. The user will be prompted to ensure the delete is OK.

ClassesMemberAttributes (MemberAttributes)

Parameters: `FileName$`
String. The name of the source file.

`AccessKeyword%`
Constant that specifies the access level keyword used. The following are the acceptable values and their meanings:

   1=public,

   2=protected,

   3=private,

   4=package (no access keyword).

`Reserved1%`
Integer. Reserved for future use. Set to 0.

`Reserved2%`
Integer. Reserved for future use. Set to 1.

Menu selection: With a member selected in the Class Browser, choose **Classes** from the main menu then select **Member Attributes...**.

Right-click on a member in the Class Browser, then select **Member Attributes...** from the pop-up menu.

Description: Use this command to modify the selected member's access level and storage class.

ClassRemoveInheritance

Menu selection: Right-click on a connection in the Hierarchy Editor window, then select **Remove Inheritance** from the resulting pop-up menu.

Description: Use this command to delete the selected connection between a derived class and its base class.

MessagesClearAll

Description: Use this command to clear the Messages window.

MessagesCopyAll

Menu selection: Right-click in the Messages window and choose **Select All** from the pop-up menu, then right-click
again and choose **Copy**.

Description: Use this command to copy all the error message text from the Messages window to the clipboard.

SearchCompareFiles

Menu selection: From the main menu select **Search** then **Compare Files...**.

Description: Use this command to bring up the Compare Files dialog, which lets you find differences between two files.

FileExit

Menu selection: From the main menu select **File** then **Exit**.
Description: Use this command to exit Visual Café.

FilePrintSetup

Menu selection: From the main menu select **File** and then **Print Setup...**.
Description: Use this command to open the Print Setup dialog box.

BufferGotoPane(1-9)

Menu selection: From the main menu select **Search** then **Goto Buffer...**. Then double-click the name of buffer you want to switch to, or click on the name then press the Go To button.

Description: Use this command to display the specified text pane. They are numbered in the order they were created.

HelpAbout

Menu selection: From the main menu select **Help** then **About Visual Cafe**.

Description: Use this command to display the About dialog, which shows the version number and other product information.

HierarchyPrint

Menu selection: With the Hierarchy Editor active, select **File** from the main menu then choose **Print**.

Description: This command opens the Print dialog, letting you print the current class hierarchy displayed in the Hierarchy Editor.

HierarchyViewImports

Menu selection: With the Hierarchy Editor active, select **Hierarchy** from the main menu then choose **View Imports**.

Right-click in the Hierarchy Editor window then choose **View Imports** from the pop-up menu.

Description: This command toggles the display of all imported classes in the Hierarchy Editor window.

MemberGoto

Menu selection: With a member selected in the Class Browser, choose **Classes** from the main menu then select **Go to Source**.

Right-click on a member in the Class Browser, then select **Go to Source** from the pop-up menu.

Description: Use this command to open a Source window showing the current member's implementation. It makes the first line of its implementation the current line.

FindInFilesAddAllToProject

Menu selection: Right-click in the Find In Files window then select **Add All to Project** from the pop-up menu.

Description: Use this command to add all the files in the Find In Files window to the current project.

FindInFilesAddSelectedToProject

Menu selection: Right-click in the Find In Files window then select **Add Item to Project** from the pop-up menu.
Description: Use this command to add all the selected files in the Find In Files window to the current project.

FindInFilesChangeFind

Menu selection: Right-click in the Find In Files window then select **Change Find...** from the pop-up menu.

Description: Use this command to search only those files listed in the Find In Files window.

FindInFilesGotoSource

Menu selection: Right-click in the Find In Files window then select **Go to Source** from the pop-up menu.

Description: Open a Source window displaying the file selected in the Find In Files window.

TextAlignComment

Description: Use this command to align the selected comment(s) with the column specified in the Environment Options dialog Format page. A comment can be selected by highlighting or by placing the insertion point in or before the comment text. Note that this is a user-configurable feature; by default, it has no key binding.

TextBackspace

Description: This command is equivalent to pressing the Backspace key.

       If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextBeginningOfBuffer

Description: Use this command to move the cursor to the beginning of the file (buffer).

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextBeginningOfLine

Description: Use this command to move the cursor to the beginning of the current line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextBottomOfWindow

Description:	Use this command to move the cursor to the bottom line of the window. If possible, the cursor column remains the same.
	If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextBufferOptions

Parameters: TabSize%
Integer. The new tab size.

RightMargin%
Integer. The new right margin

WordWrap?
Boolean. If true, words will automatically wrap onto the next line as you type them.

AutoIndent?
Boolean. If true, the text editor will auto-indent new lines.

ExpandTabsWithSpaces?
Boolean. If true, each tab will be replaced with the corresponding number of spaces.

ReadOnly?
Boolean. If true, the file becomes read-only.

Persistent?
Boolean. If true, the file becomes persistent.

UseAsDefault?
Boolean. If true, these settings will become default for all text buffers.

LanguageKeywords%
Integer.

CheckDelimiters?
Boolean. If true, will check for matching delimiters as text is entered.

IndentAfterBrace?
Boolean. If true, will auto-indent after all braces that are entered.

IndentComments?
Boolean. If true, will indent all single line comments at a specific column as they are entered.

CommentCol%
Integer. The column to start single-line comments on.

EnableCustomKeywords?
Boolean. If true, custom keywords in the text will be highlighted.

Menu selection: To get to the text format page either select **File** from the main menu, choose **Environment Options...**, then select the Format page, or right-click in a source window or pane and select **Format Options** from the pop-up menu.

Description: Use this command to customize settings for the file in the active window.

TextChangeCase

Description: Use this command to change all lower-case characters in a selected block of text to upper case, and all upper-case characters to lower case. If no text is selected, changes the case of the character at the current cursor position.

TextClearSelect

Description: Unselects the current text selection. Ends Brief Selection Mode.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

EditCopy (TextCopy)

Menu selection: From the main menu select **Edit** then **Copy**.

Right-click in the Source window and select **Copy** from the pop-up menu.

Right-click in the Class Browser source pane and select **Copy** from the pop-up menu.

Right-click on an object in the Form Designer and select **Copy** from the pop-up menu.

