

Softbridge Basic Language (SBL)

Reference Manual

© Copyright 1996 by Primavera Systems, Inc. All rights reserved. No part of this publication may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, mimeographing, recording, taping, or in information storage and retrieval systems—without written permission from the publisher.

Please send your comments about SureTrak Project Manager for Windows to:

Primavera Systems, Inc.
Two Bala Plaza
Bala Cynwyd, PA 19004
Telephone: 610-667-8600
FAX: 610-667-7894

Primavera Project Planner, P3, Finest Hour, Expedition, and Parade are registered trademarks, and PENGUIN, SureTrak Project Manager, Executive Summary Presentation, Monte Carlo, QuickRisk, Buy The Hour for Primavera, ReportSmith for Primavera, and Concentric Project Management are trademarks of Primavera Systems, Inc. All other brands and product names are trademarks or registered trademarks of their respective companies.

U.S. GOVERNMENT RESTRICTED RIGHTS: The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights 48 CFR 52.227-19, and our GSA contract, as applicable.



This manual is printed on recycled paper.

Using Basic

Softbridge Basic language (SBL) is a programming language delivered with SureTrak that provides power, ease of integration, and VBA compatibility. SBL supports standard Basic numeric, string, record and array data, along with dialog box records. SBL is similar to Visual Basic; for example, it can run any DLL. You should find it easy to use the SBL to create scripts that automate any variety of daily tasks. For example, you can do something as simple as repaint the screen on demand or for something as complex as retrieve data from another application, perform calculations with conditional expressions, bring the new data into SureTrak, and calculate new activity totals.

This section	Contains
SBL Functional List	All statements and functions in SBL organized by functional group, such as Dialog Boxes, Arrays, or Math Functions
SBL Basic Conventions	A list of topics that describe how to use features in SBL.
Dialog Boxes	Instructions for incorporating dialog boxes into your scripts.
Error Handling	Instructions for trapping errors in your script.
Expressions	Instructions for using logical expressions in your script.
Object Handling	Instructions for using objects, properties, and methods in your script.
Derived Trigonometric Functions	Instructions for using trigonometric functions in your script.
SBL Versus Other Basics	A comparison of SBL Basic to other versions of Basic.
SBL Compared to Visual Basic	A comparison SBL Basic to Visual Basic.

SBL Functional Index

This chapter contains a list of SBL statements and functions grouped by function.

Arrays

Erase	Reinitialize contents of an array
LBound	Return the lower bound of an array's dimension
ReDim	Declare dynamic arrays and reallocate memory
UBound	Return the upper bound of an array's dimension

Compiler Directives

\$CStrings	Treat backslash in string as an escape character as in 'C'
\$Include	Tell the compiler to include statements from another file
\$NoCStrings	Tell the compiler to treat a backslash as a normal character
Line Continuation	Continuing a long statement across multiple lines
Rem	Treat the remainder of the line as a comment

Control Flow

Call	Transfer control to a subprogram
Do...Loop	Control repetitive actions
Exit	Cause the current procedure or loop structure to return
For...Next	Loop a fixed number of times
GetCurValues	Retrieve current values for a dialog box
Goto	Send control to a line label
If ... Then ... Else	Branch on a conditional value
Let	Assign a value to a variable
Lset	Left-align one string or a user-defined variable within another
On...Goto	Branch to one of several labels depending upon value
Rset	Right-align one string within another
Select Case	Execute one of a series of statement blocks
Set	Set an object variable to a value
Stop	Stop program execution
While ... Wend	Control repetitive actions
With	Execute a series of statements on a specified variable

Dates & Times

Date Function	Return the current date
Date Statement	Set the system date
DateSerial	Return the date value for year, month, and day specified
DateValue	Return the date value for string specified
Day	Return the day of month component of a date-time value
Hour	Return the hour of day component of a date-time value
IsDate	Determine whether a value is a legal date.
Minute	Return the minute component of a date-time value
Month	Return the month component of a date-time value
Now	Return the current date and time
Second	Return the second component of a date-time value
Time Function	Return the current time
Time Statement	Set the current time
Timer	Return the number of seconds since midnight
TimeSerial	Return the time value for hour, minute, and second specified
TimeValue	Return the time value for string specified
Weekday	Return the day of the week for the specified date-time value
Year	Return the year component of a date-time value

Declarations

Const	Declare a symbolic constant
Declare	Forward declare a procedure in the same module or in a dynamic link library
Deftype	Declare the default data type for variables
Dim	Declare variables
Function ... End Function	Define a function
Global	Declare a global variable
Option Base	Declare the default lower bound for array dimensions
Option Compare	Declare the default case sensitivity for string comparisons
Option Explicit	Force all variables to be explicitly declared
ReDim	Declare dynamic arrays and reallocate memory
Static	Define a static variable or subprogram
Sub ... End Sub	Define a subprogram
Type	Declare a user-defined data type

Dialog Boxes

Defining Dialog Boxes

Begin Dialog	Begin a dialog box definition
Button	Define a button dialog box control
ButtonGroup	Begin definition of a group of button dialog box controls
CancelButton	Define a Cancel button dialog box control
Caption	Define the title of a dialog box
CheckBox	Define a checkbox dialog box control
ComboBox	Define a combo box dialog box control
DropComboBox	Define a drop-down combo box dialog box control
DropListBox	Define a drop-down list box dialog box control
GroupBox	Define a group box in a dialog box
ListBox	Define a list box dialog box control
OKButton	Define an OK button dialog box control
OptionButton	Define an OptionButton dialog box control
OptionGroup	Begin definition of a group of OptionButton dialog box controls
Picture	Define a Picture control
PushButton	Define a pushbutton dialog box control
StaticComboBox	Define a static combo box dialog box control
Text	Define a line of text in a dialog box
TextBox	Define a text box in a dialog box

Running Dialog Boxes

Dialog Function	Display a dialog box and return the button pressed
Dialog Statement	Display a dialog box
DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Tell whether a dialog control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgFocus Function	Return ID of the dialog control having input focus
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgSetPicture	Change the picture in the Picture control

Dialog Boxes

DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Tell whether a control is visible or hidden
DlgVisible Statement	Show or hide a dialog control

Environment Control

AppActivate	Activate another application
Command	Return the command line specified when the MAIN sub was run
Date Statement	Set the current date
DoEvents	Let operating system process messages
Environ	Return a string from the operating system's environment
Randomize	Initialize the random-number generator
SendKeys	Send keystrokes to another application
Shell	Run an executable program

Errors

Assert	Trigger an error if a condition is false
Erl	Return the line number where a run-time error occurred
Err Function	Return a run-time error code
Err Statement	Set the run-time error code
Error	Generate an error condition
Error Function	Return a string representing an error
On Error	Control run-time error handling
Resume	End an error-handling routine
Trappable Errors	Errors which can be trapped by SBL code

Files

Disk and Directory Control

ChDir	Change the default directory for a drive
ChDrive	Change the default drive
CurDir	Return the current directory for a drive
Dir	Return a filename which matches a pattern
MkDir	Make a directory on a disk
RmDir	Remove a directory from a disk

File Control

FileAttr	Return information about an open file
FileCopy	Copy a file
FileDateTime	Return modification date and time of a specified file
FileLen	Return the length of specified file in bytes
GetAttr	Return attributes of specified file, directory or volume label
Kill	Delete files from a disk
Name	Rename a disk file
SetAttr	Set attribute information for a file

File Input/Output

Close	Close a file
Eof	Check for end of file
FreeFile	Return the next unused file number
Get	Read bytes from a file
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
Line Input	Read a line from a sequential file
Loc	Return current position of an open file
Lock, Unlock	Control access to some/all of open file by other processes
Lof	Return the length of an open file
Open	Open a disk file or device for I/O
Print	Print data to a file or to the screen
Put	Write data to an open file
Reset	Close all open disk files
Seek Function	Return the current position for a file
Seek Statement	Set the current position for a file
Spc	Output given number of spaces
Tab	Move print position to the given column
Width	Set output-line width for an open file
Write	Write data to a sequential file

Math Functions

Financial Functions

FV	Return future value of a cash flow stream
IPmt	Return interest payment for a given period
IRR	Return internal rate of return for a cash flow stream
NPV	Return net present value of a cash flow stream
Pmt	Return a constant payment per period for an annuity
PPmt	Return principal payment for a given period
PV	Return present value of a future stream of cash flows
Rate	Return interest rate per period

Numeric Functions

Abs	Return the absolute value of a number
Exp	Return the value of e raised to a power
Fix	Return the integer part of a number
Int	Return the integer part of a number
IsNumeric	Determine whether a value is a legal number
Log	Return the natural logarithm of a value
Rnd	Return a random number
Sgn	Return a value indicating the sign of a number
Sqr	Return the square root of a number
Derived Functions	How to compute other numeric functions

Trigonometric Functions

Atn	Return the arc tangent of a number
Cos	Return the cosine of an angle.
Sin	Return the sine of an angle
Tan	Return the tangent of an angle
Derived Functions	How to compute other trigonometric functions

Objects

CreateObject	Create an OLE2 automation object
GetObject	Retrieve an OLE2 object from a file or get the active OLE2 object for an OLE2 class
Is	Determine whether two object variables refer to the same object
Me	Get the current object

Objects

New	Allocate and initialize a new OLE2 object
Nothing	Set an object variable to not refer to an object
Object	Declare an OLE2 automation object
Typeof	Check the class of an object
With	Execute statements on an object or a user-defined type

Screen Input/Output

Beep	Produce a short beeping tone through the speaker
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
InputBox	Display a dialog box which prompts for input
MsgBox Function	Display a Windows message box
MsgBox Statement	Display a Windows message box
PasswordBox	Display a dialog box which prompts for input. Don't echo input.
Print	Print data to a file or to the screen

Strings

String Functions

GetField	Return a substring from a delimited source string
Hex	Return the hexadecimal representation of a number, as a string
InStr	Return the position of one string within another
LCase	Convert a string to lower case
Left	Return the left portion of a string
Len	Return the length of a string or size of a variable
Like Operator	Compare a string against a pattern
LTrim	Remove leading spaces from a string
Mid Fun866	Return a portion of a string
Mid Statement	Replace a portion of a string with another string
Oct	Return the octal representation of a number, as a string
Right	Return the right portion of a string
RTrim	Remove trailing spaces from a string

Strings

SetField	Replace a substring within a delimited target string
Space	Return a string of spaces
Str	Return the string representation of a number
StrComp	Compare two strings.
String	Return a string consisting of a repeated character
Trim	Remove leading and trailing spaces from a string
UCase	Convert a string to upper case

String Conversions

Asc	Return an integer corresponding to a character code
CCur	Convert a value to currency
CDbl	Convert a value to double-precision floating point
Chr	Convert a character code to a string
CInt	Convert a value to an integer by rounding
CLng	Convert a value to a long by rounding
CSng	Convert a value to single-precision floating point
CStr	Convert a value to a string
CVar	Convert an number or string to a variant
CVDat	Convert a value to a variant date
Format	Convert a value to a string using a picture format
Val	Convert a string to a number

Variants

IsEmpty	Determine whether a variant has been initialized
IsNull	Determine whether a variant contains a NULL value
Null	Return a null variant
VarType	Return the type of data stored in a variant

SBL Basic Conventions

SBL uses the programming conventions described in this section.

Arguments Arguments to subprograms and functions you write are listed after the subprogram or function and may or may not be enclosed in parentheses. Whether you use parentheses depends on how you want to pass the argument to the subprogram or function: either by value or by reference.

If an argument is passed by value, it means that the variable used for that argument retains its value when the subprogram or function returns to the caller. If an argument is passed by reference, it means that the variable's value may be (and probably will be) changed for the calling procedure. For example, suppose you set the value of a variable, *x*, to 5 and pass *x* as an argument to a subprogram, named *mysub*. If you pass *x* by value to *mysub*, the value of *x* is still 5 after *mysub* returns. If you pass *x* by reference to *mysub*, however, *x* could be 5 or any other value resulting from the actions of *mysub*.

To pass an argument by value, use one of the following syntax options:

```
Call mysub((x))
mysub(x)
Call mysub(x byVal)
mysub x byVal
y=myfunction(x)
Call myfunction((x))
```

To pass an argument by reference, use one of the following options:

```
Call mysub(x)
mysub x
y=myfunction x
Call myfunction(x)
```

Externally declared subroutines and functions (such as DLL functions) can be declared to take byVal arguments in their declaration. In that case, those arguments are always passed byVal.

Arrays Array dimensions are enclosed in parentheses after the array name:

```
arrayname(a,b,c)
```

Comments Comments are preceded by an apostrophe and may appear on their own line in a procedure or directly after a statement or function on the same line:

```
'this comment is on its own line
Dim i as Integer 'this comment is on the code line
```

Line Continuation Long statements may be continued across more than one line by typing a space-underscore at the end of a line and continuing the statement on the next line. (You may add a comment after the underscore.)

```
Dim trMonth As Integer _      'month of transaction
    trYear As Integer        ' year of transaction
```

Records Elements in a record are identified using the following syntax:

```
record.element
```

where *record* is the previously defined record name and *element* is a member of that record.

Typographic Conventions This chapter uses the following typographic conventions:

To represent	Documentation syntax is
Statements and functions	Boldface; initial letter uppercase: Abs Len(<i>variable</i>)
Arguments to statements or functions	All lowercase, italicized letters: <i>variable</i> , <i>rate</i> , <i>prompt</i> \$
Optional arguments and/or characters	Italicized arguments and/or characters in brackets: [, <i>caption</i> \$], [<i>type</i> \$], [\$]
Required choice for an argument from a list of choices	List inside braces, with OR operator () separating choices: {Goto <i>label</i> Resume Next Goto 0}

Data Types Basic is a strongly typed language. Variables can be declared implicitly on first reference by using a type character; if no type character is present, the default type of **Variant** is assumed. Alternatively, the type of a variable can be declared explicitly with the **Dim** statement. In either case, the variable can only contain data of the declared type. Variables of user-defined type must be explicitly declared. SBL supports standard Basic numeric, string, record and array data. SBL also supports Dialog Box Records and Objects (which are defined by the application).

Arrays Arrays are created by specifying one or more subscripts at declaration or **Redim** time. Subscripts specify the beginning and ending index for each dimension. If only an ending index is specified, the beginning index depends on the **Option Base** setting. Array elements are referenced by enclosing the proper number of index values in parentheses after the array name, for example, *arrayname(i,j,k)*. See the **Dim** statement for more information.

Numbers The five numeric types are:

Type	From	To
Integer	-32,768	32,767
Long	-2,147,483,648	2,147,483,647
Single	-3.402823e+38 0.0, 1.401298e-45	-1.401298e-45, 3.402823466e+38
Double	-1.797693134862315d+308 0.0, 2.2250738585072014d-308	-4.94065645841247d-308, 1.797693134862315d+308
Currency	-922,337,203,685,477.5808	922,337,203,685,477.5807

Numeric values are always signed.

Basic has no true Boolean variables. Basic considers 0 to be FALSE and any other numeric value to be TRUE. Only numeric values can be used as Booleans. Comparison operator expressions always return 0 for FALSE and -1 for TRUE.

Integer constants can be expressed in decimal, octal, or hexadecimal notation. Decimal constants are expressed by simply using the decimal representation. To represent an octal value, precede the constant with “&O” or “&o” (for example, &o177). To represent a hexadecimal value, precede the constant with “&H” or “&h” (for example, &H8001).

Records A record, or record variable, is a data structure containing one or more elements, each of which has a value. Before declaring a record variable, a **Type** must be defined. Once the **Type** is defined, the variable can be declared to be of that type. The variable name should not have a type character suffix. Record elements are referenced using dot notation, for example, *varname.elementname*. Records can contain elements which are themselves records.

Dialog box records look like any other user-defined data type. Elements are referenced using the same *recname.elementname* syntax. The difference is that each element is tied to an element of a dialog box. Some dialog boxes are defined by the application, others by the user. See the **Begin Dialog** statement for more information.

Strings Basic strings can be either fixed or dynamic. Fixed strings have a length specified when they are defined, and the length cannot be changed. Fixed strings cannot be of 0 length. Dynamic strings have no specified length. Any string can vary in length from 0 to 32,767 characters. There are no restrictions on the characters which can be included in a string. For example, the character whose ANSI value is 0 can be embedded in strings.

Data Type Conversions Basic will automatically convert data between any two numeric types. When converting from a larger type to a smaller type (for example **Long** to **Integer**), a runtime numeric overflow may occur. This indicates that the number of the larger type is too large for the target data type. Loss of precision is not a runtime error (for example, when converting from **Double** to **Single**, or from either float type to either integer type).

Basic will also automatically convert between fixed strings and dynamic strings. When converting a fixed string to dynamic, a dynamic string which has the same length and contents as the fixed string will be created. When converting from a dynamic string to a fixed string, some adjustment may be required. If the dynamic string is shorter than the fixed string, the resulting fixed string will be extended with spaces. If the dynamic string is longer than the fixed string, the resulting fixed string will be a truncated version of the dynamic string. No runtime errors are caused by string conversions.

Basic will automatically convert between any data type and **variants**. Basic will convert variant strings to numbers when required. A type mismatch error will occur if the variant string does not contain a valid representation of the required number.

No other implicit conversions are supported. In particular, Basic will not automatically convert between numeric and string data. Use the functions **Val** and **Str\$** for such conversions.

Dynamic Arrays

Dynamic arrays differ from fixed arrays in that you do not specify a subscript range for the array elements when you dimension the array. Instead, the subscript range is set using the **Redim** statement. With dynamic arrays, you can set the size of the array elements based on other conditions in your procedure. For example, you may want to use an array to store a set of values entered by the user, but you don't know in advance how many values the user has. In this case, you dimension the array without specifying a subscript range and then execute a **ReDim** statement each time the user enters a new value. Or, you might want to prompt for the number of values a user has and execute one **ReDim** statement to set the size of the array before prompting for the values.

If you use **ReDim** to change the size of an array and want to preserve the contents of the array at the same time, be sure to include the **Preserve** argument to the **ReDim** statement.

If you **Dim** a dynamic array before using it, the maximum number of dimensions it can have is 8. To create dynamic arrays with more dimensions (up to 60), do not **Dim** the array at all; instead use just the **ReDim** statement inside your procedure.

The following procedure uses a dynamic array, *varray*, to hold cash flow values entered by the user:

```
Sub main
    Dim aprate as Single
    Dim varray() as Double
    Dim cflowper as Integer
    Dim msgtext
    Dim x as Integer
    Dim netpv as Double
    cflowper=InputBox("Enter number of cash flow periods")
    ReDim varray(cflowper)
    For x= 1 to cflowper
        varray(x)=InputBox("Enter cash flow amount for period #" & x & ":")
    Next x
    aprate=InputBox("Enter discount rate: ")
    If aprate>1 then
        aprate=aprate/100
    End If
    netpv=NPV(aprate,varray())
    msgtext="The net present value is: "
    msgtext=msgtext & Format(netpv, "Currency")
    MsgBox msgtext
End Sub
```

Variant Data Type

The variant data type may be used to define variables that contain any type of data. A tag is stored with the variant data to identify the type of data that it currently contains. You may examine the tag by using the **VarType** function.

A variant may contain a value of any of the following types:

Type/Name	Size of Data	Range
0 (Empty)	0	N/A
1 Null	0	N/A
2 Integer	2 bytes (short)	-32768 to 32767
3 Long	4 bytes (long)	-2.147E9 to 2.147E9
4 Single	4 bytes (float)	-3.402E38 to -1.401E-45 (negative) 1.401E-45 to 3.402E38 (positive)
5 Double	8 bytes (double)	-1.797E308 to -4.94E-324 (negative) 4.94E-324 to 1.797E308 (positive)
6 Currency	8 bytes (fixed)	-9.223E14 to 9.223E14
7 Date	8 bytes (double)	Jan 1st, 100 to Dec 31st, 9999
8 String	0 to ~64kbytes	0 to ~64k characters
9 Object	N/A	N/A

Any newly defined Variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty Variant converts to zero when used in a numeric expression, or an empty string in a string expression. You may test whether a variant is uninitialized (empty) with the **IsEmpty** function.

Null variants have no associated data and serve only to represent invalid or ambiguous results. You may test whether a variant contains a null value with the **IsNull** function. Null is not the same as Empty, which indicates that a variant has not yet been initialized.

Dialog Boxes

You can use SBL to create dialog boxes for your scripts.

Step 1: Define a Dialog Box

The **Begin Dialog... End Dialog** statements define a dialog box. The last parameter to the Begin Dialog statement is the name of a function, prefixed by a period (.). This function handles interactions between the dialog box and the user.

The Begin Dialog statement supplies three parameters to your function: an **identifier** (a dialog control ID), the **action** taken on the control, and a **value** with additional action information. Your function should have these three arguments as input parameters. See the Begin Dialog...End Dialog statement for more information.

Step 2: Write a Dialog Box Function

This function defines dialog box behavior. For example, your function could disable a check box, based on a user's action. The body of the function uses the "Dlg"-prefixed SBL statements and functions to define dialog box actions.

Define the function itself using the **Function...End Function** statement or declare it using the **Declare** statement *before* using the **Begin Dialog** statement. Enter the name of the function as the last argument to Begin Dialog. The function receives three parameters from Begin Dialog and returns a value. Return a non-zero value to leave the dialog box open after the user clicks a command button (such as Help).

Step 3: Display the Dialog Box

You use the **Dialog** function (or statement) to display a dialog box. The argument to Dialog is a variable name that you previously dimensioned as a dialog box record. The name of the dialog box record comes from the **Begin Dialog... End Dialog** statement. The return values for the Dialog function determine which key was pressed: -1 for OK, 0 for Cancel, >0 for a command button. If you use the **Dialog** statement, it returns an error if the user presses Cancel, which you can then trap with the **On Error** statement.

To create and run a dialog box, follow these three steps:

- 1 Define a dialog box record using the Begin Dialog...End Dialog statements and the dialog box definition statements such as TextBox, OKButton.
- 2 Create a function to handle dialog box interactions using the Dialog Functions and Statements (optional).
- 3 Display the dialog box using either the Dialog Function or Dialog Statement.

Dialog Functions and Statements

The function you create uses the “Dlg” dialog functions and statements to manipulate the active dialog box. This is the *only* function that can use these functions and statements. The list of the “Dlg” functions and statements is as follows:

Functions & Statements	Description
DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Tell whether a control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgFocus Function	Return ID of the dialog control having input focus.
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with a dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Tell whether a control is visible or disabled
DlgVisible Statement	Show or hide a dialog control

Most of these functions and statements take control ID as their first argument. For example, if a checkbox was defined with the following statement:

```
CheckBox 20, 30, 50, 15, "My check box", .Check1
```

Then **DlgEnable "Check1"**, 1 enables the checkbox, and **DlgValue("Check1")** returns 1 if the checkbox is currently checked, 0 if not. Note that the IDs are case-sensitive and do not include the dot which appears before the ID. Dialog functions and statements can also work with numeric IDs. Numeric IDs depend on the order in which dialog controls are defined.

For example, if the checkbox that we considered was the first control defined in the dialog record, then **DlgValue(0)** would be equivalent to **DlgValue("Check1")**. (The control numbering begins from 0, and the **Caption** control does not count.) Find the numeric ID using the **DlgControlID** function.

Note that for some controls (such as buttons and texts) the last argument in the control definition, ID, is optional. If it is not specified, the text of the control becomes its ID. For example, the Cancel button can be referred as "Cancel" if its ID was not specified in the **CancelButton** statement.

Error Handling

SBL contains three error handling statements and functions for trapping errors in your program: **Err**, **Error**, and **On Error**. SBL returns a code for many of the possible runtime errors you may encounter. See **Trappable Errors** for a complete list of codes.

In addition to the errors trapped by SBL, you may want to create your own set of codes for trapping errors specific to your program. You would do this if, for example, your program establishes rules for file input and the user does not follow the rules. You can trigger an error and respond appropriately using the same statements and functions you would use for SBL-returned error codes.

Regardless of the error trapped, you have one of two methods to handle errors; one is to put error-handling code directly before a line of code where an error may occur (such as after a File Open statement), and the other is to label a separate section of the procedure just for error handling, and force a jump to that label if any error occurs. The **On Error** statement handles both options.

Trapping Errors Returned by SBL

Option 1, Within Body of Code: The **On Error** statement identifies the line of code to go to in case of an error. In this case, the Resume Next parameter means execution continues with the next line of code after the error. In this example, the line of code to handle errors is the **If** statement. It uses the **Err** statement to determine which error code is returned.

Option 2, Using Error Handler: The **On Error** statement used here specifies a label to jump to in case of errors. The code segment is part of the main procedure and uses the **Err** statement to determine which error code is returned. To make sure your code doesn't accidentally fall through to the error handler, precede it with an **Exit** statement.

Trapping User-Defined (Non-SBL) Errors

These code examples show the two ways to set and trap user-defined errors. Both options use the **Error** statement to set the user-defined error to the value 30000. To trap the error, option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any user-defined errors.

Trappable Errors

The following table lists the runtime errors which SBL returns. These errors can be trapped by **On Error**. The **Err** function can be used to query the error code, and the **Error** function can be used to query the error text.

Error code	Error Text
5	Illegal function call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition
11	Division by zero
13	Type Mismatch
14	Out of string space
19	No Resume
20	Resume without error
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
58	File already exists
61	Disk full

Error code	Error Text
62	Input past end of file
63	Bad record number
64	Bad file name
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable set to Nothing
93	Invalid pattern
94	Illegal use of NULL
102	Command failed
429	Object creation failed
438	No such property or method
439	Argument type mismatch
440	Object error
901	Input buffer would be larger than 64K
902	Operating system error
903	External procedure not found
904	Global variable type mismatch
905	User-defined type mismatch
906	External procedure interface mismatch
907	Pushbutton required
908	Module has no MAIN
910	Dialog box not declared

Expressions

An expression is a collection of two or more terms that perform a mathematical or logical operation. The terms are usually either variables or functions that are combined with an operator to evaluate to a string or numeric result. You use expressions to perform calculations, manipulate variables, or concatenate strings.

Expressions are evaluated according to precedence order. Use parentheses to override the default precedence order.

The precedence order (from high to low) for the operators is as follows:

- Numeric Operators
- String Operators
- Comparison Operators
- Logical Operators

Numeric Operators

^	Exponentiation
-, +	Unary minus and plus
*, /	Numeric multiplication or division. For division, the result is a Double .
\	Integer division. The operands can be Integer or Long .
Mod	Modulus or Remainder. The operands can be Integer or Long .
-, +	Numeric addition and subtraction. The + operator can also be used for string concatenation.

String Operators

&	String concatenation
+	String concatenation

Comparison Operators (Numeric and String)

>	Greater than
<	Less than
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

For numbers, the operands are widened to the least common type (**Integer** is preferred over **Long**, which is preferred over **Single**, which is preferred over **Double**). For **Strings**, the comparison is case-sensitive, and based on the collating sequence used by the language specified by the user using the Windows Control Panel. The result is 0 for FALSE and -1 for TRUE.

Logical Operators

Not	Unary Not - operand can be Integer or Long. The operation is performed bitwise (one's complement).
And	And - operands can be Integer or Long. The operation is performed bitwise.
Or	Inclusive Or - operands can be Integer or Long. The operation is performed bitwise.
Xor	Exclusive Or - operands can be Integer or Long. The operation is performed bitwise.
Eqv	Equivalence - operands can be Integer or Long. The operation is performed bitwise. (A Eqv B) is the same as (Not (A Xor B)).
Imp	Implication - operands can be Integer or <i>Long</i> . The operation is performed bitwise. (A Imp B) is the same as ((Not A) OR B).

Object Handling

Objects are the end products of a software application, such as a spreadsheet, graph, or document. Each software application has its own set of properties and methods that change the characteristics of an object.

Properties affect how an object behaves. For example, width is a property of a range of cells in a spreadsheet, colors are a property of graphs, and margins are a property of word processing documents.

Methods cause the application to do something to an object. Examples are Calculate for a spreadsheet, Snap to Grid for a graph, and AutoSave for a document.

In SBL, you can access an object and use the originating software application to change properties and methods of that object. Before you can use an object in a procedure, however, you must access the software application associated with the object by assigning it to an object variable. Then you attach an object name (with or without properties and methods) to the variable to manipulate the object.

Step 1: Create an object variable to access the application

The **Dim** statement creates an object variable called “visio” and assigns the application, Visio, to it. The **Set** statement assigns the Visio application to the variable visio using either **GetObject** or **CreateObject**. You use **GetObject** if the application is already open on the Windows desktop. Use **CreateObject** if the application is not open.

Step 2: Use methods and properties to act on objects.

To access an object, property or method, use this syntax:

```
appvariable.object.property
appvariable.object.method
```

For example, **visio.document.count** is a value returned by the Count method of the Document object for the Visio application, which is assigned to the Integer variable doccount.

Alternatively, you can create a second object variable and assign the Document object to it using Visio’s Document method, as the Set statement shows.

Derived Trigonometric Functions

A number of trigonometric functions may be written in Basic using the built-in functions. The following table lists several of these functions:

Function	Computed by
Secant	$\text{Sec}(x) = 1/\text{Cos}(x)$
CoSecant	$\text{CoSec}(x) = 1/\text{Sin}(x)$
CoTangent	$\text{CoTan}(x) = 1/\text{Tan}(x)$
ArcSine	$\text{ArcSin}(x) = \text{Atn}(x/\text{Sqr}(-x*x+1))$
ArcCosine	$\text{ArcCos}(x) = \text{Atn}(-x/\text{Sqr}(-x*x+1))+1.5708$
ArcSecant	$\text{ArcSec}(x) = \text{Atn}(x/\text{Sqr}(x*x-1))+\text{Sgn}(x-1)*1.5708$
ArcCoSecant	$\text{ArcCoSec}(x) = \text{Atn}(x/\text{Sqr}(x*x-1))+(\text{Sgn}(x)-1)*1.5708$
ArcCoTangent	$\text{ArcTan}(x) = \text{Atn}(x)+1.5708$
Hyperbolic Sine	$\text{HSin}(x) = (\text{Exp}(x)-\text{Exp}(-x))/2$
Hyperbolic Cosine	$\text{HCos}(x) = (\text{Exp}(x)+\text{Exp}(-x))/2$

Function	Computed by
Hyperbolic Tangent	$\text{HTan}(x) = (\text{Exp}(x) - \text{Exp}(-x)) / (\text{Exp}(x) + \text{Exp}(-x))$
Hyperbolic Secant	$\text{HSec}(x) = 2 / (\text{Exp}(x) + \text{Exp}(-x))$
Hyperbolic CoSecant	$\text{HCoSec}(x) = 2 / (\text{Exp}(x) - \text{Exp}(-x))$
Hyperbolic Cotangent	$\text{HCotan}(x) = (\text{Exp}(x) + \text{Exp}(-x)) / (\text{Exp}(x) - \text{Exp}(-x))$
Hyperbolic ArcSine	$\text{HArcSin}(x) = \text{Log}(x + \text{Sqr}(x^2 + 1))$
Hyperbolic ArcCosine	$\text{HArcCos}(x) = \text{Log}(x + \text{Sqr}(x^2 - 1))$
Hyperbolic ArcTangent	$\text{HArcTan}(x) = \text{Log}((1+x)/(1-x))/2$
Hyperbolic ArcSecant	$\text{HArcSec}(x) = \text{Log}((\text{Sqr}(-x^2 + 1) + 1)/x)$
Hyperbolic ArcCoSecant	$\text{HArCoSec}(x) = \text{Log}((\text{Sgn}(x) * \text{Sqr}(x^2 + 1) + 1)/x)$
Hyperbolic ArcCoTangent	$\text{HArCoTan}(x) = \text{Log}((x+1)/(x-1))/2$

SBL Versus Other Basics

If you are familiar with older versions of Basic (those that predate Windows), you will notice that SBL includes many new features and changes from the language you have learned. SBL more closely resembles other higher level languages popular today, such as C++ and Pascal. The topics listed here describe some of the differences you will notice between the older Basics and SBL.

Line Numbers and Labels

The line numbers used in earlier Basics are no longer required. To reference a line of code, you use a label. A label is a single word followed by a colon, which is placed at the beginning of a line of code.

Subprograms and Global Variables

SBL is more modular, with code divided into subprograms and functions. The subprograms and functions you write use the SBL statements and functions to perform actions. In SBL, the first subprogram executed must be named “main” and take no arguments (and contain no parentheses). You use the **Sub...End Sub** statements to define it, as in the example that follows:

```
Sub main
    MsgBox "Hello, World"
End Sub
```

The Main subprogram can then call other subprograms or functions included in a .SBL file.

The placement of variable declarations determines their scope:

Scope	Definition
Local	Dimensioned inside a subroutine or function. The variable is accessible only to the subroutine or function that dimensioned it.
Module	Dimensioned outside any subroutine or function. The variable is accessible to any subroutine or function in the same file.
Global	Dimensioned outside any subroutine or function using the Global statement. The variable is accessible to any subroutine or function in any module (file).

Data Types

In addition to the standard data types—numeric, string, array, and record—SBL includes Variants and objects. Variables that are defined as Variants can store any type of data. For example, the same variable could hold integers one time and strings later in a procedure. Objects give you the ability to manipulate complex data supplied by an application, such as windows, forms or OLE2 objects.

Dialog Box Handling

SBL contains extensive dialog box statements and functions to give you great flexibility in creating and running custom dialog boxes. You define the contents of a dialog box using dialog statements and functions between the **Begin Dialog...End Dialog** statements, and then display it using the **Dialog** statement (or function).

SBL records all selections the user makes in the dialog box. You can retrieve the selections when the dialog box is closed. In addition, your program may include a dialog function which, through the use of dialog functions and statements prefixed with “Dlg”, such as **DlgVisible**, can customize the behavior of the dialog box (such as validating fields as they are entered).

SBL also includes statements and functions to display **message boxes**, that notify the user of an event; **password boxes**, where the user's keystrokes are not echoed on the screen; and **input boxes**, that prompt for a single line of input.

Financial Functions

SBL includes a list of financial functions, for calculating such things as loan payments, internal rates of return, or future values based on a company's cash flows.

Date and Time Functions

The date and time functions have been expanded to make it easier to compare a file's date to today's date, set the current date and time, time events, and perform scheduling-type functions (such as finding the date for next Tuesday).

Object Handling

Windows includes OLE2 Object Handling, the ability to link and embed objects from one application into another. An object is the end product of a software application, such as a document from a word processing application. An offshoot of that ability is the **Object** data type which permits your SBL code to access another software application through its objects and change those objects.

Environment Control

SBL includes the ability to call another software application (**AppActivate**), and send the application keystrokes (**SendKeys**). Other environment control features include the ability to run an executable program (**Shell**), temporarily suspend processing to allow the operating system to process messages (**DoEvents**), and return values in the operating system environment table (**Environ\$**).

SBL Compared to Visual Basic

Although SBL is a subset of Microsoft's Visual Basic (VB), it does contain a few statements and functions not found in the standard version of VB, notably:

- | | |
|----------------|--------------|
| ■ \$CStrings | ■ GetField\$ |
| ■ \$Include | ■ SetField\$ |
| ■ \$NoCStrings | ■ With |
| ■ Assert | |

In addition, VB does not include the statements and functions needed to create or run dialog boxes. These features are available, however, in subsets of VB that are provided with other Microsoft products, such as Word and Excel. These versions, called Visual Basic for Applications (VBA), provide the dialog box handling statements and functions found in SBL, except for the following:

- | | |
|---------------|------------------|
| ■ ButtonGroup | ■ DropComboBox |
| ■ Caption | ■ StaticComboBox |

+ *VBA does include some dialog box statements and functions that are not included in SBL, such as `DlgFilePreview`.*

SBL Reference

This chapter lists the SBL commands and functions in alphabetical order, and indicates action, syntax, and any relevant comments.