Description: Use this command to copy the selected item or text from the active window to the clipboard.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCopyBlock

Description: Use this command to copy selected text from the active window to the clipboard.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCopyLine

Description: Use this command to copy the line containing the cursor from the active window to the clipboard.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCursorDown

Description: Use this command to move the cursor down one line in the active source window.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCursorLeft

Description: Use this command to move the cursor one character to the left in the active source window.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCursorRight

Description: Use this command to move the cursor one character to the right in the active source window.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCursorUp

Description: Use this command to move the cursor up one line in the active source window.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

EditCut (TextCut)

Menu selection: From the main menu select **Edit** then **Cut**.

Right-click in the Source window and select **Cut** from the pop-up menu.

Right-click in the Class Browser source pane and select **Cut** from the pop-up menu.

Description: Use this command to delete the selected text from the active window and move it to the clipboard.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCutBlock

Description: Use this command to delete the selected text from the active window and move it to the clipboard.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextCutLine

Description: Use this command to delete the line containing the cursor from the active source window and move it to the clipboard. The cursor is placed at column 1 of the next line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

EditDelete (TextDelete)

Menu selection: From the main menu select **Edit** then **Delete**.

Description: Use this command to delete the selected item or text. If no text is selected, deletes the character to the right of the cursor.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDisableBreakpoint

Menu selection: With a source window active select **Source** from the main menu then choose **Clear Breakpoint**.

Right-click in a Source window and select **Clear Breakpoint** from the pop-up menu.

Right-click in the source pane of the Class Browser and select **Clear Breakpoint** from the pop-up menu.

Description: Use this command to disable the breakpoint at the current line. Available only while in debugging mode.

TextDebugContinueToCursor

Menu selection: While debugging with a Source window active, select **Debug** from the main menu then choose **Continue to Cursor**.

Right-click in the Source window and select **Continue to Cursor** from the pop-up menu.

Right-click in the source pane of the Class Browser window and select **Continue to Cursor** from the pop-up menu.

Description: Use this command to execute the program until it reaches the current line. Available only while in debugging mode.

TextDebugJumpToCursor

Description: Use this command to skip all the instructions until the current line. This command is similar to performing a jump to the address of the current line. Available only while in debugging mode.

TextToggleBreakpoint

Menu selection: With a source window active select **Source** from the main menu then choose **Set** or **Clear Breakpoint**.

Right-click in a Source window and select **Set** or **Clear Breakpoint** from the pop-up menu.

Right-click in the source pane of the Class Browser and select **Set** or **Clear Breakpoint** from the pop-up menu.

Description: Use this command to toggle a breakpoint at a current line.

TextDeleteBlock

Menu selection: From the main menu select **Edit** then **Delete**.

Description: Use this command to delete the selected text without copying it into the clipboard. If no text is selected no action is performed.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDeleteLine

Description: Use this command to delete the line containing the cursor. The cursor moves to the first column of the following line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDeleteToEol

Description: Use this command to delete the text from the current cursor position to the end of the current line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDeleteWordLeft

Description: Use this command to delete the text from the current cursor position to the start of the word, if the cursor is within a word. If the cursor is not within a word, it deletes text until the next word to the left is encountered.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDeleteWordRight

Description: Use this command to delete the text from the current cursor position to the end of the word, if the cursor is within a word. If the cursor is not within a word, it deletes text until the next word to the right is encountered.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextDropBookmark(1-10)

Menu selection: With Source window active select **Search** from the main menu then choose **Bookmarks....** In the Bookmarks dialog select the desired bookmark and press the Drop button.

Description: This command sets the specified bookmark to the current cursor position.

TextDupLine

Description: Use this command to create a copy of the current line.

This command is not useful when using key bindings for Brief compatibility. If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextEndOfBuffer

Description: Use this command to move the cursor to the end of the current file.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextEndOfLine

Description: Use this command to move the cursor to the end of the current line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextEnter

Description: This command will insert a carriage return before the selection in insert mode, and move the cursor to the beginning of the line following the start of the selection in the overwrite mode, unless the Typing Replaces Selection option is currently disabled, as it is in the key file for Brief compatibility.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextFind

Parameters:	`Pattern$`
	String. The pattern to search for.
	`WholeWords?`
	Boolean. If true, only whole words will be matched to the pattern.
	`RegularExpression?`
	Boolean. If true, the command will evaluate regular expressions in the pattern.
	`IgnoreCase?`
	Boolean. If true, the search will be case-insensitive.
	`Forward?`
	Boolean. If true, the search will be performed scanning forward, otherwise it will scan backwards.
Menu selection:	With a Source window or a Class Browser source pane active, select **Search** from the main menu then choose **Find...**.
Description:	Use this command to search the active file for specified text or a regular expression.
	If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextFindSelection

Description: Use this command to find the next occurrence of the selected text, scanning in the forward direction.

TextGotoFunction

Parameters:	`FunctionName$`
	String. The function name to go to.
Menu selection:	With a Source window or a Class Browser source pane active, select **Search** from the main menu then choose **Go to Function...**.
Description:	Use this command to jump to a function in the active source window or pane.
	If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextGotoLine

Parameters:	LineNumber% Integer. The line number to go to.
Menu selection:	With a Source window or a Class Browser source pane active, select **Search** from the main menu then choose **Go to Line...**.
Description:	Use this command to position the cursor in the beginning of a specific line in the active source window or pane.

TextIndentBlock

Description: Use this command to insert a single tab character before each non-blank line in a selected block of text.

TextTab

Description: Use this command to insert one tab space at the cursor position. If there is a selection, this command will indent a block.

TextInsertFile

Description: Use this command opens the Insert File dialog so a different file may be selected for insertion into the current file at the cursor position.

If the Typing Replaces Selection option is currently disabled, as it is in the key file for Brief compatibility, and no text is selected this command enables it.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextLowercase

Description: Use this command to set all the characters to lower case in a selected block of text.

TextMatchDelimiter

Menu selection: With a Source window or a Class Browser source pane active, select **Search** from the main menu then choose **Go to Matching Delimiter**.

Description: Use this command to make the insertion point go just to the left of the delimiter that matches the one just to the right of the insertion point's current position.