Abs Function

Action	Returns the absolute value of a number.
Syntax	Abs (<i>number</i>) where <i>number</i> is <i>any valid numeric expression</i> .
Comments	The data type of the return value matches the type of the <i>number</i> . If <i>number</i> is a Variant string (vartype 8), the return value will be converted to vartype 5 (Double). If the absolute value evaluates to vartype 0 (Empty), the return value will be vartype 3 (Long).
Example	<p>This example finds the difference between two variables, oldacct and newacct.</p> <pre> Sub main Dim oldacct, newacct, count oldacct=InputBox("Enter the oldacct number") newacct=InputBox("Enter the newacct number") count=Abs(oldacct-newacct) MsgBox "The absolute value is: " &count End Sub </pre>
See Also	Exp , Fix , Int , Log , Rnd , Sgn , Sqr

AppActivate Statement

Action	Activates an application window.
Syntax	AppActivate <i>title</i> where <i>title</i> is a string expression for the title-bar name of the application window to activate.
Comments	<i>Title</i> must match the name of the window character for character, but comparison is not case sensitive, e.g., “File Manager” is the same as “file manager” or “FILE MANAGER”. If there is more than one window with a name matching <i>title</i> , a window is chosen at random.

AppActivate changes the focus to the specified window but does not change whether the window is minimized or maximized. Use **AppActivate** with the **SendKeys** statement to send keys to another application.

Example This example opens the Windows bitmap file ARCADE.BMP in Paintbrush. (Paintbrush must already be open before running this example. It must also not be minimized.)

```
Sub main
  MsgBox "Opening C:\WINDOWS\ARCADE.BMP in Paintbrush."
  AppActivate "Paintbrush - (Untitled)"
  SendKeys "%FOC:\WINDOWS\ARCADE.BMP{Enter}",1
  MsgBox "File opened."
End Sub
```

See Also [SendKeys](#), [Shell](#)

Asc Function

Action Returns an integer corresponding to the ANSI code of the first character in the specified string

Syntax **Asc**(*string\$*) where *string\$* is a string expression of one or more characters.

Comments To change an ANSI code to string characters, use **Chr**.

Example This example asks the user for a letter and returns its ASCII value.

```
Sub main
  Dim userchar
  userchar=InputBox("Type a letter:")
  MsgBox "The ASC value for " & userchar & " is: " & Asc(userchar)
End Sub
```

See Also [Chr](#)

Assert Statement [SBL Extension]**

Action Triggers a run-time error if the condition specified is FALSE.

Syntax **Assert** *condition* where *condition* is a numeric or string expression that can evaluate to TRUE or FALSE.

Comments The **Assert** statement should be used by SBL clients to handle an application specific error. An assertion error cannot be trapped by the **On Error** statement.

Use the **Assert** statement to ensure that a procedure is performing in the expected manner.

**SBL offers a number of extensions that are not included in Visual Basic.

Atn Function

Action	Returns the angle (in radians) for the arc tangent of the specified number.
Syntax	Atn (<i>number</i>) where number is any valid numeric expression.
Comments	<p>The Atn function assumes <i>number</i> is the ratio of two sides of a right triangle: the side opposite the angle to find and the side adjacent to the angle. The function returns a single-precision value for a ratio expressed as an integer, a currency, or a single-precision numeric expression. The return value is a double-precision value for a long, Variant or double-precision numeric expression.</p> <p>To convert radians to degrees, multiply by (180/PI). The value of PI is approximately 3.14159.</p>
Example	<p>This example finds the roof angle necessary for a house with an attic ceiling of 8 feet (at the roof peak) and a 16 foot span from the outside wall to the center of the house. The Atn function returns the angle in radians; it is multiplied by 180/PI to convert it to degrees.</p> <pre> Sub main Dim height, span, angle, PI PI=3.14159 height=8 span=16 angle=Atn(height/span)*(180/PI) MsgBox "The angle is " & Format(angle, "##.###") & " degrees" End Sub </pre>

See Also [Cos, Sin, Tan, Derived Trigonometric Functions](#)

Beep Statement

Action	Produces a tone through the computer speaker.
Syntax	Beep
Comments	The frequency and duration of the tone depends on the hardware.
Example	<p>This example beeps and displays a message in a box if the variable <i>balance</i> is less than 0. (If you have a set of speakers hooked up to your computer, you may need to turn them on to hear the beep.)</p> <pre> Sub main Dim expenses, balance, msgtext balance=InputBox("Enter your account balance") expenses=1000 balance=balance-expenses </pre>

```

If balance<0 then
  Beep
  MsgBox "Im sorry, your account is overdrawn."
Else
  MsgBox "Your balance minus expenses is: " &balance
End If
End Sub

```

See Also [InputBox](#), [MsgBox Statement](#), [Print](#)

Begin Dialog ... End Dialog Statement

Action Produces a tone through the computer speaker.

Action Begins and ends a dialog-box declaration.

Syntax **Begin Dialog** *dialogName* [*x*, *y*,] *dx*, *dy* [, *caption\$*] [, *.dialogfunction*]

' dialog box definition statements

End Dialog

where	is
<i>dialogName</i>	The record name for the dialog box definition.
<i>x</i> , <i>y</i>	The coordinates for the upper left corner of the dialog box.
<i>dx</i> , <i>dy</i>	The width and height of the dialog box (relative to <i>x</i> and <i>y</i>).
<i>caption\$</i>	The title for the dialog box.
<i>.dialogfunction</i>	A Basic function to process user actions in the dialog box.

Comments To display the dialog box, you create a dialog record variable with the **Dim** statement, and then display the dialog box using the **Dialog function** or **Dialog statement** with the variable name as its argument. In the **Dim** statement, this variable is defined **As** *dialogName*.

The *x* and *y* coordinates are relative to the upper left corner of the client area of the parent window. The *x* argument is measured in units that are 1/4 the average width of the system font. The *y* argument is measured in units 1/8 the height of the system font. For example, to position a dialog box 20 characters in, and 15 characters down from the upper left hand corner, enter 80, 120 as the *x*, *y* coordinates. If these arguments are omitted, the dialog box is centered in the client area of the parent window.

The *dx* argument is measured in 1/4 system-font character-width units. The *dy* argument is measured in 1/8 system-font character-width units. For example, to create a dialog box 80 characters wide, and 15 characters in height, enter 320, 120 for the *dx*, *dy* coordinates.

If the *caption\$* argument is omitted, a standard default caption is used.

The optional *.dialogfunction* function must be defined (using the **Function** statement) or declared (using **Dim**) before being used in the **Begin Dialog** statement. Define the *dialogfunction* with the following three arguments:

```
Function dialogfunction% ( id$, action%, suppvalue& )
                        ' function body
```

End Function

<i>id\$</i>	The text string that identifies the dialog control that triggered the call to the dialog function (usually because the user changed this control).
<i>action%</i>	An integer from 1 to 5 identifying the reason why the dialog function was called.
<i>suppvalue&</i>	Gives more specific information about why the dialog function was called.

As with any Basic function, these arguments may have different names. The arguments of the dialog function may also be Variants.

In most cases, the return value of *dialogfunction* is ignored. The exceptions are a return value of 2 or 5 for *action%*. If the user clicks the OK button, Cancel button, or a command button (as indicated by an *action%* return value of 2 and the corresponding *id\$* for the button clicked), and the dialog function returns a non-zero value, the dialog box will *not* be closed.

Unless the **Begin Dialog** statement is followed by at least one other dialog-box definition statement and the **End Dialog** statement, an error will result. The definition statements must include an **OkButton**, **CancelButton** or **Button** statement. If this statement is left out, there will be no way to close the dialog box, and the procedure will be unable to continue executing.

Id\$ is the same value for the dialog control that you use in the definition of that control. For example, the *id\$* value for a text box is Text1 if it is defined this way:

```
Textbox 271 , 78, 33, 18, .Text1
```

The following table summarizes the possible *action%* values and their meanings.

action%	Meaning
1	Dialog box initialization. This value is passed before the dialog box becomes visible.
2	Command button selected or dialog box control changed (except typing in a text box or combo box).
3	Change in a text box or combo box. This value is passed when the control loses the input focus: the user presses the TAB key or clicks another control.
4	Change of control focus. <i>Id\$</i> is the id of the dialog control gaining focus. <i>Suppvalue&</i> contains the numeric id of the control losing focus. A dialog function cannot display a message box or dialog box in response to an action value 4.
5	An idle state. As soon as the dialog box is initialized (<i>action%</i> = 1), the dialog function will be continuously called with <i>action%</i> = 5 if no other action occurs. If <i>dialog function</i> wants to receive this message continuously while the dialog box is idle, return a non-zero value. If 0 (zero) is returned, <i>action%</i> = 5 will be passed only while the user is moving the mouse. For this action, <i>Id\$</i> is equal to empty string (""), and <i>suppvalue&</i> is equal to the number of times action 5 was passed before.

If the user clicks a command button or changes a dialog box control, *action%* returns 2 or 3 and *suppvalue&* identifies the control affected. The value returned depends on the type of control or button the user changed or clicked. The following table summarizes the possible values for *suppvalue&*.

Control	<i>suppvalue&</i>
List box	Number of the item selected, 0-based.
Checkbox	1 if selected, 0 if cleared, -1 if filled with gray.
Option button	Number of the option button in the option group, 0-based.
Text box	Number of characters in the text box.
Combo box	The number of the item selected (0-based) for action 2, the number of characters in its text box for action 3.
OK button	1
Cancel button	2

Example

This example defines and displays a dialog box with each type of item in it: list box, combo box, buttons, etc.

```
Sub main
    Dim ComboBox1() as String
    Dim ListBox1() as String
    Dim DropListBox1() as String
    ReDim ListBox1(0)
```

```
ReDim ComboBox1(0)
ReDim DropListBox1(3)
ListBox1(0)="C:\"
ComboBox1(0)=Dir("C:\*.**")
For x=0 to 2
  DropListBox1(x)=Chr(65+x) & ":"
Next x
Begin Dialog UserDialog 274, 171, "SBL Dialog Box"
  ButtonGroup .ButtonGroup1
  Text 9, 3, 69, 13, "Filename:", .Text1
  DropComboBox 9, 14, 81, 119, ComboBox1(), .ComboBox1
  Text 106, 2, 34, 9, "Directory:", .Text2
  ListBox 106, 12, 83, 39, ListBox1(), .ListBox2
  Text 106, 52, 42, 8, "Drive:", .Text3
  DropListBox 106, 64, 95, 44, DropListBox1(), .DropListBox1
  CheckBox 9, 142, 62, 14, "List .TXT files", .CheckBox1
  GroupBox 106, 111, 97, 57, "File Range"
  OptionGroup .OptionGroup2
    OptionButton 117, 119, 46, 12, "All pages", .OptionButton3
    OptionButton 117, 135, 67, 8, "Range of pages", .OptionButton4
  Text 123, 146, 20, 10, "From:", .Text6
  Text 161, 146, 14, 9, "To:", .Text7
  TextBox 177, 146, 13, 12, .TextBox4
  TextBox 145, 146, 12, 11, .TextBox5
  OkButton 213, 6, 54, 14
  CancelButton 214, 26, 54, 14
  PushButton 213, 52, 54, 14, "Help", .Push1
End Dialog
Dim mydialog as UserDialog
On Error Resume Next
Dialog mydialog
If Err=102 then
  MsgBox "Dialog box canceled."
End If
End Sub
```

See Also [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Button Statement

Action	Defines a custom pushbutton.
Syntax A	Button <i>x, y, dx, dy, text\$</i> [, <i>.id</i>]
Syntax B	PushButton <i>x, y, dx, dy, text\$</i> [, <i>.id</i>]
where	is
<i>x,y</i>	The position of the button relative to the upper left corner of the dialog box.
<i>dx,dy</i>	The width and height of the button.

<i>text\$</i>	The name for the pushbutton. If the width of this string is greater than <i>dx</i> , trailing characters are truncated.
<i>.id</i>	An optional identifier used by the dialog statements that act on this control.

Comments A *dy* value of 14 typically accommodates text in the system font.

Use this statement to create buttons other than OK and Cancel. Use this statement in conjunction with the **ButtonGroup** statement. The two forms of the statement (**Button** and **PushButton**) are equivalent.

Use the **Button** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a combination list box and three buttons.

```
Sub main
  Dim fchoices as String
  fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
  Begin Dialog UserDialog 185, 94, "SBL Dialog Box"
    Text 9, 5, 69, 10, "Filename:", .Text1
    DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1
    ButtonGroup .ButtonGroup1
    OkButton 113, 14, 54, 13
    CancelButton 113, 33, 54, 13
    Button 113, 57, 54, 13, "Help", .Push1
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also [Begin Dialog...End Dialog Statement](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [DropComboBox](#), [DropListBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

ButtonGroup Statement

Action Begins the definition of a group of custom buttons for a dialog box.

Syntax **ButtonGroup** *.field* where *.field* is the field to contain the user's custom button selection.

Comments If **ButtonGroup** is used, it must appear before any **Button** statement which creates a custom button (one other than OK or Cancel.) Only one **ButtonGroup** statement is allowed within a dialog box definition.

Use the **ButtonGroup** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a combination list box and three buttons.

```
Sub main
  Dim fchoices as String
  fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
  Begin Dialog UserDialog 185, 94, "SBL Dialog Box"
    Text 9, 5, 69, 10, "Filename:", .Text1
    DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1
    'The next four lines create three buttons
    ButtonGroup .ButtonGroup1
    OkButton 113, 14, 54, 13
    CancelButton 113, 33, 54, 13
    PushButton 113, 57, 54, 13, "Help", .Push1
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also [Begin Dialog...End Dialog Statement](#), [Button](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [DropComboBox](#), [DropListBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Call Statement

Action	Transfers control to a subprogram or function						
Syntax A	Call <i>subprogram name</i> [(<i>argumentlist</i>)]						
Syntax B	<i>subprogram name</i> <i>argumentlist</i>						
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>subprogram-name</i></td><td>the name of the subroutine or function to call.</td></tr> <tr> <td><i>argumentlist</i></td><td>the arguments for the subroutine or function (if any).</td></tr> </table>	where	is	<i>subprogram-name</i>	the name of the subroutine or function to call.	<i>argumentlist</i>	the arguments for the subroutine or function (if any).
where	is						
<i>subprogram-name</i>	the name of the subroutine or function to call.						
<i>argumentlist</i>	the arguments for the subroutine or function (if any).						
Comments	Use the Call statement to call a subprogram or function written in Basic or to call C procedures in a DLL. These C procedures must be described in a Declare statement or be implicit in the application.						

Arguments are passed by reference to procedures written in Basic. If you pass a variable to a procedure which modifies its corresponding formal parameter, and you do not wish to have your variable modified, enclose the variable in parentheses in the Call statement. This will tell SBL to pass a copy of the variable. Note that this will be less efficient, and should not be done unless necessary.

When a variable is passed to a procedure which expects its argument by reference, the variable must match the exact type of the formal parameter of the function. (This restriction does not apply to expressions or Variants.)

When calling an external DLL procedure, arguments can be passed by value rather than by reference. This is specified either in the **Declare** statement, the **Call** itself, or both, using the **ByVal** keyword. If **ByVal** is specified in the declaration, then the **ByVal** keyword is optional in the call. If present, it must precede the value. If **ByVal** was not specified in the declaration, it is illegal in the call unless the data type specified in the declaration was **Any**.