TextNextPane

Description: Use this command to switch to the next text buffer in the buffer list.

TextPageDown

Description: Use this command to move the cursor down one page. If possible, the cursor column remains the same.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextPageUp

Description: Use this command to move the cursor up one page. If possible, the cursor column remains the same.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

EditPaste (TextPaste)

Menu selection: From the main menu select **Edit** then choose **Paste**.

Right-click in the Source window and select **Paste** from the pop-up menu.

Right-click in the Class Browser source pane and select **Paste** from the pop-up menu.

Right-click on an object in the Form Designer and select **Paste** from the pop-up menu.

Description: Use this command to insert the contents of the clipboard at the current cursor position.

When pasting text, this command will delete the selected text before pasting unless the Typing Replaces Selection option is currently disabled, as it is in the key file for Brief compatibility .

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextPrevPane

Description: Use this command to switch to the previous text buffer in the buffer list.

TextQueryValue

Description: Use this command to display a value of a token at run-time. Available only while in debugging mode.

TextReplace

Parameters: `Pattern$`
String. The pattern to search for.

`WholeWords?`
Boolean. If true, only whole words will be matched to the pattern.

`RegularExpression?`
Boolean. If true, evaluate regular expressions in the pattern.

`IgnoreCase?`
Boolean. If true, the search will be case-insensitive.

`ReplacePattern$`
String. The replacement pattern.

`RestrictToCurrentSelection?`
Boolean. If true, the replace operation will be confined to the current selection.

`Confirm?`
Boolean. If true, you will be asked to confirm each replacement.

Menu selection: With a Source window or a Class Browser source pane active, select **Search** from the main menu then choose **Replace...**.

Description: Use this command to find and replace occurrences of text in the active source window or pane with different text.

TextRevert

Description: Use this command to revert to the previously saved version of the current file.

EditSelectAll (TextSelectAll)

Menu selection: From the main menu select **Edit** then choose **Select All**.

Description: Use this command to select everything in the current window.

TextSelectCharLeft

Description: This command extends the current selection one character to the left of the cursor.

TextSelectCharRight

Description: This command extends the current selection one character to the right of the cursor.

TextSelectLine

Description: Use this command to select the entire current line.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextSelectLineDown

Description:　　This command extends the current selection down by one line.

TextSelectLineUp

Description: This command extends the current selection up by one line.

TextSelectPageDown

Description: This command extends the current selection down by one page.

TextSelectPageUp

Description: This command extends the current selection up by one page.

TextSelectToBol

Description: This command extends the current selection to the beginning of the line.

TextSelectToEnd

Description: This command extends the current selection to the end of the the file.

TextSelectToEol

Description: This command extends the current selection to the end of the line.

TextSelectToTop

Description: This command extends the current selection to the beginning of the file.

TextSelectWord

Description:　Use this command to select the word containing the cursor.

TextSelectWordLeft

Description: This command extends the current selection to the beginning of the word to the left.

TextSelectWordRight

Description: This command extends the current selection to the beginning of the word to the right.

TextShowTabs

Description: Use this command to toggle on and off showing of tabs. When on, indicates where tabs are located.

TextSpacesToTabs

Description: Use this command to substitute spaces for tabs without changing the layout of the text. The typical number of spaces a tab can replace is specified in the Tab Width field in Format page. From the main menu select **File** then **Environment Options...** then choose the Format page. Or right-click in the Source window then select **Format Options...** in the pop-up menu. Or right-click in the Source pane of the Class Browser then select **Format Options...** in the pop-up menu.

TextSplitLine

Description: Use this command to insert a line break, leaving the insertion point before the break.

TextStamp

Description: Use this command to insert the current date and time into the active file at the insertion point.

This command will delete the selection before inserting the date/time stamp unless the Typing Replaces Selection option is currently disabled, as it is in the key file for Brief compatibility.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextSwapMark

Description: Use this command to move the caret to the opposite mark within a selection.

TextTabLeft

Description: Use this command to move to the next tab position to the left of the insertion point.

TextTab

Description: If text is selected, shifts text to the right one tabstop. If no text is selected, inserts a tab, or the specified number of spaces, at the insertion point. In Insert mode, if the Change Tabs to Spaces option (Format Options dialog or Format page of Environment Options dialog) is selected for the active file, this command inserts a sufficient number of spaces at the current cursor location to advance the cursor to the next tab position. If the Change Tabs to Spaces option is not selected, this command inserts a single tab at the current cursor location.

TextTabsToSpaces

Description: Use this command to convert all tabs to the appropriate number of spaces without changing the layout of the text. The typical number of spaces required to replace a tab is specified in the Format Options dialog and the Environment Options dialog's Format page.

TextToBottom

Description: This command scrolls the source so that the line containing the cursor ends up at the bottom of the window. The file and the position of the cursor in the file is not changed.

TextToCenter

Description: This command scrolls the source so that the line containing the cursor ends up at the center of the window. The file and the position of the cursor in the file is not changed.

TextToggleColumnSelect

Description: Defines the beginning of a column block. If in column mark mode, this command turns off marking.

This command starts, ends, or switches Brief Selection Mode on or off.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextToggleExclusiveSelect

Description: Begins a non-inclusive block. New mark commands replace marks that already exist in the current buffer. This command starts, ends, or switches Brief Selection Mode on or off.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextToggleInsert

Description: Use this command to switch back and forth between Insert mode (default) and Overwrite mode. In Insert mode, each typed character is inserted in front of the characters at the current cursor location. In Overwrite mode, each typed character replaces the character at the cursor location.

TextToggleLineSelect

Description: Defines the beginning of a line block. If in line mark mode, this command turns off marking. This command starts, ends, or switches Brief Selection Mode on or off .

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextToggleMarkSelect

Description: Defines the beginning of a character block. If in select mark mode, this command turns off marking. This command starts, ends, or switches Brief Selection Mode on or off.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextToggleWordwrap

Menu selection: Right-click in a Source window or Class Browser Source pane and select **Format Options...** from the pop-up menu. Toggle the Word wrap checkbox.

Description: Use this command to enable or disable word-wrap in the active file. The right margin may be changed by using the Format Options dialog. Open by right-clicking in a Source window or Class Browser Source pane and selecting **Format Options...** from the pop-up menu.