Example

This example calls a subprogram named CREATEFILE to open a file, write the numbers 1 to 10 in it and leave it open. The calling procedure then checks the file's mode. If the mode is 1 (open for Input) or 2 (open for Output), the procedure closes the file.

```
Declare Sub createfile()  
Sub main  
  Dim filemode as Integer  
  Dim attrib as Integer  
  Call createfile  
  attrib=1  
  filemode=FileAttr(1,attrib)  
  If filemode=1 or 2 then  
    MsgBox "File was left open. Closing now."  
    Close #1  
  End If  
  Kill "C:\TEMP001"  
End Sub  
Sub createfile()  
  Rem Put the numbers 1-10 into a file  
  Dim x as Integer  
  Open "C:\TEMP001" for Output as #1  
  For x=1 to 10  
    Write #1, x  
  Next x  
End Sub
```

See Also [Declare](#)

CancelButton Statement

Action	Sets the position and size of a Cancel button in a dialog box.
Syntax	<p>CancelButton <i>x, y, dx, dy</i> [<i>, .id</i>]</p> <p>where is</p> <p><i>x,y</i> the position of the Cancel button relative to the upper left corner of the dialog box.</p> <p><i>dx,dy</i> the width and height of the button.</p> <p><i>.id</i> an optional identifier for the button.</p>
Comments	<p>A <i>dy</i> value of 14 can usually accommodate text in the system font.</p> <p><i>.Id</i> is used by the dialog statements that act on this control.</p> <p>If you use the Dialog statement to display the dialog box and the user clicks Cancel, the box is removed from the screen and an Error 102 is triggered. If you use the Dialog function to display the dialog box, the function will return 0 and no error occurs.</p> <p>Use the CancelButton statement only between a Begin Dialog and an End Dialog statement.</p>
Example	<p>This example defines a dialog box with a combination list box and three buttons.</p> <pre> Sub main Dim fchoices as String fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3" Begin Dialog UserDialog 185, 94, "SBL Dialog Box" Text 9, 5, 69, 10, "Filename:", .Text1 DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1 ButtonGroup .ButtonGroup1 OkButton 113, 14, 54, 13 CancelButton 113, 33, 54, 13 PushButton 113, 57, 54, 13, "Help", .Push1 End Dialog Dim mydialog as UserDialog On Error Resume Next Dialog mydialog If Err=102 then MsgBox "Dialog box canceled." End If End Sub </pre>
See Also	Begin Dialog...End Dialog Statement , Button , ButtonGroup , Caption , CheckBox , ComboBox , DropComboBox , DropListBox , GroupBox , ListBox , OKButton , OptionButton , OptionGroup , Picture , StaticComboBox , Text , TextBox

Caption Statement

Action	Defines the title of a dialog box.
Syntax	Caption <i>text\$</i> where <i>text\$</i> is a string expression containing the title of the dialog box.
Comments	Use the Caption statement only between a Begin Dialog and an End Dialog statement. If no Caption statement is specified for the dialog box, a default caption is used.
Example	<p>This example defines a dialog box with a combination list box and three buttons. The Caption statement changes the dialog box title to “Example -Caption Statement”.</p> <pre> Sub main Dim fchoices as String fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3" Begin Dialog UserDialog 185, 94 Caption "Example-Caption Statement" Text 9, 5, 69, 10, "Filename:", .Text1 DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1 ButtonGroup .ButtonGroup1 OkButton 113, 14, 54, 13 CancelButton 113, 33, 54, 13 PushButton 113, 57, 54, 13, "Help", .Push1 End Dialog Dim mydialog as UserDialog On Error Resume Next Dialog mydialog If Err=102 then MsgBox "Dialog box canceled." End If End Sub </pre>
See Also	Begin Dialog...End Dialog Statement , Button , CancelButton , ButtonGroup , CheckBox , ComboBox , DropComboBox , DropListBox , GroupBox , ListBox , OKButton , OptionButton , OptionGroup , Picture , StaticComboBox , Text , TextBox

CCur Function

Action	Converts an expression to the data type Currency .
Syntax	CCur (<i>expression</i>) where <i>expression</i> is any expression that evaluates to a number.
Comments	CCur accepts any type of <i>expression</i> . Numbers that do not fit in the Currency data type result in an “Overflow” error. Strings that cannot be converted result in a “Type Mismatch” error. Variants containing null result in an “Illegal Use of Null” error.

Example This example converts a yearly payment on a loan to a currency value with four decimal places. A subsequent Format statement formats the value to two decimal places before displaying it in a message box.

```
Sub main
Dim aprate, totalpay, loanpv
Dim loanfv, due, monthlypay
Dim yearlypay, msgtext
loanpv=InputBox("Enter the loan amount: ")
aprate=InputBox("Enter the annual percentage rate: ")
If aprate >1 then
    aprate=aprate/100
End If
aprate=aprate/12
totalpay=InputBox("Enter the total number of pay periods: ")
loanfv=0
Rem Assume payments are made at end of month
due=0
monthlypay=Pmt(aprate,totalpay,-loanpv,loanfv,due)
yearlypay=CCur(monthlypay*12)
msgtext= "The yearly payment is: " & Format(yearlypay, "Currency")
MsgBox msgtext
End Sub
```

See Also [Cdbl](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#)

CDbl Function

- Action** Converts an expression to the data type **Double**.
- Syntax** **CDbl**(expression) where *expression* is any expression that evaluates to a number.
- Comments** **CDbl** accepts any type of *expression*. Strings that cannot be converted to a double-precision floating point result in a “Type Mismatch” error. Variants containing null result in an “Illegal Use of Null” error.
- Example** This example calculates the square root of 2 as a double-precision floating point value and displays it in scientific notation.

```
Sub main
Dim value
Dim msgtext
value=CDbl(Sqr(2))
msgtext= "The square root of 2 is: " & Value
MsgBox msgtext
End Sub
```

See Also [Ccur](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#)

ChDir Statement

Action	Changes the default directory for the specified drive.
Syntax	ChDir <i>path\$</i> where <i>path\$</i> is a string expression identifying the new default directory.
Comments	<p>The syntax for <i>path\$</i> is:</p> <p style="text-align: center;">[<i>drive:</i>] [\] <i>directory</i> [\ <i>directory</i>]</p> <p>If the drive argument is omitted, ChDir changes the default directory on the current drive. The ChDir statement does not change the default drive. To change the default drive, use ChDrive.</p>
Example	<p>This example changes the current directory to C:\WINDOWS, if it is not already the default.</p> <pre> Sub main Dim newdir as String newdir="c:\windows" If CurDir <> newdir then ChDir newdir End If MsgBox "The default directory is now: " & newdir End Sub </pre>
See Also	ChDrive , CurDir , Dir , MkDir , Rmdir

ChDrive Statement

Action	Changes the default drive.
Syntax	ChDrive <i>drive\$</i> where <i>drive\$</i> is a string expression designating the new default drive.
Comments	<p>This drive must exist and must be within the range specified by the LASTDRIVE statement in the CONFIG.SYS file. If a null argument (" ") is supplied, the default drive remains the same. If the <i>drive\$</i> argument is a string, ChDrive uses the first letter only. If the argument is omitted, an error message is produced. To change the current directory on a drive, use ChDir.</p>
Example	<p>This example changes the default drive to A:.</p> <pre> Sub main Dim newdrive as String newdrive="A:" If Left(CurDir,2) <> newdrive then ChDrive newdrive End If MsgBox "The default drive is now " & newdrive End Sub </pre>
See Also	ChDir , CurDir , Dir , MkDir , Rmdir

CheckBox Statement

Action Creates a checkbox in a dialog box.

Syntax **CheckBox** *x, y, dx, dy, text\$, .field*

where	is
<i>x,y</i>	the upper left corner coordinates of the checkbox, relative to the upper left corner of the dialog box.
<i>dx</i>	the sum of the widths of the checkbox and <i>text\$</i> .
<i>dy</i>	the height of <i>text\$</i> .
<i>text\$</i>	the title shown to the right of the checkbox .
<i>.field</i>	the name of the dialog-record field that will hold the current checkbox setting (0=unchecked, -1=grey, 1=checked).

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-height units. (See **Begin Dialog** for more information.)

Because proportional spacing is used, the *dx* argument width will vary with the characters used. To approximate the width, multiply the number of characters in the *text\$* field (including blanks and punctuation) by 4 and add 12 for the checkbox.

A *dy* value of 12 is standard, and should cover typical default fonts. If larger fonts are used, the value should be increased. As the *dy* number grows, the checkbox and the accompanying text will move down within the dialog box.

If the width of the *text\$* field is greater than *dx*, trailing characters will be truncated. If you wish to include underlined characters so that the checkbox selection can be made from the keyboard, precede the character to be underlined with an ampersand (&).

SBL treats any other value of *.field* the same as a 1. The *.field* argument is also used by the dialog statements that act on this control.

Use the **CheckBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a combination list box, a checkbox , and three buttons.

```
Sub main
  Dim ComboBox1() as String
  ReDim ComboBox1(0)
  ComboBox1(0)=Dir("C:\*.txt")
  Begin Dialog UserDialog 166, 76, "SBL Dialog Box"
    Text 9, 3, 69, 13, "Filename:", .Text1
    DropComboBox 9, 14, 81, 119, ComboBox1(), .ComboBox1
    CheckBox 10, 39, 62, 14, "List .TXT files", .CheckBox1
```

```

        OkButton 101, 6, 54, 14
        CancelButton 101, 26, 54, 14
        PushButton 101, 52, 54, 14, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub

```

See Also [Begin Dialog...End Dialog Statement](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [ComboBox](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Chr Function

Action Returns the one-character string corresponding to an ANSI code.

Syntax Chr[\$](*charcode*) where *charcode* is an integer between 0 and 255.

Comments The dollar sign, “\$”, in the function name is optional. If specified, the return type is String. If omitted, the function will return a Variant of vartype 8 (string).

Example This example displays the character equivalent for an ASCII code between 65 and 122 typed by the user.

```

Sub main
    Dim numb as Integer
    Dim msgtext
    Dim out
    out=0
    Do Until out
        numb=InputBox("Type a number between 65 and 122:")
        If Chr$(numb)>="A" AND Chr$(numb)<="Z" OR Chr$(numb)>="a" AND _
            Chr$(numb)<="z" then
            msgtext="The letter for the number " & numb & " is: " & Chr$(numb)
            out=1
        ElseIf numb=0 then
            Exit Sub
        Else
            Beep
            msgtext="Does not convert to a character; try again."
        End If
        MsgBox msgtext
    Loop
End Sub

```

See Also [Asc](#), [Ccur](#), [Cdbl](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#), [Format](#), [Val](#)

CInt Function

Action	Converts an expression to the data type Integer by rounding.
Syntax	CInt (<i>expression</i>) where <i>expression</i> is any expression that can evaluate to a number.
Comments	<p>After rounding, the resulting number must be within the range of 32767 to 32767, or an error occurs.</p> <p>Strings that cannot be converted to an integer result in a “Type Mismatch” error. Variants containing null result in an “Illegal Use of Null” error.</p>
Example	<p>This example calculates the average of ten golf scores.</p> <pre> Sub main Dim score As Integer Dim x, sum Dim msgtext Let sum=0 For x=1 to 10 score=InputBox("Enter golf score #"&x &":") sum=sum+score Next x msgtext="Your average is: " & Format(CInt(sum/(x-1)),"General Number") MsgBox msgtext End Sub </pre>
See Also	Ccur , Cdbl , Clng , Csng , Cstr , Cvar , CVDate

CLng Function

Action	Converts an expression to the data type Long by rounding.
Syntax	CLng (<i>expression</i>) where <i>expression</i> is any expression that can evaluate to a number.
Comments	<p>After rounding, the resulting number must be within the range of 2,147,483,648 to 2,147,483,647, or an error occurs.</p> <p>Strings that cannot be converted to a long result in a “Type Mismatch” error. Variants containing null result in an “Illegal Use of Null” error.</p>

Example This example divides the US national debt by the number of people in the country to find the amount of money each person would have to pay to wipe it out. This figure is converted to a Long integer and formatted as Currency.

```
Sub main
  Dim debt As Single
  Dim msgtext
  Const Populace = 250000000
  debt=InputBox("Enter the current US national debt:")
  msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")
  MsgBox msgtext
End Sub
```

See Also [Ccur](#), [Cdbl](#), [Cint](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#)

Close Statement

Action Closes a file, concluding input/output to that file.

Syntax **Close** [[#] *filename%* [, [] *filename%* ...]] where *filename%* is an integer expression identifying the file to close.

Comments *Filename%* is the number assigned to the file in the **Open** statement and may be preceded by a pound sign (#). If this argument is omitted, all open files are closed. Once a **Close** statement is executed, the association of a file with *filename%* is ended, and the file can be reopened with the same or a different file number.

When the **Close** statement is used, the final output buffer is written to the operating system buffer for that file. **Close** frees all buffer space associated with the closed file. Use the **Reset** statement so that the operating system will flush its buffers to disk.

Example This example opens a file for Random access, gets the contents of one variable, and closes the file again. The subprogram, **CREATEFILE**, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Sub main
  Dim acctno as String*3
  Dim recno as Long
  Dim msgtext as String
  Call createfile
  recno=1
  newline=Chr(10)
  Open "C:\TEMP001" For Random As #1 Len=3
  msgtext="The account numbers are:" & newline & newline
  Do Until recno=11
    Get #1,recno,acctno
    msgtext=msgtext & acctno
```

```

        recno=recno+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub

```

See Also [Open, Reset, Stop](#)

ComboBox Statement

Action	Creates a combination text box and list box in a dialog box.												
Syntax A	ComboBox <i>x, y, dx, dy, text\$, .field</i>												
Syntax B	ComboBox <i>x, y, dx, dy, stringarray\$, .field</i>												
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>x,y</i></td><td>the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.</td></tr> <tr> <td><i>dx,dy</i></td><td>the width and height of the combo box in which the user enters or selects text.</td></tr> <tr> <td><i>text\$</i></td><td>A string containing the selections for the combo box.</td></tr> <tr> <td><i>stringarray\$</i></td><td>An array of dynamic strings for the selections in the combo box.</td></tr> <tr> <td><i>.field</i></td><td>The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.</td></tr> </table>	where	is	<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.	<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.	<i>text\$</i>	A string containing the selections for the combo box.	<i>stringarray\$</i>	An array of dynamic strings for the selections in the combo box.	<i>.field</i>	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.
where	is												
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.												
<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.												
<i>text\$</i>	A string containing the selections for the combo box.												
<i>stringarray\$</i>	An array of dynamic strings for the selections in the combo box.												
<i>.field</i>	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.												
Comments	<p>The <i>x</i> argument is measured in 1/4 system-font character-width units. The <i>y</i> argument is measured in 1/8 system-font character-width units. (See Begin Dialog for more information.)</p> <p>The <i>text\$</i> argument must be defined, using a Dim Statement, before the Begin Dialog statement is executed. The arguments in the <i>text\$</i> string are entered as shown in the following example:</p> <pre>dimname = "listchoice"+Chr\$(9)+"listchoice"+Chr\$(9)+"listchoice"...</pre>												

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *.field* argument is also used by the dialog statements that act on this control.

Use the **ComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example

This example defines a dialog box with a combination list and text box and three buttons.

```
Sub main
  Dim ComboBox1() as String
  ReDim ComboBox1(0)
  ComboBox1(0)=Dir("C:\*.*)"
  Begin Dialog UserDialog 166, 142, "SBL Dialog Box"
    Text 9, 3, 69, 13, "Filename:", .Text1
    ComboBox 9, 14, 81, 119, ComboBox1(), .ComboBox1
    OkButton 101, 6, 54, 14
    CancelButton 101, 26, 54, 14
    PushButton 101, 52, 54, 14, "Help", .Push1
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also

[Begin Dialog...End Dialog Statement](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [DropComboBox](#), [DropListBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Command Function

Action Returns the command line specified when the MAIN subprogram was invoked.

Syntax **Command**[\$]

Comments After the MAIN subprogram returns, further calls to the Command function will yield an empty string. This function may not be supported in some implementations of SBL.

The dollar sign, "\$", in the function name is optional. If specified, the return type is String. If omitted, the function returns a **Variant** of vartype 8 (string).

Example

This example opens the file entered by the user on the command line.

```
Sub main
  Dim filename as String
  Dim cmdline as String
  Dim cmdlength as Integer
```



```
Dim position as Integer
cmdline=Command
If cmdline="" then
    MsgBox "No command line information."
    Exit Sub
End If
cmdlength=Len(cmdline)
position=InStr(cmdline,Chr(32))
filename=Mid(cmdline,position+1,cmdlength-position)
On Error Resume Next
Open filename for Input as #1
If Err<>0 then
    MsgBox "Error loading file."
    Exit Sub
End If

MsgBox "File " & filename & " opened."
Close #1
MsgBox "File " & filename & " closed."
End Sub
```

See Also [AppActivate](#), [DoEvents](#), [Environ](#), [SendKeys](#), [Shell](#)

Const Statement

Action	Declares symbolic constants for use in a Basic program.
Syntax	<div><div>[Global] Const <i>constantName</i> [As <i>type</i>]= <i>expression</i> [,<i>constantName</i> = <i>expression</i>] ...</div><div><div>where</div><div><i>constantName</i></div><div><i>type</i></div><div><i>expression</i></div></div><div><div>is</div><div>the variable name to contain a constant value.</div><div>the data type of the constant (Number or String).</div><div>any expression that evaluates to a constant number.</div></div></div>
Comments	<p>Instead of using the As clause, the type of the constant may be specified by using a type character as a suffix (for numbers, \$ for strings) to the <i>constantName</i>. If no type character is specified, the type of the <i>constantName</i> is derived from the type of the expression.</p> <p>If Global is specified, the constant is validated at module load time. If the constant has already been added to the run-time global area, the constant's type and value are compared to the previous definition, and the load fails if a mismatch is found. This is useful as a mechanism for detecting version mismatches between modules.</p>
Example	This example divides the US national debt by the number of people in the country to find the amount of money each person would have to pay to wipe it out. This figure is converted to a Long integer and formatted as Currency.

```

Sub main
    Dim debt As Single
    Dim msgtext
    Const Populace=250000000
    debt=InputBox("Enter the current US national debt:")
    msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")
    MsgBox msgtext
End Sub

```

See Also [Declare, Deftype, Dim, Let, Type](#)

Cos Function

Action Returns the cosine of an angle.

Syntax **Cos**(*number*) where *number* is an angle in radians.

Comments The return value will be between -1 and 1. The return value is a single-precision number if the angle has a data type **Integer**, **Currency**, or is a single-precision value. The return value will be a double precision value if the angle has a data type **Long**, **Variant** or is a double-precision value.

The angle can be either positive or negative. To convert degrees to radians, multiply by (PI/180). The value of PI is approximately 3.14159.

Example This example finds the length of a roof, given its pitch and the distance of the house from its center to the outside wall.

```

Sub main
    Dim bwidth, roof, pitch
    Dim msgtext
    Const PI=3.14159
    Const conversion=PI/180
    pitch=InputBox("Enter roof pitch in degrees")
    pitch=Cos(pitch*conversion)
    bwidth=InputBox("Enter 1/2 of house width in feet")
    roof=bwidth/pitch
    msgtext="The length of the roof is " & Format(roof, "##.##") & " feet."
    MsgBox msgtext
End Sub

```

See Also [Atn, Sin, Tan, Derived Trigonometric Functions](#)

CreateObject Function

Action	Creates a new OLE2 automation object.
Syntax	CreateObject (<i>class</i>) where <i>class</i> is the name of the application, a period, and the name of the object to be used.
Comments	To create an object, you first must declare an object variable, using Dim , and then Set the variable equal to the new object, as follows:

```
Dim OLE2 As Object
Set OLE2 = CreateObject("spoly.cpoly")
```

To refer to a method or property of the newly created object, use the syntax *objectvar.property* or *objectvar.method*, as follows: *OLE2.reset*

Refer to the documentation provided with your OLE2 automation server application for correct application and object names.

Example	This example uses the CreateObject function to open the software product VISIO (if it is not already open).
----------------	---

```
Sub main
  Dim visio as Object
  Dim doc as Object
  Dim i as Integer, doccount as Integer
  "Initialize Visio
  Set visio = GetObject(,"visio.application")      ' find Visio
  If (visio Is Nothing) then
    Set visio = CreateObject("visio.application") ' find Visio

    If (visio Is Nothing) then
      MsgBox "Couldn't find Visio!"
    Exit Sub
  End If
  MsgBox "Visio is open."
End If
End Sub
```

See Also [GetObject](#), [Is](#), [Me](#), [New](#), [Nothing](#), [Object Class](#), [Typeof](#)

CSng Function

Action	Converts an expression to the data type Single .
Syntax	CSng (<i>expression</i>) where <i>expression</i> is any expression that can evaluate to a number.

Comments	<p>The expression must have a value within the range allowed for the Single data type, or an error occurs.</p> <p>Strings that cannot be converted to an integer result in a “Type Mismatch” error. Variants containing null result in an “Illegal Use of Null” error.</p>
Example	<p>This example calculates the factorial of a number. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.</p> <pre> Sub main Dim number as Integer Dim factorial as Double Dim msgtext number=InputBox("Enter an integer between 1 and 170:") If number<=0 then Exit Sub End If factorial=1 For x=number to 2 step -1 factorial=factorial*x Next x Rem If number <=35, then its factorial is small enough to be stored Rem as a single-precision number If number<35 then factorial=CStr(factorial) End If msgtext="The factorial of " & number & " is: " & factorial MsgBox msgtext End Sub </pre>

See Also [Ccur](#), [Cdbl](#), [Cint](#), [Clng](#), [Cstr](#), [Cvar](#), [CVDate](#)

CStr Function

Action	Converts an expression to the data type String .														
Syntax	CStr (<i>expression</i>) where <i>expression</i> is any expression that can evaluate to a number.														
Comments	<p>The CStr statement accepts any type of <i>expression</i>:</p> <table> <tr> <td><i>expression</i> is</td><td>CStr returns:</td></tr> <tr> <td><i>Boolean</i></td><td>a String containing “True” or “False”.</td></tr> <tr> <td><i>Date</i></td><td>a String containing a date.</td></tr> <tr> <td><i>Empty</i></td><td>a zero-length String ("").</td></tr> <tr> <td><i>Error</i></td><td>a String containing “Error”, followed by the error number.</td></tr> <tr> <td><i>Null</i></td><td>a run-time error.</td></tr> <tr> <td><i>Other Numeric</i></td><td>a String containing the number.</td></tr> </table>	<i>expression</i> is	CStr returns:	<i>Boolean</i>	a String containing “True” or “False”.	<i>Date</i>	a String containing a date.	<i>Empty</i>	a zero-length String ("").	<i>Error</i>	a String containing “Error”, followed by the error number.	<i>Null</i>	a run-time error.	<i>Other Numeric</i>	a String containing the number.
<i>expression</i> is	CStr returns:														
<i>Boolean</i>	a String containing “True” or “False”.														
<i>Date</i>	a String containing a date.														
<i>Empty</i>	a zero-length String ("").														
<i>Error</i>	a String containing “Error”, followed by the error number.														
<i>Null</i>	a run-time error.														
<i>Other Numeric</i>	a String containing the number.														

Example This example converts a variable from a value to a string and displays the result. Variant type 5 is Double and type 8 is String.

```
Sub main
  Dim var1
  Dim msgtext as String
  var1=InputBox("Enter a number:")
  var1=var1+10
  msgtext="Your number + 10 is: " & var1 & Chr(10)
  msgtext=msgtext & "which makes its Variant type: " & Vartype(var1)
  MsgBox msgtext
  var1=CStr(var1)
  msgtext="After conversion to a string," & Chr(10)
  msgtext=msgtext & "the Variant type is: " & Vartype(var1)
  MsgBox msgtext
End Sub
```

See Also [Asc](#), [Ccur](#), [Cdbl](#), [Chr](#), [Cint](#), [Clng](#), [Csng](#), [Cvar](#), [CVDate](#), [Format](#)

'\$CStrings Metacommand [SBL Extension]** '\$

- Action** Tells the compiler to treat a backslash character inside a string (\) as an escape character.
- Syntax** '\$CStrings [*Save* | *Restore*] where *Save* saves the current \$Cstrings setting and *Restore* restores a previously saved \$CStrings setting.
- Comments** This treatment of a backslash in a string is based on the 'C' language.
- Save** and **Restore** operate as a stack and allow the user to change the setting for a range of the program without impacting the rest of the program.

The special characters supported are following:

Newline (Linefeed)	\n
Horizontal Tab	\t
Vertical Tab	\v
Backspace	\b
Carriage Return	\r
Formfeed	\f
Backslash	\\
Single Quote	\'
Double Quote	\"
Null Character	\0

The instruction "Hello\r World" is the equivalent of "Hello" + Chr\$(13)+"World".

In addition, any character can be represented as a 3-digit octal code or a 3-digit hexadecimal code:

Octal Code	\ddd
Hexadecimal Code	\xdd

For both hexadecimal and octal, fewer than 3 characters can be used to specify the code as long as the subsequent character is not a valid (hex or octal) character.

To tell the compiler to return to the default string processing mode, where the backslash character has no special meaning, use the **\$NoCStrings** Metacommand.

**SBL offers a number of extensions that are not included in Visual Basic.

Example

This example displays two lines, the first time using the C-language characters “\n” for a carriage return and line feed.

```
Sub main
  $CStrings
  MsgBox "This is line 1\n This is line 2 (using C Strings)"
  $NoCStrings
  MsgBox "This is line 1" + Chr$(13)+Chr$(10)+"This is line 2 (using Chr)"
End Sub
```

See Also [\\$Include](#), [\\$NoCStrings](#), [Rem](#)

CurDir Function

Action Returns the default directory (and drive) for the specified drive.

Syntax **CurDir**[\$] [(*drive*\$)] where *drive*\$ is a string expression containing the drive to search.

Comments The drive must exist, and must be within the range specified in the LASTDRIVE statement of the CONFIG.SYS file. If a null argument (" ") is supplied, or if no drive\$ is indicated, the path for the default drive is returned.

The dollar sign, “\$”, in the function name is optional. If specified, the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

To change the current drive, use **ChDrive**. To change the current directory, use **ChDir**.

Example This example changes the current directory to C:\WINDOWS, if it is not already the default.

```
Sub main
  Dim newdir as String
  newdir="c:\windows"
  If CurDir <> newdir then
    ChDir newdir
  End If
  MsgBox "The default directory is now: " & newdir
End Sub
```

See Also [ChDir](#), [ChDrive](#), [Dir](#), [MkDir](#), [Rmdir](#)

CVar Function

Action Converts an expression to the data type **Variant**.

Syntax **CVar**(*expression*) where *expression* is any expression that can evaluate to a number.

Comments **CVar** accepts any type of expression.

CVar generates the same result as you would get by assigning the *expression* to a **Variant** variable.

Example This example converts a string variable to a variant variable.

```
Sub main
  Dim answer as Single
  answer=100.5
  MsgBox "'Answer' is DIM'ed as Single with the value: " & answer
  answer=CVar(answer)
  answer=Fix(answer)
  MsgBox "'Answer' is now a variant with a type of: " & VarType(answer)
End Sub
```

See Also [Ccur](#), [Cdbl](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [CVDate](#)

CVDate Function

Action Converts an expression to the data type **Variant Date**.

Syntax **CVDate**(*expression*) where *expression* is any expression that can evaluate to a number.

Comments **CVDate** accepts both string and numeric values.

The **CVDate** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999. A value of 2 represents January 1, 1900. Times are represented as fractional days.

Example This example displays the date for one week from the date entered by the user.

```
Sub main
Dim str1 as String
Dim nextweek
Dim msgtext
i: str1=InputBox$("Enter a date:")
answer=IsDate(str1)
If answer=-1 then
str1=CVDate(str1)
nextweek=DateValue(str1)+7
msgtext="One week from the date entered is:
msgtext=msgtext & "Format(nextweek,"dddddd")
MsgBox msgtext
Else
MsgBox "Invalid date or format. Try again."
Goto i
End If
End Sub
```

See Also [Asc](#), [Ccur](#), [Cdbl](#), [Chr](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [Format](#), [Val](#)

Date Function

Action Returns a string representing the current date.

Syntax **Date**[\$]

Comments The Date function returns a ten character string.

The dollar sign, "\$", in the function name is optional. If specified, the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

Example This example displays the date for one week from the today's date (the current date on the computer).

```
Sub main
Dim nextweek
nextweek=CVar(Date)+7
MsgBox "One week from today is: " & Format(nextweek,"dddddd")
End Sub
```

See Also [CVDate](#), [Date Statement](#), [Format](#), [Now](#), [Time Function](#), [Time Statement](#), [Timer](#), [TimeSerial](#)

Date Statement

Action	Sets the system date.
Syntax	Date [\$] = <i>expression</i> where <i>expression</i> is a string in one of the following forms: <i>mm-dd-yy</i> , <i>mm-dd-yyyy</i> , <i>mm/dd/yy</i> or <i>mm/dd/yyyy</i> where <i>mm</i> denotes a month (01-12), <i>dd</i> denotes a day (01-31), and <i>yy</i> or <i>yyyy</i> denotes a year (1980-2099).
Comments	<p>If the dollar sign, "\$", is omitted, <i>expression</i> can be a string containing a valid date, a Variant of vartype 7 (date), or a Variant of vartype 8 (string).</p> <p>If <i>expression</i> is not already a Variant of vartype 7 (date), Date attempts to convert it to a valid date from January 1, 1980 through December 31, 2099. Date uses the Short Date format in the International section of Windows Control Panel to recognize day, month, and year if a string contains three numbers delimited by valid date separators. In addition, Date recognizes month names in either full or abbreviated form.</p>
Example	<p>This example changes the system date to a date entered by the user.</p> <pre> Sub main Dim userdate Dim answer i: userdate=InputBox("Enter a date for the system clock:") If userdate="" then Exit Sub End If answer=IsDate(userdate) If answer=-1 then Date=userdate Else MsgBox "Invalid date or format. Try again." Goto i End If End Sub </pre>
See Also	Date Function , Time Function , Time Statement

DateSerial Function

Action	Returns a date value for year, month, and day specified.
Syntax	DateSerial (<i>year%</i> , <i>month%</i> , <i>day%</i>) where <i>year%</i> is a year between 100 and 9999, or a numeric expression, <i>month%</i> is a month between 1 and 12, or a numeric expression, and <i>day%</i> is a day between 1 and 31, or a numeric expression.
Comments	The DateSerial function returns a Variant of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999, where January 1, 1900 is 2.

A numeric expression can be used for any of the arguments to specify a relative date: a number of days, months, or years before or after a certain date.

Example This example finds the day of the week New Year's day will be for the year 2000.

```
Sub main
  Dim newyearsday
  Dim daynumber
  Dim msgtext
  Dim newday as Variant
  Const newyear=2000
  Const newmonth=1
  Let newday=1
  newyearsday=DateSerial(newyear,newmonth,newday)
  daynumber=Weekday(newyearsday)
  msgtext="New Year's day 2000 falls on a " & Format(daynumber, "dddd")
  MsgBox msgtext
End Sub
```

See Also [DateValue](#), [Day](#), [Month](#), [Now](#), [TimeSerial](#), [TimeValue](#), [Weekday](#), [Year](#)

DateValue Function

Action Returns a date value for the string specified.

Syntax **DateValue**(*date\$*) where *date\$* is a string representing a valid date.

Comments The **DateValue** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999, where January 1, 1900 is 2.

DateValue accepts several different string representations for a date. It makes use of the operating system's international settings for resolving purely numeric dates.

Example This example displays the date for one week from the date entered by the user.

```
Sub main
  Dim str1 as String
  Dim nextweek
  Dim msgtext
  i: str1=InputBox$("Enter a date:")
  answer=IsDate(str1)
  If answer=-1 then
    str1=CDate(str1)
    nextweek=DateValue(str1)+7
    msgtext="One week from your date is: " & Format(nextweek,"dddddd")
    MsgBox msgtext
  Else
    MsgBox "Invalid date or format. Try again."
    Goto i
  End If
End Sub
```

See Also [DateSerial](#), [Day](#), [Month](#), [Now](#), [TimeSerial](#), [TimeValue](#), [Weekday](#), [Year](#)

Day Function

Action	Returns the day of the month (1-31) of a date-time value.
Syntax	Day (<i>date</i>) where <i>date</i> is any expression that can evaluate to a date.
Comments	Day attempts to convert the input value of date to a date value. The return value is a Variant of vartype 2 (integer). If the value of date is null, a Variant of vartype 1 (null) is returned.
Example	This example finds the month (1-12) and day (1-31) values for this Thursday.

```

Sub main
    Dim x, today, msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " & Month(Today+x) & "/" & Day(Today+x)
    MsgBox msgtext
End Sub

```

See Also [Date Function](#), [DateStatement](#), [Hour](#), [Minute](#), [Month](#), [Now](#), [Second](#), [Weekday](#), [Year](#)

Declare Statement

Action	Declares a procedure in a module or dynamic link library (DLL).												
Syntax A	Declare Sub <i>name</i> [<i>libSpecification</i>] [(<i>parameter</i> [As <i>type</i>])]												
Syntax B	Declare Function <i>name</i> [<i>libSpecification</i>] [(<i>parameter</i> [As <i>type</i>])] [As <i>functype</i>]												
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>name</i></td><td>the subprogram or function procedure to declare.</td></tr> <tr> <td><i>libSpecification</i></td><td>the location of the procedure (module or DLL).</td></tr> <tr> <td><i>parameter</i></td><td>the arguments to pass to the procedure, separated by commas.</td></tr> <tr> <td><i>type</i></td><td>the type for the arguments.</td></tr> <tr> <td><i>functype</i></td><td>the type of the return value for a function procedure.</td></tr> </table>	where	is	<i>name</i>	the subprogram or function procedure to declare.	<i>libSpecification</i>	the location of the procedure (module or DLL).	<i>parameter</i>	the arguments to pass to the procedure, separated by commas.	<i>type</i>	the type for the arguments.	<i>functype</i>	the type of the return value for a function procedure.
where	is												
<i>name</i>	the subprogram or function procedure to declare.												
<i>libSpecification</i>	the location of the procedure (module or DLL).												
<i>parameter</i>	the arguments to pass to the procedure, separated by commas.												
<i>type</i>	the type for the arguments.												
<i>functype</i>	the type of the return value for a function procedure.												
Comments	A Sub procedure does not return a value. A Function procedure returns a value, and can be used in an expression. To specify the data type for the return value of a function, end the Function name with a type character or use the As <i>functype</i> clause shown above. If no type is provided, the function defaults to data type Variant .												

If the *libSpecification* is of the format:

BasicLib *libName* [**Alias** "*aliasname*"]

the procedure is in another Basic module named *libName*. The **Alias** keyword specifies that the procedure in *libName* is called *aliasname*. The other module will be loaded on demand whenever the procedure is called. SBL will not automatically unload modules which are loaded in this fashion. SBL will detect errors of mis-declaration.

If the *libSpecification* is of the format:

Lib *libName* [**Alias** ["*ordinal*"]]

or

Lib *libName* [**Alias** "*aliasname*"]

the procedure is in a Dynamic Link Library (DLL) named *libName*. The *ordinal* argument specifies the ordinal number of the procedure within the external DLL. Alternatively, *aliasname* specifies the name of the procedure within the external DLL. If neither *ordinal* nor *aliasname* is specified, the DLL function is accessed by name. It is recommended that the *ordinal* be used whenever possible, since accessing functions by name may cause the module to load more slowly.

A forward declaration is needed only when a procedure in the current module is referenced before it is defined. In this case, the **BasicLib**, **Lib** and **Alias** clauses are not used.

The data type of a parameter may be specified by using a type character or by using the **As** clause. Record parameters are declared by using an **As** clause and a *type* which has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*: array dimensions are not specified in the **Declare** statement.

External DLL procedures are called with the PASCAL calling convention (the actual arguments are pushed on the stack from left to right). By default, the actual arguments are passed by Far reference. For external DLL procedures, there are two additional keywords, **ByVal** and **Any**, that can be used in the parameter list.

When **ByVal** is used, it must be specified before the parameter it modifies. When applied to numeric data types, **ByVal** indicates that the parameter is passed by value, not by reference. When applied to string parameters, **ByVal** indicates that the string is passed by Far pointer to the string data. By default, strings are passed by Far pointer to a string descriptor.

Any can be used as a type specification, and permits a call to the procedure to pass a value of any datatype. When **Any** is used, type checking on the actual argument used in calls to the procedure is disabled (although other arguments not declared as type **Any** are fully type-safe). The actual argument is passed by Far reference, unless **ByVal** is specified, in which case the actual value is placed on the stack (or a pointer to the string in the case of string data). **ByVal** may also be used in the call. It is the external DLL procedure's responsibility to determine the type and size of the passed-in value.

When an empty string ("") is passed **ByVal** to an external procedure, the external procedure will receive a valid (non-NULL) pointer to a character of 0. To send a NULL pointer, **Declare** the procedure argument as **ByVal As Any**, and call the procedure with an argument of 0.