TextTopOfWindow

Description: Use this command to move the cursor to the top line of the window. If possible, the cursor column remains the same.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextToTop

Description: This command scrolls the source so that the line containing the cursor ends up at the top of the window. The file and the position of the cursor in the file is not changed.

EditUndo (TextUndo, ProjectUndo)

Menu selection: From the main menu select **Edit** then choose **Undo**.

Description: Use this command to reverse the effects of the last editing command performed on the active window.

TextUnindentBlock

Description: Use this command to delete the first tab character before each text line in a selected block of text.

TextUnmarkBlock

Description: Use this command to deselect the selected text and return the cursor to where it was before the text was selected.

TextUppercase

Description: Use this command to make all characters in a selected block of text upper case.

TextWindowDown

Description: This commands scrolls the window down by one line.

TextWindowUp

Description: This command scrolls the window up by one line.

TextWordLeft

Description: Use this command to move the cursor to the beginning of the previous word or to the end of the previous line, whichever comes first.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextWordRight

Description: Use this command to move the cursor to the beginning of the next word or to the end of the line, whichever comes first.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

TextWrapPara

Description: Use this command to reformat the current paragraph (or the previous paragraph, if the cursor is between paragraphs) and leaves the cursor at the end of the reformatted paragraph.

TextWriteBlock

Menu selection: From the Text Editor, select **File**, then **Write Block.**

Description: Use this command to save the currently selected text to a file.

TextZoomPane

Description: Use this command to center the Source window on the screen and enlarge it to fill most of the screen.

DebugStepInto

Menu selection: While debugging select **Debug** from the main menu then choose **Step Into**.

Description: Executes the next line of source code in the current method. Calls to other methods are entered.

ClassGoto

Menu selection: With a class selected in the Hierarchy Editor choose **Hierarchy** from the main menu, then **Go To Source**, or right-click to get the context menu then select **Go To Source**.

With a class selected in the Class Browser choose **Classes** from the main menu, then **Go To Source**, or right -click on a class to get the context menu then select **Go To Source.**

Description: The file associated with the currently selected class is opened in a source window.

dbAddTableWizard

Description: This command runs a wizard which steps you through the process of adding dbAWARE
components to the form, based on database catalog information.

dbNavigator

Description: This command display a "tree" of database catalog information. The root level is a list of dbANYWHERE servers. The next level is a list of data sources at a given dbANYWHERE server. Under each data source is a list of tables, and under each table is a list of its columns.

DebugRestart

Menu selection: While debugging select **Debug** from the main menu then choose **Restart**.

Description: Restarts the current debugging session from the beginning.

DebugStepOut

Menu selection: While debugging select **Debug** from the main menu then choose **Step Out**.

Description: Executes till the current method returns.

SearchFindInFiles

Parameters: `Pattern$`
String. The pattern to search for.

`IgnoreCase?`
Boolean. If true, the search will be case-insensitive.

`RegularExpression?`
Boolean. If true, evaluate regular expressions in the pattern.

`WholeWords?`
Boolean. If true, only whole words will be matched to the pattern.

`Sources%`
Integer. Specifies which files to search. The following are the acceptable values and their meanings:

　　1=project files,

　　2=refine previous found files list

　　3=matching criteria.

`DirectoryName$`
String. The directory of the files to be searched.

`FileNameWildcard$`
String. Specifies which file names to search.

`IncludeSubdirectories?`
Boolean. If true, subdirectories will be included in the search.

`DateCreatedOrModified%`
Integer. Selects which files to search by date. The following are the acceptable values and their meanings:

　　0=ignore,

　　1=on,

　　2=not on,

　　3=before,

　　4=before or on,

　　5=after,

　　6=after or on.

`TimeCreatedOrModified%`
Integer. Selects which files to search by time. The following are the acceptable values and their meanings:

0=ignore,

1=at,

2=not at,

3=before,

4=before or at,

5=after,

6=after or at.

`Month%`
Integer. The month the file was created (1-12).

`Day%`
Integer. The day the file was created. (1-31).

`Year%`
Integer. The year the file was created.

`Hour%`
Integer. The hour the file was created.

`Minute%`
Integer. The minute the file was created.

`Reserved0%`
Integer. Reserved for future use. Set to 0.

`PM?`
Boolean. Specifies whether the time of file creation is PM (true) or AM (false).

`Reserved1%`
Integer. Reserved for future use. Set to 0.

`Reserved2%`
Integer. Reserved for future use. Set to 0.

`ReadOnly%`
Integer. Specifies what the read-only attribute of the searched files is set to:
    TRUE=1, FALSE=0, IGNORE=2.

`Hidden%`
Integer. Specifies what the hidden attribute of the searched files is set to:
    TRUE=1, FALSE=0, IGNORE=2.

`Archive%`
Integer. Specifies what the archive attribute of the searched files is set to:
    TRUE=1, FALSE=0, IGNORE=2.

`System%`
Integer. Specifies what the system attribute of the searched files is set to:
    TRUE=1, FALSE=0, IGNORE=2.

Menu selection: From the main menu select **Search** then **Find in Files...**.

Description: Use this command to perform a multi-file search. Regular expressions, attribute, and date-based file selection are available.

SearchGotoDefinition

Menu selection: With a Source window or Class Browser active select **Search** from the main menu then choose **Go to Definition**.

Right-click in the Source window, then select **Go to Definition** from the resulting pop-up menu.

Right-click in the Class Browser's Source pane, then select **Go to Definition** from the resulting pop-up menu.

Description: This command uses the selected name to find and display the matching class member implementation in the Class Browser. If the selected name is not a unique member name, then the Members window is opened with a list of all the declarations in all the project's classes which match the name currently selected in the source.

LiveUpdate

Menu selection: From the main menu select **Help**, then choose **Live Update**.

Description: Updates Visual Café from the internet.

ObjectGotoDefinition

Menu selection: Right-click on an object in the Project window then select **Go to Definition** from the pop-up menu.

Description: Displays the definition of the selected object in the Class Browser window.

ProjectMinimizeAll

Menu selection: Click on the system menu of the Project window then choose **Minimize All**.

Description: Minimizes all of the windows associated with the project.

TextQuickWatch

Menu selection: While debugging, select an object name in a source window then choose **Source** from the main menu then select **Add Watch**. Or right-click on an object name in a source window and select **Add Watch** from the resulting pop-up.

Description: Adds the object selected or pointed to in the source window to the debugger's Watch window.

TextBacktab

Description: If no text is selected, moves the insertion point to the previous tabstop. If text is selected shifts text to the left one tabstop.

TextFindNext

Description: Repeats the latest find command, scanning in the forward direction.

TextFindPrev

Description: Repeats the latest find command, scanning in the reverse direction.

TextGotoVariables

Description: This command activates the Variables window and changes the context it displays to correspond to the selection in the source window. It only works while debugging when the source window or the source pane of the Class Browser is active.

WindowNext

Description: Use this command to activate each window in sequence.

WindowVariables

Menu selection: From the main menu select Window then choose Variables.

Description: This command opens the Variables window.

ClassesGotoSource (ClassGoto)

Menu selection: With a class selected in the Hierarchy Editor choose **Hierarchy** from the main menu, then **Go To Source**, or right-click to get the context menu then select **Go To Source**.

With a class selected in the Class Browser choose **Classes** from the main menu, then **Go To Source**, or right -click on a class to get the context menu then select **Go To Source.**

Description: The file associated with the currently selected class is opened in a source window.

ClassesOptions (ClassOptions, MemberOptions)

Menu selection: With the Class Browser active select **Classes** from the main menu then choose **Options...**.

Right-click in the Class or Members pane of the Class Browser then select **Options...** from the pop-up menu.

Description: Opens the Class Options dialog which allows various options for the Class Browser to be viewed and set.

WindowClose

Menu selection: From the main menu, select **File**, then **Close**.

Description: Closes the currently active window.

DebugPause

Menu selection: With debugger running select **Debug**, then **Pause**.

Description: Suspends program execution during a debug session.

DebugStepOver

Menu selection: With debugger running select **Debug**, then **Step Over**.

Description: Executes the next line of source code in the current method. Calls to other methods are not entered.

FileClose

Menu selection: From the main menu, select **File**, then **Close**.
Description: Closes the currently active window.

FileEnvironmentOptions

Menu selection: From the main menu, select **File**, then **Environment Options...**.
Description: Opens the Environment Options dialog

FileNew

Menu selection: From the main menu, select **File**, then **New**.

Description: Creates a new untitled.source Window.

FileNewProject

Menu selection: From the main menu select **File**, then **New Project...**.

Description: Brings up the New Project dialog, allowing the user to create a new untitled project.

FileOpen

Menu selection: From the main menu select **File**, then **Open...**.

Description: Opens an existing file in a new source Window. The user will be prompted to specify the file name.

FilePrint (TextPrint)

Menu selection: When a source window or pane is active or the Hierarchy Editor, select **File** from the main menu then choose **Print**.

Description: Opens the Print dialog so the window contents may be printed.

FileSave (TextSave)

Menu selection: From the main menu select **File** then choose **Save**.

Description: Use this command to save the contents of the topmost window. If it is untitled, this command brings up the Save As dialog box.

FileSaveAs

Menu selection: From the main menu select **File** then choose **Save As...**.

Description: Use this command to bring up the Save As dialog box, which lets you save the current file under a new name.

SearchGotoBookmark(1-10)

Menu selection: From the main menu, select **Search**, then **Bookmarks...**. In the Bookmarks dialog select the desired bookmark and press the Go To button.

Description: Position cursor to previously defined bookmark, opening new source window if necessary.

MessagesCurrentError

Menu selection: From the main menu, select **Search**, then **Go to Current Error**.

Description: After a build or compile where errors occur this command opens the appropriate source window, positions the cursor to the location of the current error, hilites the line in error, and displays the error description at the bottom of the window. The current error is determined when the MessagesFirstError, MessagesNextError, and MessagesPrevError commands are used.

MessagesFirstError

Menu selection: From the main menu, select **Search**, then **Go to First Error**.

Description: After a build or compile where errors occur this command opens the appropriate source window, positions the cursor to the location of the first error, hilites the line in error, and displays the error description at the bottom of the window.

MessagesNextError

Menu selection: From the main menu, select **Search**, then **Go to Next Error**.

Description: After a build or compile where errors occur this command opens the appropriate source window, positions the cursor to the location of the next error, hilites the line in error, and displays the error description at the bottom of the window.

MessagesPreviousError

Menu selection: From the main menu, select **Search**, then **Go to Previous Error**.

Description: After a build or compile where errors occur this command opens the appropriate source window, positions the cursor to the location of the previous error, hilites the line in error, and displays the error description at the bottom of the window.

InsertApplet

Menu selection: From the main menu, select **Insert**, then **Applet...**.

Description: Opens the Insert Applet dialog allowing the user to add an Applet to the currently active project.

InsertClass (ClassInsert)

Parameters: ClassInterface
Class or interface flag, ignored, use a zero value

NewClassName$
String. Name of the new class to create and add.to the project.

FileName$
String. Name of the source file for the new class.

PackageName$
String. Ignored, use an empty string: ""

Access
Zero indicates a package class, one indicates a public class

Abstract
Ignored, use a zero value

Final
Ignored, use a zero value

BaseClassName$
String. Name of the new class's base class.

ToOverride$
String. Methods to override, ignored, use an empty string: ""

Menu selection: From main menu select **Insert** then **Class...**.

Right-click in the Class Browser's Class pane, then select **Insert Class...** from the resulting pop-up menu.

Description: Use this command to create a new class based on an already existing one, and add it to the project.

InsertComponent

Menu selection: From the main menu, select **Insert**, then **Component...**.

Description: Opens the Insert Object dialog to allow the insertion of a new component into the active project.

InsertForm

Menu selection: From the main menu, select **Insert**, then **Form...**.

Description: Opens the Insert Form dialog and allows the user to add a new form to the current project.

InsertGroup

Menu selection: With the Component Library window active, from the main menu select **Insert**, then **Group**.

Description: Adds a new group to the component library.

InsertMember (MemberInsert)

Parameters: `Declaration$`
String. The member declaration to insert.

`Reserved1$`
String. Reserved for future use.