Example This example declares a function that is later called by the main subprogram. The function does nothing but set its return value to 1.

```

Declare Function SBL_exfunction()
Sub main
    Dim y as Integer
    Call SBL_exfunction
    y=SBL_exfunction
    MsgBox "The value returned by the function is: " & y
End Sub

Function SBL_exfunction()
    SBL_exfunction=1
End Function

```

See Also [Call](#), [Const](#), [Deftype](#), [Dim](#), [Static](#), [Type](#)

Deftype Statement

Action	Specifies the default data type for one or more variables.
Syntax	<p> DefCur <i>varTypeLetters</i> DefInt <i>varTypeLetters</i> DefLng <i>varTypeLetters</i> DefSng <i>varTypeLetters</i> DefDbl <i>varTypeLetters</i> DefStr <i>varTypeLetters</i> DefVar <i>varTypeLetters</i> </p> <p>where <i>varTypeLetters</i> is a first letter of the variable name to use.</p>
Comments	<p><i>VarTypeLetters</i> may be a single letter, a comma-separated list of letters, or a range of letters. For example, a-d indicates the letters a, b, c and d.</p> <p>The case of the letters is not important, even in a letter range. The letter range a-z is treated as a special case: it denotes all alpha characters, including the international characters.</p> <p>The Deftype statement affects only the module in which it is specified. It must precede any variable definition within the module.</p> <p>Variables defined using the Global or Dim may override the Deftype statement by using an As clause or a type character.</p>

Example This example finds the average of bowling scores entered by the user. Since the variable *average* begins with A, it is automatically defined as a single-precision floating point number. The other variables will be defined as Integers.

```
DefInt c,s,t
DefSng a
Sub main
  Dim count
  Dim total
  Dim score
  Dim average
  Dim msgtext
  For count=0 to 4
    score=InputBox("Enter bowling score #" & count+1 & ".")
    total=total+score
  Next count
  average=total/count
  msgtext="Your average is: " & average
  MsgBox msgtext
End Sub
```

See Also [Declare, Dim, Let, Type](#)

Dialog FunctionD

Action Displays a dialog box and returns a number for the button selected (-1= OK, 0=Cancel).

Syntax **Dialog** (*recordName*) where *recordName* is a variable name declared as a dialog box record.

Comments If the dialog box contains additional command buttons (for example, Help), the **Dialog** function returns a number greater than 0. 1 corresponds to the first command button, 2 to the second, and so on.

The dialog box *recordName* must have been declared using the **Dim** statement with the **As** parameter followed by a dialog box definition name. This name comes from the name argument used in the Begin Dialog statement.

To trap a user's selections within a dialog box, you must create a function and specify it as the last argument to the Begin Dialog statement. See **Begin Dialog** for more information.

The **Dialog** function does not return until the dialog box is closed.

Example This example creates a dialog box with a drop down combo box in it and three buttons: OK, Cancel, and Help. The Dialog function used here enables the subroutine to trap when the user clicks on any of these buttons.

```
Sub main
  Dim cchoices as String
  cchoices="All"+Chr$(9)+"Nothing"
  Begin Dialog UserDialog 180, 95, "SBL Dialog Box"
    ButtonGroup .ButtonGroup1
    Text 9, 3, 69, 13, "Filename:", .Text1
    ComboBox 9, 17, 111, 41, cchoices, .ComboBox1
    OkButton 131, 8, 42, 13
    CancelButton 131, 27, 42, 13
    PushButton 132, 48, 42, 13, "Help", .Push1
  End Dialog
  Dim mydialogbox As UserDialog
  answer= Dialog(mydialogbox)
  Select Case answer
    Case -1
      MsgBox "You pressed OK"
    Case 0
      MsgBox "You pressed Cancel"
    Case 1
      MsgBox "You pressed Help"
  End Select
End Sub
```

See Also [Begin Dialog...End Dialog, Dialog Statement](#)

Dialog Statement

Action	Displays a dialog box.
Syntax	Dialog <i>recordName</i> where <i>recordName</i> is a variable name declared as a dialog box record.
Comments	<p>The dialog box <i>recordName</i> must have been declared using the Dim statement with the As parameter followed by a dialog box definition name. This name comes from the name argument used in the Begin Dialog statement.</p> <p>If the user exits the dialog box by pushing the Cancel button, the run-time error 102 is triggered, which can be trapped using On Error.</p> <p>To trap a user's selections within a dialog box, you must create a function and specify it as the last argument to the Begin Dialog statement. See Begin Dialog for more information.</p> <p>The Dialog statement does not return until the dialog box is closed.</p>

Example This example defines and displays a dialog box defined as *UserDialog* and named *mydialogbox*. If the user presses the Cancel button, an error code of 102 is returned and is trapped by the If...Then statement listed after the Dialog statement.

```
Sub main
  Dim cchoices as String
  On Error Resume Next
  cchoices="All"+Chr$(9)+"Nothing"
  Begin Dialog UserDialog 180, 95, "SBL Dialog Box"
    ButtonGroup .ButtonGroup1
    Text 9, 3, 69, 13, "Filename:", .Text1
    ComboBox 9, 17, 111, 41, cchoices, .ComboBox1
    OkButton 131, 8, 42, 13
    CancelButton 131, 27, 42, 13
  End Dialog
  Dim mydialogbox As UserDialog
  Dialog mydialogbox
  If Err=102 then
    MsgBox "You pressed Cancel."
  Else
    MsgBox "You pressed OK."
  End If
End Sub
```

See Also [Begin Dialog...End Dialog](#), [Dialog Function](#)

Dim Statement

Action Declares variables for use in a Basic program.

Syntax **Dim** [**Shared**] *variableName* [**As** [**New**] *type*] [,*variableName* [**As** [**New**] *type*]] ... where *variableName* is the name of the variable to declare and *type* is the data type of the variable.

Comments *VariableName* must begin with a letter and contain only letters, numbers and underscores. A name may also be delimited by brackets, and any character may be used inside the brackets, except for other brackets.

Dim *my_1st_variable* **As String**

Dim [one long and strange! variable name] **As String**

If the **As** clause is not used, the type of the variable may be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single **Dim** statement (although not on the same variable).

Basic is a strongly typed language: all variables must be given a data type or they will be automatically assigned the data type **Variant**. The available data types are:

Arrays
Numbers
Objects
Records
Strings
Variants

Variables may be shared across modules. A variable declared inside a procedure has scope Local to that procedure. A variable declared outside a procedure has scope Local to the module. If you declare a variable with the same name as a module variable, the module variable is not accessible. See the **Global** statement for details.

The **Shared** keyword is included for backward compatibility with older versions of Basic. It is not allowed in **Dim** statements inside a procedure. It has no effect.

It is considered good programming practice to declare all variables. To force all variables to be explicitly declared use the **Option Explicit** statement. It is also recommended that you place all procedure-level **Dim** statements at the beginning of the procedure.

Regardless of which mechanism you use to declare a variable, you may choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

Arrays

The available data types for arrays are: numbers, strings, variants, objects and records. Arrays of arrays, dialog box records, and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

```
Dim variable( [ subscriptRange, ... ] ) As typeName
or
Dim variable_with_suffix( [ subscriptRange, ... ] )
```

where *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

Both the *startSubscript* and the *endSubscript* are valid subscripts for the array. The maximum number of subscripts which may be specified in an array definition is 60. The maximum total size for an array is only limited by the amount of memory available.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the **ReDim** statement must be used to specify the dimensions of the array before the array can be used.

Numbers Numeric variables can be declared using the **As** clause and one of the following numeric types: **Currency**, **Integer**, **Long**, **Single**, **Double**. Numeric variables can also be declared by including a type character as a suffix to the name. Numeric variables are initialized to 0.

Objects Object variables are declared using an **As** clause and a *typeName* of a **class**. Object variables may be **Set** to refer to an object, and then used to access members and methods of the object using dot notation.

```
Dim OLE2 As Object
Set OLE2 = CreateObject("spoly.cpoly")
OLE2.reset
```

An object may be declared as **New** for some classes. In such instances, the object variable does not need to be **Set**; a new object will be allocated when the variable is used. Note: The class **Object** does not support the **New** operator.

```
Dim variableName As New className
variableName.methodName
```

Records Record variables are declared by using an **As** clause and a *typeName* which has been defined previously using the **Type** statement. The syntax to use is:

```
Dim variableName As typeName
```

Records are made up of a collection of data elements called fields. These fields may be of any numeric, string, Variant, or previously defined record type. See **Type** for details on accessing fields within a record.

You can also use the **Dim** statement to declare a dialog box record. In this case, *type* is specified as *dialogName*, where *dialogName* matches a dialog box name previously defined using **Begin Dialog**. The dialog record variable can then be used in a **Dialog** statement.

Dialog box records have the same behavior as regular records; they differ only in the way they are defined. Some applications may provide a number of predefined dialog boxes.

Strings SBL supports two types of strings: fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

Dim *variableName* **As String****length*

Dynamic strings have no declared length, and can vary in length from 0 to 32,767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

Dim *variableName*\$
or **Dim** *variableName* **As String**

When initialized, fixed-length strings are filled with zeros. Dynamic strings are initialized as zero-length strings.

Variants Declare variables as Variants when the type of the variable is not known at the start of, or may change during, the procedure. For example, a Variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a Variant:

Dim *variableName*
or **Dim** *variableName* **As Variant**

Variant variables are initialized to vartype Empty.

Example This example shows a Dim statement for each of the possible data types.

```
Rem Must define a record type before you can declare a record variable
Type Testrecord
    Custno As Integer
    Custname As String
End Type
Sub main
    Dim counter As Integer
    Dim fixedstring As String*25
    Dim varstring As String
    Dim myrecord As Testrecord
    Dim ole2var As Object
    Dim F(1 to 10), A()
    '(code here)
End Sub
```

See Also [Global](#), [Option Base](#), [ReDim](#), [Set](#), [Static](#), [Type](#)

Dir Function

Action Returns a filename that matches the specified pattern.

Syntax **Dir**[\$] [(*pathname*\$ [,*attributes*%)] where *pathname*\$ is a string expression identifying a path or filename and *attributes*% is an integer expression specifying the file attributes to select.

Comments *Pathname*\$ may include a drive specification and wildcard characters ('?' and '*'). **Dir** returns the first filename that matches the *pathname*\$ argument. To retrieve additional matching filenames, call the **Dir** function again, omitting the *pathname*\$ and *attributes*% arguments. If no file is found, an empty string ("") is returned.

The default value for *attributes*% is 0. In this case, **Dir** returns only files without directory, hidden, system, or volume label attributes set.

Here are the possible values for *attributes*%:

Value	Meaning
0	return normal files
2	add hidden files
4	add system files
8	return volume label
16	add directories

The values in the table can be added together to select multiple attributes. For example, to list hidden and system files in addition to normal files set *attributes*% to 6 (6=2+4).

If *attributes*% is set to 8, the **Dir** function returns the volume label of the drive specified in the *pathname*\$. If no volume label attribute is set, all other attributes are ignored.

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

Example This example lists the contents of the diskette in drive A.

```
Sub main
    Dim msgret
    Dim directory, count
    Dim x, msgtext
    Dim A()
    msgret=MsgBox("Insert a disk in drive A.")
    count=1
    ReDim A(100)
    directory=Dir ("A:\*.*")
    Do While directory<>""
```

```

A(count)=directory
count=count+1
directory=Dir
Loop
msgtext="Contents of drive A:\ is:" & Chr(10) & Chr(10)
For x=1 to count
    msgtext=msgtext & A(x) & Chr(10)
Next x
MsgBox msgtext
End Sub

```

See Also [ChDir](#), [ChDrive](#), [CurDir](#), [MkDir](#), [Rmdir](#)

DlgControlID Function

Action	Returns the numeric ID for a dialog control in the active dialog box.
Syntax	DlgControlID (Id\$) where Id\$ is the string ID for a dialog control.
Comments	The DlgControlID function translates a string Id\$ into a numeric ID. Numeric ids correspond to the position of a control within a dialog box definition. The first control has ID 0 (zero), the second 1, and so on. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the TextBox or ComboBox statements. The string IDs does not include the period (.) and is case-sensitive.

Use **DlgControlID** only while a dialog box is running. See the **Begin Dialog** statement for more information.

Example This example displays a dialog box similar to File Open.

```

Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Dim filetypes as String
    Dim exestr$()
    Dim button as Integer
    Dim x as Integer
    Dim directory as String
    filetypes="Program files (*.exe)+Chr$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
        %CStrings Save
        Text 8, 6, 60, 11, "&Filename:"
        TextBox 8, 17, 76, 13, .TextBox1
        ListBox 9, 36, 75, 61, exestr$(), .ListBox1
        Text 8, 108, 61, 9, "List Files of &Type:"
        DropListBox 7, 120, 78, 30, filetypes, .DropListBox1
        Text 98, 7, 43, 10, "&Directories:"
        Text 98, 20, 46, 8, "c:\\windows"
        ListBox 99, 34, 66, 66, "", .ListBox2
    End Dialog
End Sub

```

```

Text 98, 108, 44, 8, "Dri&ves:"
DropListBox 98, 120, 68, 12, "", .DropListBox2
OkButton 177, 6, 50, 14
CancelButton 177, 24, 50, 14
PushButton 177, 42, 50, 14, "&Help"
'$CStrings Restore
End Dialog
Dim dlg As newdlg
button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
DlgText 1, str1$
x=0
Redim exestr$(x)
directory=Dir$("c:\windows\" & str1$, 16)
If directory<>"" then
Do
exestr$(x)=LCase$(directory)
x=x+1
Redim Preserve exestr$(x)
directory=Dir
Loop Until directory=""
End If
DlgListBoxArray 2, exestr$()
End Sub

Function FileDlgFunction(identifier$, action, supvalue)
Select Case action
Case 1
str1$="*.exe" 'dialog box initialized
ListFiles str1$
Case 2 'button or control value changed
If DlgControlId(identifier$) = 4 Then
If DlgText(4)="All Files (*.*)" then
str1$="*.*"
Else
str1$="*.exe"
End If
ListFiles str1$
End If
Case 3 'text or combo box changed
str1$=DlgText$(1)
ListFiles str1$
Case 4 'control focus changed

Case 5 'idle
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgEnable Function

Action	Returns the enable state for the specified dialog control (-1=enabled, 0=disabled).
Syntax	DlgEnable (<i>Id</i>) where <i>Id</i> is the numeric ID for the dialog control.
Comments	<p>If a dialog box control is enabled, it is accessible to the user. You may want to disable a control if its use depends on the selection of other controls.</p> <p>Use the DlgControlID function to find the numeric ID for a dialog control, based on its string identifier.</p> <p>Use DlgEnable only while a dialog box is running. See the Begin Dialog statement for more information.</p>

Example This example displays a dialog box with two checkboxes, one labeled Either, the other labeled Or. If the user clicks on Either, the Or option is grayed. Likewise, if Or is selected, Either is grayed. This example uses the DlgEnable statement to toggle the state of the buttons.

```

Declare Function FileDlgFunction(identifier$, action, supvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim supvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgEnable example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        CheckBox 34, 25, 75, 19, "Either", .CheckBox1
        CheckBox 34, 43, 73, 25, "Or", .CheckBox2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, supvalue)
    Select Case action
        Case 2 'button or control value changed
            If DlgControlID(identifier$) = 2 Then
                DlgEnable 3
            Else
                DlgEnable 2
            End If
        End Select
    End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgEnable Statement

Action	Enables, disables, or toggles the state of the specified dialog control.
Syntax	DlgEnable <i>Id</i> [, <i>mode</i>] where <i>Id</i> is the numeric ID for the dialog control to change. <i>Mode</i> is an integer representing the enable state (1=enable, 0=disable)
Comments	<p>If <i>mode</i> is omitted, the DlgEnable toggles the state of the dialog control specified by <i>Id</i>. If a dialog box control is enabled, it is accessible to the user. You may want to disable a control if its use depends on the selection of other controls.</p> <p>Use the DlgControlID function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the TextBox or ComboBox statements.</p> <p>Use DlgEnable only while a dialog box is running. See the Begin Dialog statement for more information.</p>
Example	<p>This example displays a dialog box with one checkbox , labeled Show More, and a group box, labeled More, with two option buttons, Option 1 and Option 2. It uses the DlgEnable function to enable the More group box and its options if the Show More checkbox is selected.</p>

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgEnable example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        CheckBox 13, 6, 75, 19, "Show more", .CheckBox1
        GroupBox 16, 28, 94, 50, "More"
        OptionGroup .OptionGroup1
            OptionButton 23, 40, 56, 12, "Option 1", .OptionButton1
            OptionButton 24, 58, 61, 13, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

```

```

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgEnable 3,0
            DlgEnable 4,0
            DlgEnable 5,0
        Case 2 'button or control value changed
            If DlgControlID(identifier$) = 2 Then
                If DlgEnable (3)=0 then
                    DlgEnable 3,1

```



```

        DlgEnable 4,1
        DlgEnable 5,1
    Else
        DlgEnable 3,0
        DlgEnable 4,0
        DlgEnable 5,0
    End If
End If
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgFocus Function

Action	Returns the numeric ID of the dialog control having the input focus.
Syntax	DlgFocus [\$]()
Comments	<p>A control has focus when it is active and responds to keyboard input.</p> <p>Use DlgFocus only while a dialog box is running. See the Begin Dialog statement for more information.</p>
Example	<p>This example displays a dialog box with a checkbox , labeled Check1, and a text box, labeled Text Box 1, in it. When the box is initialized, the focus is set to the text box. As soon as the user clicks the checkbox , the focus goes to the OK button.</p>

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgFocus Example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        TextBox 15, 37, 82, 12, .TextBox1
        Text 15, 23, 57, 10, "Text Box 1"
        CheckBox 15, 6, 75, 11, "Check1", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

```

```

Function FileDlgFunction(identifier$, action, supvalue)
Select Case action
Case 1
    DlgFocus 2
Case 2
    'user changed control or clicked a button
    If DlgFocus() <> "OkButton" then
        DlgFocus 0
    End If
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgFocus Statement

Action	Sets the focus for the specified dialog control.
Syntax	DlgFocus <i>Id</i> where <i>Id</i> is the ID for the dialog control to make active.
Comments	<p>Use the DlgControlID function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the TextBox or ComboBox statements.</p> <p>Use DlgFocus only while a dialog box is running. See the Begin Dialog statement for more information.</p>
Example	<p>This example displays a dialog box with a checkbox , labeled Check1, and a text box, labeled Text Box 1, in it. When the box is initialized, the focus is set to the text box. As soon as the user clicks the checkbox , the focus goes to the OK button.</p>

```

Declare Function FileDlgFunction(identifier$, action, supvalue)
Sub Main
    Dim button as Integer
    Dim identifier$
    Dim action as Integer
    Dim supvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgFocus Example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        TextBox 15, 37, 82, 12, .TextBox1
        Text 15, 23, 57, 10, "Text Box 1"
        CheckBox 15, 6, 75, 11, "Check1", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

```

```

Function FileDlgFunction(Identifier$, action, suppvale)
Select Case action
Case 1
    DlgFocus 2
Case 2      'user changed control or clicked a button
    If DlgFocus() <> "OkButton" then
        DlgFocus 0
    End If
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgListBoxArray Function

- Action** Returns the number of elements in a list or combo box.
- Syntax** **DlgListBoxArray** (*Id*[, *Array\$*]) where *Id* is the numeric ID for the list or combo box and *Array\$* is the entries in the list box or combo box returned.
- Comments** *Array\$* is a one-dimensional array of dynamic strings. If *array\$* is dynamic, its size is changed to match the number of strings in the list or combo box. If *array\$* is not dynamic and it is too small, an error occurs. If *array\$* is omitted, the function returns the number of entries in the specified dialog control.
- Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.
- Use **DlgListBoxArray** only while a dialog box is running. See the **Begin Dialog** statement for more information.
- Example** This example displays a dialog box with a checkbox , labeled “Display List”, and an empty list box. If the user clicks the checkbox , the list box is filled with the contents of the array called “myarray”. The DlgListBox Array function makes sure the list box is empty.