`AccessKeyword%`
Constant that specifies the access level keyword used. The following are the acceptable values and their meanings:

    1=public,

    2=protected,

    3=private,

    4=package (no access keyword).

`Reserved1%`
Integer. Reserved for future use. Set to 0.

`Reserved2`
Integer. Reserved for future use. Set to 1.

Menu selection: With a class selected in the Class Browser, choose **Insert** from the main menu then select **Member...**.

Right-click in the Members pane of the Class Browser then select **Insert Member** from the pop-up menu.

Description: Use this command to add a member to the current class.

ObjectAddInteraction

Menu selection: With the Form Designer or the Project window's Objects page active, from the main menu select **Object**, then **Add Interaction...**. Or right-click in the Project window's Objects page to bring up the context menu and select **Add Interaction...**.

Description: Opens the Interaction Wizard, allowing the user choose the way objects interact.
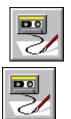
ObjectAddToLibrary

Menu selection: With the Form Designer or Project window active, from the main menu select **Object**, then **Add to Library...**.

Description: Opens the Add to Library dialog so the user can add an object to the Component Library.

ObjectEditObject

Menu selection: With the Form Designer or the Project window's Objects page active, from the main menu select **Object**, then **Edit**. Or right-click in the Project window's Objects page to bring up the context menu and select **Edit**.

Description: Opens the Form Designer so the user can visually edit an object.

ObjectEditSource

Menu selection: With the Form Designer or the Project window's Objects page active, from the main menu select **Object**, then **Edit Source**. Or right-click in the Project window's Objects page to bring up the context menu and select **Edit Source**.

Description: Displays the source code for the currently selected object.

ProjectAddItem

Menu selection: With a Source Window active and titled, from the main menu select **Project**, then select **Add**.

Description: Adds the file in the currently active Source Window to the project.

ProjectBuild

Menu selection: From the main menu select **Project**, then select **Build**.

Description: Builds the currently active project.

ProjectCompile

Menu selection: From the main menu select **Project**, then select **Compile**.

Description: Compiles the currently active source file.

ProjectCreateProjectTemplate

Menu selection: From the main menu select **Project**, then select **Create Project Template...**.

Description: Opens the Create Project Template dialog to allow the user to create a new project template.

ProjectDebug

Menu selection: From the main menu select **Project**, then select **Run in Debugger**.

Description: Runs the project under the debugger, building if needed.

ProjectExecute

Menu selection: From the main menu select **Project**, then select **Execute**.

Description: Runs the project, building if needed.

ProjectOptions

Menu selection: From the main menu select **Project**, then select **Options...**.

Description: Opens the Project Options dialog.

SearchFind

Menu selection: With the Source Window or the Class Broswer editing pane active, from the main menu select **Search**, then select **Find...**.

Description: Opens the Find dialog.

SearchFindAgain (TextFindAgain)

Menu selection: With the Source Window or the Class Broswer editing pane active, from the main menu select **Search**, then select **Find Again**.

Description: Finds the next occurance of the current search text.

If using Brief key file, see "Notes on Commands for Brief Compatibility".

SearchReplace

Menu selection: With the Source Window or the Class Broswer editing pane active, from the main menu select **Search**, then select **Search Replace...**.

Description: Opens the Replace dialog.

SearchClearBookmark(1-10)

Menu selection: From the main menu select **Search**, then select **Bookmarks...**, choose a bookmark then use the **Clear** button.

Description: Removes the specified bookmark.

TextClearAllBreakpoints

Menu selection: With the Breakpoint Window active select **Breakpoints** from the main menu then select **Clear All**.
Or right-click in Breakpoint Window to bring up context menu then select **Clear All**.

Description: Removes all previously set breakpoints.

TextSetConditionalBreakpoint

Menu selection: With the source window or the Class Broswer editing pane active, from the main menu select **Source**, then select **Set Conditional Breakpoint...**.

Description: Opens the Conditional Breakpoint dialog so that a conditional breakpoint may be inserted at the current location in the source.

TextInsertTab

Description: Inserts a tab character in the active Source Window or Class Browser editing pane at the current cursor location.

TextLoadFile

Parameters:	FileName$ String. Name of the source file to load.
Description:	Reads the specified file into the currently active source window. If the currently active source has been modified the user will be prompted for saving before the new file is loaded into the window.

TextViewEvents

Description: Shows or hides the Objects and Events/Methods drop down boxes in the currently active source window.

WindowBreakpoints

Menu selection: From the main menu select **Window**, then select **Breakpoints**.

Description: Opens the Breakpoint window.

WindowCallStack

Menu selection: From the main menu select **Window**, then select **Call Stack**.

Description: Opens the Call Stack window.

WindowClassBrowser

Menu selection: From the main menu select **Window**, then select **Class Browser**.

Description: Opens the Class Browser window.

WindowComponentLibrary

Menu selection: From the main menu select **Window**, then select **Component Library**.

Description: Opens the Component Library window.

WindowHierarchyEditor

Menu selection: From the main menu select **Window**, then select **Hierarchy Editor**.

Description: Opens the Hierarchy Editor window.

WindowMessages

Menu selection: From the main menu select **Window**, then select **Messages**.

Description: Opens the Messages window.

WindowNew

Menu selection: From the main menu select **Window**, then select **New**.

Description: Opens a new copy of the active Source or Class Browser window.

WindowPropertyList

Menu selection: From the main menu select **Window**, then select **Property List**.

Description: Opens the Property List window.

WindowThreads

Menu selection: From the main menu select **Window**, then select **Threads**.

Description: Opens theThreads window.

WindowWatch

Menu selection: From the main menu select **Window**, then select **Watch**.

Description: Opens the Watch window.

WorkspaceNew

Menu selection: From the main menu select **Window**, then select **Workspaces**, then select **New...**. Or right-click on the Workspace Palette to bring up the context menu then select **New...**.

Description: Opens the New Workspace dialog so a new workspace may be created.

WorkspaceDelete

Menu selection: From the main menu select **Window**, then select **Workspaces**, then select **Delete**. Or right-click on the Workspace Palette to bring up the context menu then select **Delete**.

Description: Deletes the currently active workspace.

WorkspaceRename

Menu selection: From the main menu select **Window**, then select **Workspaces**, then select **Rename...**. Or right-click on the Workspace Palette to bring up the context menu then select **Rename...**.

Description: Opens the Rename Workspace dialog so that the current workspace may be renamed.

Dialog Box Information

Use the Dialog Box Information dialog box to set the position or size of a dialog box template.

Position

X Position: The distance from the left border of the active window to the left border of this dialog box. Entering a higher number moves the dialog box farther to the right within the active window.
Y Position: The distance from the top border of the active window to the top border of this dialog box. Entering a higher number moves the dialog box farther down from the top of the active window.

Size

Width: The distance between the left and right borders of this dialog box. Enter a higher number to increase, or a lower number to decrease the width of this dialog box.
Height: The distance between the top and bottom borders of this dialog box. Enter a higher number to increase, or a lower number to decrease the height of this dialog box.
The distances are expressed in dialog units.
You can also set the position of a dialog box by dragging it in the Dialog Editor, or set the size by dragging the borders of the dialog box.

Style

Specify whether the close box and title bar are displayed in the check boxes.

Text$

Specify the text to appear in the title bar of your dialog box. You can use a constant or variable. You cannot leave this box blank.

Variable Name

If you want the text in the Title bar to be a variable, check this box. If you use a variable, you must define it in the macro above the dialog box template declaration.

Name

Specify the name of the dialog box template. This is the name you will use to identify   this template in your macro: it will never be displayed in an instance of the dialog box itself. Dialog Editor provides a default name of UserDialog. You cannot leave this box blank.

Function (optional)

Enter the name of a Symantec Basic function that is used in your dialog box.

Picture Library (optional)

Specify the library from which one or more picutres in the dialog box are obtained.

Variable Name

If you want the Picture Library bar to be a variable, check this box. If you use a variable, you must define it in the macro above the dialog box template declaration.

See Also:

Setting Dialog Box Attributes

Default OK Button Information

Use the Default OK Button Information dialog box to set the position or size of an OK button in your dialog box template.

Position

X Position: The distance from the left border of the dialog box to the left border of this button. Entering a higher number moves the button farther to the right within the dialog box.
Y Position: The distance from the top border of the dialog box to the top border of this button. Entering a higher number moves the button farther down from the top of the dialog box.

Size

Width: The distance between the left and right borders of this button. Enter a higher number to increase, or a lower number to decrease the width of this button.
Height: The distance between the top and bottom borders of this button. Enter a higher number to increase, or a lower number to decrease the height of this button.
The distances are expressed in dialog units.
You can also set the position of a button by dragging it in the dialog box template, or set the size by dragging the borders of the button.

Identifier (optional)

Enter the name by which you refer to the OK control in your Symantec Basic macro in this Idenifier box.

See Also:

OK Button (Controls Menu)

Default Cancel Button Information

Use the Default Cancel Button Information dialog box to set the position or size of a Cancel button in your dialog box template.

Identifier (optional)

Enter the name by which you refer to the Cancel control in your Symantec Basic macro in this Idenifier box.

See Also:

Cancel Button (Controls Menu)

Push Button Information

Use the Push Button Information dialog box to set the attributes of a Push button in your dialog box template.

Variable Name

If you want the text in the Text$ box to be a variable, check this box. If you use a variable, you must define it in the macro above the dialog box template declaration.

Text$

Enter the label to appear on the Push button.

Identifier (optional)

Enter the name by which you refer to the push button in your Symantec Basic macro in this Idenifier box.

See Also:

Push Button (Controls Menu)

Dialog Unit

A unit of measure relatively independent of the resolution of the monitor. A horizontal unit is based on the average character width of the font divided by 4, a vertical dialog unit is based on the character height of the font divided by 8. Since the characters in the font used by Dialog Editor are nearly twice as high as they are wide, horizontal and vertical units are roughly the same size.

Option Button Information

Use the Default Option Button Information dialog box to set the attributes of an Option button in your dialog box template.

Option Group

Specify the name for a group of option buttons in your macro in this box. The text you enter must be a valid identifier and the first character must be a period (which serves as the separator between the dialog box instance and the Option Group name). Each option button in a group must have the same name. You cannot leave this box blank.

Identifier (optional)

Enter the name by which you refer to the option button in your Symantec Basic macro in this Idenifier box.

Text$

 Enter the label to appear beside the Option button. The label can be a constant or a variable. Dialog Editor provides a default label of "Select Me." You cannot leave this box blank.

See Also:

[Option Button (Controls Menu)](#)

Check Box Information

Use the Check Box Information dialog box to set the attributes of a Check Box in your dialog box template.

Position

X Position: The distance from the left border of the dialog box to the left border of this box. Entering a higher number moves the box farther to the right within the dialog box.
Y Position: The distance from the top border of the dialog box to the top border of this box. Entering a higher number moves the box farther down from the top of the dialog box.

Size

Width: The distance between the left and right borders of this box. Enter a higher number to increase, or a lower number to decrease the width of this box.
Height: The distance between the top and bottom borders of this box. Enter a higher number to increase, or a lower number to decrease the height of this box.
The distances are expressed in dialog units.
You can also set the position of a box by dragging it in the dialog box template, or set the size by dragging the borders of the box.

Text$

Enter the label to appear beside the Check Box. The label can be a constant or a variable. Dialog Editor provides a default label of "Check Me." You cannot leave this box blank.

Variable Name

If you want the text in the Text$ box to be a variable check this box. If you use a variable, you must define it in the macro above the dialog box template declaration.

.Identifier

In this box, specify the field name identifying this Check box in your macro. The text you enter must be a valid identifier and the first character must be a period (which serves as the separator between the dialog box instance and the field name. Dialog Editor provides the default field name "CheckBox*n*". You cannot leave this box blank.

See Also:

[Check Box (Controls Menu)](#)

Group Box Information

Use the Group Box Information dialog box to set the attributes of a Group Box in your dialog box template.

Text$

Enter the label to appear at the top of the Group Box. The label can be a constant or a variable. Dialog Editor provides a default label of "Group Name." You cannot leave this box blank.

Identifier (optional)

Enter the name by which you refer to the group box in your Symantec Basic macro in this Idenifier box.