```

Declare Function FileDlgFunction(Identifier$, action, suppvale)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvale as Integer
    Begin Dialog newdlg 186, 92, "DlgListBoxArray Example", .FileDlgFunction
        'CStrings Save
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
    End Dialog
End Sub

```

```
ListBox 19, 26, 74, 59, "", .ListBox1
CheckBox 12, 4, 86, 13, "Display List", .CheckBox1
'CStrings Restore
End Dialog
Dim dlg As newdlg
button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, supvalue)
Dim myarray$(3)
Dim msgtext as Variant
Dim x as Integer
For x= 0 to 2
    myarray$(x)=Chr$(x+65)
Next x
Select Case action
Case 1
Case 2 'user changed control or clicked a button
    If DlgControlID(identifier$)=3 then
        If DlgListBoxArray(2)=0 then
            DlgListBoxArray 2, myarray$()
        End If
    End If
End Select
End Function
```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgListBoxArray Statement

Action	Fills a list or combo box with an array of strings.
Syntax	DlgListBoxArray <i>Id</i> , <i>Array\$</i> where <i>Id</i> is the ID for the list or combo box and <i>Array\$</i> is the entries for the list box or combo box.
Comments	<p><i>Array\$</i> has to be a one-dimensional array of dynamic strings. One entry appears in the list box for each element of the array. If the number of strings changes depending on other selections made in the dialog box, you should use a dynamic array and ReDim the size of the array whenever it changes.</p> <p>Use DlgListBoxArray only while a dialog box is running. See the Begin Dialog statement for more information.</p>

Example This example displays a dialog box similar to File Open.

```
Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, supvalue)

Sub main
    Dim identifier$
    Dim action as Integer
```

```

Dim supvalue as Integer
Dim filetypes as String
Dim exestr$()
Dim button as Integer
Dim x as Integer
Dim directory as String
filetypes="Program files (*.exe)+Chr$(9)+"All Files (*.*)"
Begin Dialog newdlg 230, 145, "Open", .FileDialogFunction
  $CStrings Save
  Text 8, 6, 60, 11, "&Filename:"
  TextBox 8, 17, 76, 13, .TextBox1
  ListBox 9, 36, 75, 61, exestr$(), .ListBox1
  Text 8, 108, 61, 9, "List Files of &Type:"
  DropListBox 7, 120, 78, 30, filetypes, .DropListBox1
  Text 98, 7, 43, 10, "&Directories:"
  Text 98, 20, 46, 8, "c:\windows"
  ListBox 99, 34, 66, 66, "", .ListBox2
  Text 98, 108, 44, 8, "Dir&ves:"
  DropListBox 98, 120, 68, 12, "", .DropListBox2
  OkButton 177, 6, 50, 14
  CancelButton 177, 24, 50, 14
  PushButton 177, 42, 50, 14, "&Help"
  $CStrings Restore
End Dialog
Dim dlg As newdlg
button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
  DlgText 1, str1$
  x=0
  Redim exestr$(x)
  directory=Dir$("c:\windows\" & str1$, 16)
  If directory<>"" then
    Do
      exestr$(x)=LCase$(directory)
      x=x+1
      Redim Preserve exestr$(x)
      directory=Dir
    Loop Until directory=""
  End If
  DlgListBoxArray 2, exestr$()
End Sub

Function FileDlgFunction(identifier$, action, supvalue)
  Select Case action
    Case 1
      str1$="*.exe"          'dialog box initialized
      ListFiles str1$
    Case 2                  'button or control value changed
      If DlgControlId(identifier$) = 4 Then
        If DlgText(4)="All Files (*.*)" then
          str1$="*.*"
        Else
          str1$="*.exe"
        End If
        ListFiles str1$
      End If
  End Select
End Function

```

```

Case 3          'text or combo box changed
  str1$=DlgText$(1)
  ListFiles str1$
Case 4          'control focus changed
Case 5          'idle
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgEnable](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgSetPicture Statement

Action Changes the picture in a picture dialog control for the current dialog box.

Syntax **DlgSetPicture** *Id*, *filename\$*, *type* where *Id* is the numeric ID for the picture dialog control, *filename\$* is the name of the bitmap file (.BMP) to use, and *type* is an integer representing the location of the file (0=*filename\$*, 3=Clipboard)

Comments Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

Use **DlgListBoxArray** only while a dialog box is running. See the **Begin Dialog** statement for more information.

See the **Picture** statement for more information about displaying pictures in dialog boxes.

Example This example displays a picture in a dialog box and changes the picture if the user clicks the checkbox labeled "Change Picture".

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
  Dim button as Integer
  Dim identifier$
  Dim action as Integer
  Dim suppvalue as Integer
  Begin Dialog newdlg 186, 92, "DlgSetPicture Example", .FileDlgFunction
    OkButton 130, 6, 50, 14
    CancelButton 130, 23, 50, 14
    Picture 43, 28, 49, 31, "C:\WINDOWS\THATCH.BMP", 0
    CheckBox 30, 8, 62, 15, "Change Picture", .CheckBox1
  End Dialog
  Dim dlg As newdlg
  button = Dialog(dlg)
End Sub

```

```

Function FileDlgFunction(identifier$, action, suppvalue)
Select Case action
Case 1
Case 2          'user changed control or clicked a button
If DlgControlID(identifier$)=3 then
If suppvalue=1 then
DlgSetPicture 2, "C:\WINDOWS\WINLOGO.BMP",0
Else
DlgSetPicture 2, "C:\WINDOWS\THATCH.BMP",0
End If
End If
End SelectEnd Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgText Function

Action	Returns the text associated with a dialog control for the current dialog box.
Syntax	DlgText [\$] (<i>Id</i>) where <i>Id</i> is the numeric ID for a dialog control.
Comments	<p>If the control is a text box or a combo box, DlgText function returns the text that appears in the text box. If it is a list box, the function returns its current selection. If it is a text box, DlgText returns the text. If the control is a command button, option button, option group, or a checkbox , the function returns its label.</p> <p>Use DlgText only while a dialog box is running. See the Begin Dialog statement for more information.</p>
Example	This example displays a dialog box similar to File Open. It uses DlgText to determine what group of files to display.

```

Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
Dim identifier$
Dim action as Integer
Dim suppvalue as Integer
Dim filetypes as String
Dim exestr$()
Dim button as Integer
Dim x as Integer
Dim directory as String
filetypes="Program files (*.exe)+Chr$(9)+"All Files (*.*)"
Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
'CStringgs Save
Text 8, 6, 60, 11, "&Filename:"
TextBox 8, 17, 76, 13, .TextBox1
ListBox 9, 36, 75, 61, exestr$(), .ListBox1

```

```

Text 8, 108, 61, 9, "List Files of &Type:"
DropListBox 7, 120, 78, 30, filetype$, .DropListBox1
Text 98, 7, 43, 10, "&Directories:"
Text 98, 20, 46, 8, "c:\\windows"
ListBox 99, 34, 66, 66, "", .ListBox2
Text 98, 108, 44, 8, "Drives:"
DropListBox 98, 120, 68, 12, "", .DropListBox2
OkButton 177, 6, 50, 14
CancelButton 177, 24, 50, 14
PushButton 177, 42, 50, 14, "&Help"
'$CStrings Restore
End Dialog
Dim dlg As newdlg
button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
DlGText 1, str1$
x=0
Redim exestr$(x)
directory=Dir$("c:\\windows\\" & str1$, 16)
If directory<>"" then
Do
exestr$(x)=LCase$(directory)
x=x+1
Redim Preserve exestr$(x)
directory=Dir
Loop Until directory=""
End If
DlGListBoxArray 2, exestr$()
End Sub

Function FileDlgFunction(identifier$, action, suppvale)
Select Case action
Case 1
str1$="*.exe" 'dialog box initialized
ListFiles str1$
Case 2 'button or control value changed
If DlgControlID(identifier$) = 4 Then
If DlgText(4)="All Files (*.*)" then
str1$="*.*"
Else
str1$="*.exe"
End If
ListFiles str1$
End If
Case 3 'text or combo box changed
str1$=DlgText$(1)
ListFiles str1$
Case 4 'control focus changed

Case 5 'idle
End Select
End Function

```

See Also

[BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgText Statement

Action	Changes the text associated with a dialog control for the current dialog box.
Syntax	DlgText <i>Id</i> , <i>text\$</i> where <i>Id</i> is the numeric ID for a dialog control and <i>text\$</i> is the text to use for the dialog control.
Comments	If the dialog control is a text box or a combo box, DlgText sets the text that appears in the text box. If it is a list box, a string equal to <i>text\$</i> or beginning with <i>text\$</i> is selected. If the dialog control is a text control, DlgText sets it to <i>text\$</i> . If the dialog control is a command button, option button, option group, or a checkbox, the statement sets its label.

The **DlgText** statement does not change the identifier associated with the control.

Use **DlgText** only while a dialog box is running. See the **Begin Dialog** statement for more information.

Example This example displays a dialog box similar to File Open. It uses the DlgText statement to display the list of files in the Filename list box.

```

Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, supvalue)

Sub main
  Dim identifier$
  Dim action as Integer
  Dim supvalue as Integer
  Dim filetypes as String
  Dim exestr$()
  Dim button as Integer
  Dim x as Integer
  Dim directory as String
  filetypes="Program files (*.exe)+Chr$(9)+"All Files (*.*)"
  Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
    %CStrings Save
    Text 8, 6, 60, 11, "&Filename:"
    TextBox 8, 17, 76, 13, .TextBox1
    ListBox 9, 36, 75, 61, exestr$(), .ListBox1
    Text 8, 108, 61, 9, "List Files of &Type:"
    DropListBox 7, 120, 78, 30, filetypes, .DropListBox1
    Text 98, 7, 43, 10, "&Directories:"
    Text 98, 20, 46, 8, "c:\windows"
    ListBox 99, 34, 66, 66, "", .ListBox2
    Text 98, 108, 44, 8, "Dri&ves:"
    DropListBox 98, 120, 68, 12, "", .DropListBox2
    OkButton 177, 6, 50, 14
    CancelButton 177, 24, 50, 14
    PushButton 177, 42, 50, 14, "&Help"
    %CStrings Restore
  End Dialog
  Dim dlg As newdlg
  button = Dialog(dlg)
End Sub

```

```

Sub ListFiles(str1$)
DlgText 1,str1$
x=0
Redim exestr$(x)
directory=Dir$("c:\windows\" & str1$,16)
If directory<>"" then
  Do
    exestr$(x)=LCase$(directory)
    x=x+1
    Redim Preserve exestr$(x)
  directory=Dir
  Loop Until directory=""
End If
DlgListBoxArray 2,exestr$()
End Sub

Function FileDlgFunction(identifier$, action, supvalue)
Select Case action
Case 1
  str1$="*.exe"          'dialog box initialized
  ListFiles str1$
Case 2                  'button or control value changed
  If DlgControlId(identifier$) = 4 Then
    If DlgText(4)="All Files (*.*)" then
      str1$="*.*"
    Else
      str1$="*.exe"
    End If
    ListFiles str1$
  End If
Case 3                  'text or combo box changed
  str1$=DlgText$(1)
  ListFiles str1$
Case 4                  'control focus changed

Case 5                  'idle
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgValue Function

Action Returns a numeric value for the state of a dialog control for the current dialog box.

Syntax **DlgValue** (*Id*) where *Id* is the numeric ID for a dialog control.

Comments The values returned depend on the type of dialog control:

Control	Value Returned
Checkbox	1 = Selected, 0=Cleared, -1=Grayed
Option Group	0 = 1st button selected, 1 = 2nd button selected, etc.
Listbox	0 = 1st item, 1= 2nd item, etc.
Combobox	0 = 1st item, 1 = 2nd item, etc.
Text, Textbox, Button	Error occurs

Use DlgValue only while a dialog box is running. See the Begin Dialog statement for more information.

Example This example changes the picture in the dialog box if the checkbox is selected and changes the picture to its original bitmap if the checkbox is turned off.

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
  Dim button as integer
  Dim identifier$
  Dim action as Integer
  Dim suppvalue as Integer
  Begin Dialog newdlg 186, 92, "DlgSetPicture Example", .FileDlgFunction
    OkButton 130, 6, 50, 14
    CancelButton 130, 23, 50, 14
    Picture 43, 28, 49, 31, "C:\WINDOWS\THATCH.BMP", 0
    CheckBox 30, 8, 62, 15, "Change Picture", .CheckBox1
  End Dialog
  Dim dlg As newdlg
  button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
  Select Case action
    Case 1
    Case 2 'user changed control or clicked a button
      If DlgControlID(identifier$)=3 then
        If DlgValue(3)=1 then
          DlgSetPicture 2, "C:\WINDOWS\WINLOGO.BMP",0
        Else
          DlgSetPicture 2, "C:\WINDOWS\THATCH.BMP",0
        End If
      End If
    End Select
  End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Statement](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgValue Statement

Action Changes the value associated with the dialog control for the current dialog box.

Syntax **DlgValue** *Id*, *value%* where *Id* is the numeric ID for a dialog control and *value%* is the new value for the dialog control.

Comments The values you use to set the control depend on the type of the control:

Control	Value Returned
Checkbox	1 = Select, 0=Clear, -1=Gray.
Option Group	0 = Select 1st button, 1 = Select 2nd button.
Listbox	0 = Select 1st item, 1= Select 2nd item, etc.
Combobox	0 = Select 1st item, 1 = Select 2nd item, etc.
Text, Textbox, Button	Error occurs

Use DlgValue only while a dialog box is running. See the Begin Dialog statement for more information.

Example This example displays a dialog box with a checkbox, labeled Change Option, and a group box with two option buttons, labeled Option 1 and Option 2. When the user clicks the Change Option button, Option 2 is selected.

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgValue Example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        CheckBox 30, 8, 62, 15, "Change Option", .CheckBox1
        GroupBox 28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
        OptionButton 41, 47, 52, 10, "Option 1", .OptionButton1
        OptionButton 41, 62, 58, 11, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

```

```

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
        Case 2 'user changed control or clicked a button
            If DlgControlID(identifier$)=2 then
                If DlgValue(2)=1 then
                    DlgValue 4,1
                End If
            End If
    End Select
End Function

```

```

Else
    DlgValue 4,0
End If
End If
End Select
End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgVisible Function](#), [DlgVisible Statement](#)

DlgVisible Function

- Action** Returns -1 if a dialog control is visible, 0 if it is hidden.
- Syntax** **DlgVisible** (*Id*) where *Id* is the numeric ID for a dialog control.
- Comments** Use **DlgVisible** only while a dialog box is running. See the **Begin Dialog** statement for more information.
- Example** This example displays Option 2 in the Group box if the user clicks the checkbox labeled “Show Option 2”. If the user clicks the box again, Option 2 is hidden.