See Also:

Group Box (Controls Menu)

Text (Label) Information

Use the Text Information dialog box to set the attributes of a Text component of your dialog box template.

Text$

Enter the text you want to appear in your dialog box as free-standing text or as the label for a text box, list box, or combo box. The text can be a constant or a variable. Dialog Editor provides a default label of "Read me" in this box. You cannot leave this box blank.

Identifier (optional)

Enter the name by which you refer to the text label in your Symantec Basic macro in this Idenifier box.

Font (optional)

Use the Font button to change the font in which your text label will be displayed.

See Also:

Text (Controls Menu)

Text Box Information

Use the Text Box Information dialog box to set the attributes of a Text Box for user entered text in your dialog box template.

Multiline

Allows you to determine whether users can enter a single line of text or multiple lines.

Identifier (Mandatory)

Enter the name by which you refer to the text box in your Symantec Basic macro in this Idenifier box. Also contains the result of the control after the dialog box has been processed.

See Also:

Text Box (Controls Menu)

List Box Information

Use the List Box Information dialog box to set the attributes of a List Box in your dialog box template.

Array$

Specify the array name identifying this List box in your macro. You must enter a valid identifier. Dialog Editor provides a default name of "ListBox*n*$". You cannot leave this box blank.

.Identifier

In this box, specify the name in your macro for this List box. The text you enter must be a valid identifier and the first character must be a period (which serves as the separator between the dialog box instance and the field name). Dialog Editor provides the default field name "ListBox*n*". You cannot leave this box blank. Also contains the result of the control after the dialog box has been processed.

See Also:

List Box (Controls Menu)

Combo Box Information

Use the Combo Box Information dialog box to set the attributes of a Combo Box in your dialog box template.

Array$

Specify the array name identifying this List box in your macro. You must enter a valid identifier. Dialog Editor provides a default name of "ComboBox*n*$". You cannot leave this box blank.

.Identifier

In this box, specify the name in your macro for this Combo box. The text you enter must be a valid identifier and the first character must be a period (which serves as the separator between the dialog box instance and the field name). Dialog Editor provides the default field name "ComboBox*n*". You cannot leave this box blank. Also contains the result of the control after the dialog box has been processed.

Drop List Box Information

Use the Drop List Box Information dialog box to set the attributes of a drop List box in your dialog box template.

Array$

Specify the array name identifying this Drop List box in your macro. You must enter a valid identifier. Dialog Editor provides a default name of "DropListBox*n*$". You cannot leave this box blank.

.Identifier

In this box, specify the name in your macro for this Combo box. The text you enter must be a valid identifier and the first character must be a period (which serves as the separator between the dialog box instance and the field name). Dialog Editor provides the default field name "DropListBox*n*". You cannot leave this box blank. Also contains the result of the control after the dialog box has been processed.

Picture Information

Use the Picture Information dialog box to set the attributes of a Picture in your dialog box template.

Picture Source

Select file to to specify a Windows bitmap or metafile. Use the browse button to select the file from a standard windows open file dialog box.

Name$

Displays the filename including path of the Windows bitmap or metafile you have specified or the name of a picture that you want to diplay from a specified library.

.Identifier

Name by which you refer to the control in your Symantec Basic macro.

Frame

Allows you to display a 3-D frame.

Browse

Click the browse button to select the Windows bitmap or metafile you want to use from a standard Windows Select File dialog box.

Picture Button Information

Use the Picture Button Information dialog box to set the attributes of a Picture Button in your dialog box template.

Add Watch Dialog Box (Macro Editor)

Use the Add Watch dialog box to add a watchpoint when debugging a Symantec Basic macro.

Context

Choose from Local, Public, or Private context.

Watch Pane (Macro Editor)

The watch pane of the script editor window shows the values of selected variables while your macro is running.

Variables are added to the watch pane by selecting "Watch Variable" from the debug menu.   The Add Watch dialog will appear.

Modify Dialog Box (Macro Editor)

Use the Modify dialog box to change the value of a variable when debugging a Symantec Basic macro.

Name

Name of the variable to modify.

Value

New value for that variable

Variables

List of variables to choose from.   Click on a variable in this list and its name is placed in the name field.

Goto Line Dialog Box (Macro Editor)

Use the Goto Line dialog box to move the cursor to the desired line of the currently displayed macro.

Line Number

Number of the line where you would like the cursor moved.   Enter a number in this field and press the OK button.

Find Dialog Box (ScriptMaker)

Use this dialog box to search for text.

Find What

This drop-down listbox contains the text to be found.
You can initialize the pattern by selecting text before choosing Find. (The text must not span a line break.)
Otherwise, you may type the text you wish to find into the text box, or select text from previous search strings, stored in the drop-down list box.
The pattern may also be a regular expression (text containing wildcard characters), if the Regular Expression option is selected.

Match Case

Causes the Find engine to consider case when searching.

Find Next

This command continues the search begun by Find. The search resumes from the current insertion point and proceeds in the direction previously specified.
If the search is successful, the matching text is highlighted. Otherwise, the status line displays the message "Pattern not found."

Record Over Existing Default Macro Message Box

This message box warns you prior to recording over the existing default macro.   Click OK to record your macro.

Error (Failed to Save) Message Box

Failed to complete file save successfully.

Grid Setting Dialog Box

Use this dialog box to edit grid settings in the Dialog Editor.

Show Grid

Select Show Grid to enable a grid on your dialog template.

Spacing

You can select the size of the grid in this area.

Replace Dialog Box (ScriptMaker)

Use the Replace dialog box to find and replace text strings.

Replace With

Type the replacement string here; note that wildcards can not be used in the Replace With box. Also, you can contain the changes to a selected portion of the file's text.

Find Next Button

Repeats the search done by Find in the same direction.

Replace Button

Replaces selected occurrence.

Calls Dialog Box

Use the Calls button on the script editor toolbar to invoke the calls dialog box.   This button is only available while a macro is being debugged. The resulting dialog displays the current state of the macro's call stack.

Show

Select a macro subroutine from the list box and press the Show button to display the source code for that macro subroutine.

Close

When done using the Calls Dialog Box, press close to close the dialog.