```

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgVisible Example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        CheckBox 30, 8, 62, 15, "Show Option 2", .CheckBox1
        GroupBox 28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
            OptionButton 41, 47, 52, 10, "Option 1", .OptionButton1
            OptionButton 41, 62, 58, 11, "Option 2", .OptionButton2
        End Dialog
        Dim dlg As newdlg
        button = Dialog(dlg)
    End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgVisible 6,0

        Case 2
            'user changed control or clicked a button
            If DlgControlID(identifier$)=2 then
                If DlgVisible(6)<>1 then
                    DlgVisible 6
                End If
            End If
        End Select
    End Function

```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgValue Statement](#), [DlgVisible Statement](#)

DlgVisible Statement

Action Hides or displays a dialog control for the current dialog box.

Syntax **DlgVisible** *Id* [, *mode*] where *Id* is the numeric ID for a dialog control and *mode* is the value to use to set the dialog control state:

1 = Display a previously hidden control.
0 = Hide the control.

Comments If you omit the *mode*, the dialog box state is toggled between visible and hidden.

Use **DlgVisible** only while a dialog box is running. See the **Begin Dialog** statement for more information.

Example This example displays Option 2 in the Group box if the user clicks the checkbox . labeled “Show Option 2”. If the user clicks the box again, Option 2 is hidden.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgVisible Example", .FileDlgFunction
        OkButton 130, 6, 50, 14
        CancelButton 130, 23, 50, 14
        CheckBox 30, 8, 62, 15, "Show Option 2", .CheckBox1
        GroupBox 28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
            OptionButton 41, 47, 52, 10, "Option 1", .OptionButton1
            OptionButton 41, 62, 58, 11, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub
```

```
Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgVisible 6,0
        Case 2 'user changed control or clicked a button
            If DlgControlID(identifier$)=2 then
                If DlgVisible(6)<>1 then
                    DlgVisible 6
                End If
            End If
        End Select
    End Function
```

See Also [BeginDialog...End Dialog](#), [DlgControlID Function](#), [DlgEnable Function](#), [DlgEnable Statement](#), [DlgFocus Function](#), [DlgFocus Statement](#), [DlgListBoxArray Function](#), [DlgListBoxArray Statement](#), [DlgSetPicture](#), [DlgText Function](#), [DlgText Statement](#), [DlgValue Function](#), [DlgVisible Function](#)

Do...Loop Statement

Action	Repeats a series of program lines as long as (or until) an expression is TRUE.
Syntax A	Do [{ While Until } <i>condition</i>] [<i>statementblock</i>] [Exit Do] [<i>statementblock</i>]
Syntax B	Loop Do [<i>statementblock</i>] [Exit Do] [<i>statementblock</i>] Loop [{ While Until } <i>condition</i>] where <i>Condition</i> is any expression that evaluates to TRUE (nonzero) or FALSE (0) and <i>statementblock(s)</i> is the program lines to repeat while (or until) <i>condition</i> is TRUE.
Comments	When an Exit Do statement is executed, control goes to the statement after the Loop statement. When used within a nested loop, an Exit Do statement moves control out of the immediately enclosing loop.
Example	This example lists the contents of the diskette in drive A. <pre> Sub main Dim msgret Dim directory, count Dim x, msgtext Dim A() msgret=MsgBox("Insert a disk in drive A.") count=1 ReDim A(100) directory=Dir ("A:*.**") Do While directory<>"" A(count)=directory count=count+1 directory=Dir Loop msgtext="Directory of drive A:\ is:" & Chr(10) For x=1 to count msgtext=msgtext & A(x) & Chr(10) Next x MsgBox msgtext End Sub </pre>
See Also	Exit , For...Next , Stop , While...Wend

DoEvents Statement

Action	Yields execution to Windows for processing operating system events.
Syntax	DoEvents
Comments	<p>DoEvents does not return until Windows has finished processing all events in the queue and all keys sent by SendKeys statement.</p> <p>DoEvents should not be used if other tasks can interact with the running program in unforeseen ways. Since SBL yields control to the operating system at regular intervals, DoEvents should only be used to force SBL to allow other applications to run at a known point in the program.</p>
Example	<p>This example activates the Windows Terminal application, dials the number and then allows the operating system to process events.</p> <pre> Sub main Dim phonenumber, msgtext Dim x phonenumber=InputBox("Type telephone number to call:") x=Shell("Terminal.exe",1) SendKeys "%PD" & phonenumber & "{Enter}",1 msgtext="Dialing..." MsgBox msgtext DoEvents End Sub </pre>
See Also	AppActivate , SendKeys , Shell

DropComboBox Statement

Action	Creates a combination of a drop-down list box and a text box.
Syntax A	DropComboBox <i>x, y, dx, dy, text\$, .field</i>
Syntax B	DropComboBox <i>x, y, dx, dy, stringarray\$(), .field</i>
where	is
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.
<i>text\$</i>	a string containing the selections for the combo box.
<i>stringarray\$</i>	an array of dynamic strings for the selections in the combo box.
<i>.field</i>	the name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments

The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text\$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text\$* string are entered as shown in the following example:

```
dimname = "listchoice"+Chr$(9)+"listchoice"+Chr$(9)+"listchoice"...
```

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *.field* argument is also used by the dialog statements that act on this control.

You use a drop combo box when you want the user to be able to edit the contents of the list box (such as filenames or their paths). You use a drop list box when the items in the list should remain unchanged.

Use the **DropComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example

This example defines a dialog box with a drop combo box and the OK and Cancel buttons.

```
Sub main
  Dim cchoices as String
  On Error Resume Next
  cchoices="All"+Chr$(9)+"Nothing"
  Begin Dialog UserDialog 180, 95, "SBL Dialog Box"
    ButtonGroup .ButtonGroup1
    Text 9, 3, 69, 13, "Filename:", .Text1
    DropComboBox 9, 17, 111, 41, cchoices, .ComboBox1
    OkButton 131, 8, 42, 13
    CancelButton 131, 27, 42, 13
  End Dialog
  Dim mydialogbox As UserDialog
  Dialog mydialogbox
  If Err=102 then
    MsgBox "You pressed Cancel."

  Else
    MsgBox "You pressed OK."
  End If
End Sub
```

See Also

[Begin Dialog...End Dialog Statement](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [DropListBox](#), [GroupBox](#), [ListBox](#), [OkButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

DropListBox Statement

Action Creates a drop-down list of choices.

Syntax A **DropListBox** *x*, *y*, *dx*, *dy*, *text\$*, *.field*

Syntax B **DropListBox** *x*, *y*, *dx*, *dy*, *stringarray\$()*, *.field*

where	is
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.
<i>text\$</i>	a string containing the selections for the combo box.
<i>stringarray\$</i>	an array of dynamic strings for the selections in the combo box.
<i>.field</i>	the name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text\$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text\$* string are entered as shown in the following example:

```
dimname = "listchoice"+Chr$(9)+"listchoice"+Chr$(9)+"listchoice"...
```

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

A drop list box is different from a list box. The drop list box only displays its list when the user selects it; the list box also displays its entire list in the dialog box.

Use the **DropListBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a drop list box and the OK and Cancel buttons.

```
Sub main
  Dim DropListBox1() as String
  ReDim DropListBox1(3)
  For x=0 to 2
    DropListBox1(x)=Chr(65+x) & ":"
  Next x
  Begin Dialog UserDialog 186, 62, "SBL Dialog Box"
```

```

Text 8, 4, 42, 8, "Drive:", .Text3
DropListBox 8, 16, 95, 44, DropListBox1(), .DropListBox1
OkButton 124, 6, 54, 14
CancelButton 124, 26, 54, 14
End Dialog
Dim mydialog as UserDialog
On Error Resume Next
Dialog mydialog
If Err=102 then
    MsgBox "Dialog box canceled."
End If
End Sub

```

See Also [Begin Dialog...End Dialog Statement](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [DropComboBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Environ Function

Action Returns the string setting for a keyword in the operating system's environment table.

Syntax A **Environ**[\$](*environment-string*\$)

Syntax B **Environ**[\$](*numeric expression*%)

where *Environment-string*\$ is the name of a keyword in the operating system environment and *Numeric expression*% is a number for the position of the string in the environment table. (1st, 2nd, 3rd, etc.)

Comments If you use the *environment string*\$ parameter, enter it in uppercase, or **Environ** returns a null string (""). The return value for Syntax A is the string associated with the keyword requested.

If you use the *numeric expression*% parameter, the numeric expression is automatically rounded to a whole number, if necessary. The return value for Syntax B is a string in the form "keyword=value".

Environ returns a null string if the specified argument cannot be found.

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

Example This example lists all the strings from the operating system environment table.

```

Sub main
    Dim str1(100)
    Dim msgtext
    Dim count, x
    Dim newline
    newline=Chr(10)
    x=1
    str1(x)= Environ(x)

```

```

Do While Environ(x)<>""
  str1(x)= Environ(x)
  x=x+1
  str1(x)=Environ(x)
Loop
msgtext="The Environment Strings are:" & newline & newline
count=x
For x=1 to count
  msgtext=msgtext & str1(x) & newline
Next x
MsgBox msgtext
End Sub

```

Eof Function

Action	Returns the value -1 if the end of the specified open file has been reached, 0 otherwise.
Syntax	Eof (<i>filenumber%</i>) where <i>filenumber%</i> is an integer expression identifying the open file to use.
Comments	See the Open statement for more information about assigning numbers to files when they are opened.
Example	This example uses the Eof function to read records from a Random file, using a Get statement. The Eof function keeps the Get statement from attempting to read beyond the end of the file. The subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```

Declare Sub createfile()
Sub main
  Dim acctno
  Dim msgtext as String
  newline=Chr(10)
  Call createfile
  Open "C:\temp001" For Input As #1
  msgtext="The account numbers are:" & newline
  Do While Not Eof(1)
    Input #1,acctno
    msgtext=msgtext & newline & acctno & newline
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub

```

```

Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
  Close #1
End Sub

```

See Also [Get, Input Function, Input Statement, Line Input, Loc, Lof, Open](#)

Erase Statement

Action	Reinitializes the contents of a fixed array or frees the storage associated with a dynamic array.														
Syntax	Erase <i>Array</i> [, <i>Array</i>] where <i>Array</i> is the of the array variable to re-initialize.														
Comments	<p>The effect of using Erase on the elements of a fixed array varies with the type of the element:</p> <table> <tr> <th>Element Type</th><th>Erase Effect</th></tr> <tr> <td>numeric</td><td>Each element set to zero.</td></tr> <tr> <td>variable length string</td><td>Each element set to zero length string.</td></tr> <tr> <td>fixed length string</td><td>Each element's string is filled with zeros.</td></tr> <tr> <td>Variant</td><td>Each element set to Empty.</td></tr> <tr> <td>user-defined type</td><td>Members of each element are cleared as if the members were array elements, i.e. numeric members have their value set to zero, etc.</td></tr> <tr> <td>object</td><td>Each element is set to the special value Nothing.</td></tr> </table>	Element Type	Erase Effect	numeric	Each element set to zero.	variable length string	Each element set to zero length string.	fixed length string	Each element's string is filled with zeros.	Variant	Each element set to Empty.	user-defined type	Members of each element are cleared as if the members were array elements, i.e. numeric members have their value set to zero, etc.	object	Each element is set to the special value Nothing.
Element Type	Erase Effect														
numeric	Each element set to zero.														
variable length string	Each element set to zero length string.														
fixed length string	Each element's string is filled with zeros.														
Variant	Each element set to Empty.														
user-defined type	Members of each element are cleared as if the members were array elements, i.e. numeric members have their value set to zero, etc.														
object	Each element is set to the special value Nothing.														
Example	<p>This example prompts for a list of item numbers to put into an array and clears array if the user wants to start over.</p> <pre> Sub main Dim msgtext Dim inum(100) as Integer Dim x, count Dim newline newline=Chr(10) x=1 count=x inum(x)=0 Do inum(x)=InputBox("Enter item #" & x & " (99=start over;0=end):") If inum(x)=99 then Erase inum() x=0 ElseIf inum(x)=0 then Exit Do End If x=x+1 Loop count=x-1 msgtext="You entered the following numbers:" & newline For x=1 to count msgtext=msgtext & inum(x) & newline Next x MsgBox msgtext End Sub </pre>														

See Also [Dim](#), [ReDim](#), [Lbound](#), [UBound](#)

Erl Function

Action	Returns the line number where an error was trapped.
Syntax	Erl
Comments	If you use a Resume or On Error statement after Erl, the return value for Erl is reset to 0. To maintain the value of the line number returned by Erl, assign it to a variable.

The value of the **Erl** function can be set indirectly through the **Error** statement.

Example This example prints the error number using the Err function and the line number using the Erl statement if an error occurs during an attempt to open a file. Line numbers are automatically assigned, starting with 1, which is the **Sub main** statement.

```
Sub main
  Dim msgtext, userfile
  On Error GoTo Debugger
  msgtext="Enter the filename to use:"
  userfile=InputBox$(msgtext)
  Open userfile For Input As #1
  MsgBox "File opened for input."
  ' ....etc....
Close #1
done:
  Exit Sub
Debugger:
  msgtext="Error number " & Err & " occurred at line: " & Erl
  MsgBox msgtext
  Resume done
End Sub
```

See Also [Err Function](#), [Err Statement](#), [Error Function](#), [Error Statement](#), [On Error](#), [Resume](#), [Trappable Errors](#)

Err Function

Action	Returns the run-time error code for the last error trapped.
Syntax	Err
Comments	If you use a Resume or On Error statement after Err, the return value for Err is reset to 0. To maintain the value of the line number returned by Erl, assign it to a variable.

The value of the **Err** function can be set directly through the **Err** statement, and indirectly through the **Error** statement.

The **Trappable Errors** are listed in an appendix.

Example This example prints the error number using the Err function and the line number using the Erl statement if an error occurs during an attempt to open a file. Line numbers are automatically assigned, starting with 1, which is the **Sub main** statement.

```
Sub main
  Dim msgtext, userfile
  On Error GoTo Debugger
  msgtext="Enter the filename to use:"
  userfile=InputBox$(msgtext)
  Open userfile For Input As #1
  MsgBox "File opened for input."
  ' ...etc....
  Close #1
done:
  Exit Sub
Debugger:
  msgtext="Error number " & Err & " occurred at line: " & Erl
  MsgBox msgtext
  Resume done
End Sub
```

See Also [Erl](#), [Err Statement](#), [Error Function](#), [Error Statement](#), [On Error](#), [Resume](#), [Trappable Errors](#)

Err Statement

Action	Sets a run-time error code.
Syntax	Err = <i>n%</i> where <i>n%</i> is an integer expression for the error code (between 1 and 32,767) or 0 for no run-time error.
Comments	The Err statement is used to send error information between procedures.
Example	<p>This example generates an error code of 10000 and displays an error message if a user does not enter a customer name when prompted for it. It uses the Err statement to clear any previous error codes before running the loop the first time and it also clears the error to allow the user to try again.</p> <pre>Sub main Dim custname as String On Error Resume Next Do Err=0 custname=InputBox\$("Enter customer name:") If custname="" then Error 10000 Else Exit Do End If Select Case Err Case 10000 MsgBox "You must enter a customer name." Case Else MsgBox "Undetermined error. Try again."</pre>

```

End Select
Loop Until custname<>""
MsgBox "The name is: " & custname
End Sub

```

See Also [Erl, Err Function, Error Function, Error Statement, On Error, Resume, Trappable Errors](#)

Error Function

Action	Returns the error message that corresponds to the specified error code.
Syntax	Error[\$] [(<i>errornumber%</i>)] where <i>errornumber%</i> is an integer between 1 and 32,767 for the error code.
Comments	<p>If this argument is omitted, SBL returns the error message for the run-time error which has occurred most recently.</p> <p>If no error message is found to match the errorcode, "" (a null string) is returned.</p> <p>The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will return a Variant of vartype 8 (string).</p> <p>The Trappable Errors are listed in an appendix.</p>
Example	This example prints the error number, using the Err function, and the text of the error, using the Error\$ function, if an error occurs during an attempt to open a file.

```

Sub main
Dim msgtext, userfile
On Error GoTo Debugger
msgtext="Enter the filename to use:"
userfile=InputBox$(msgtext)
Open userfile For Input As #1
MsgBox "File opened for input."
' ....etc....
Close #1
done:
Exit Sub
Debugger:
msgtext="Error " & Err & ": " & Error$
MsgBox msgtext
Resume done
End Sub

```

See Also [Erl, Err Function, Err Statement, Error Statement, On Error, Resume, Trappable Errors](#)

Error Statement

Action	Simulates the occurrence of a SBL or user-defined error.
Syntax	Error <i>errornumber%</i> where <i>errornumber%</i> is an integer between 1 and 32,767 for the error code.
Comments	<p>If an <i>errornumber%</i> is one which SBL already uses, the Error statement will simulate an occurrence of that error.</p> <p>User-defined error codes should employ values greater than those used for standard SBL error codes. To help ensure that non-SBL error codes are chosen, user-defined codes should work down from 32,767.</p> <p>If an Error statement is executed, and there is no error-handling routine enabled, SBL produces an error message and halts program execution. If an Error statement specifies an error code not used by SBL, the message “User-defined error” is displayed.</p>
Example	<p>This example generates an error code of 10000 and displays an error message if a user does not enter a customer name when prompted for it.</p> <pre>Sub main Dim custname as String On Error Resume Next Do Err=0 custname=InputBox("Enter customer name:") If custname="" then Error 10000 Else Exit Do End If Loop Select Case Err Case 10000 MsgBox "You must enter a customer name." Case Else MsgBox "Undetermined error. Try again." End Select Loop Until custname<>"" MsgBox "The name is: " & custname End Sub</pre>
See Also	Erl , Err Function , Err Statement , Error Function , On Error , Resume , Trappable Errors

Exit Statement

Action	Terminates Loop statements or transfers control to a calling procedure.
Syntax	Exit {Do For Function Sub}
Comments	<p>Use Exit Do inside a Do...Loop statement. Use Exit For inside a For...Next statement. When the Exit statement is executed, control transfers to the statement after the Loop or Next statement. When used within a nested loop, an Exit statement moves control out of the immediately enclosing loop.</p> <p>Use Exit Function inside a Function...End Function procedure. Use Exit Sub inside a Sub...End Sub procedure.</p>
Example	<p>This example uses the On Error statement to trap run-time errors. If there is an error, the program execution continues at the label “Debugger”. The example uses the Exit statement to skip over the debugging code when there is no error.</p> <pre> Sub main Dim msgtext, userfile On Error GoTo Debugger msgtext="Enter the filename to use:" userfile=InputBox\$(msgtext) Open userfile For Input As #1 MsgBox "File opened for input." etc.... Close #1 done: Exit Sub Debugger: msgtext="Error " & Err & ": " & Error\$ MsgBox msgtext Resume done End Sub </pre>

See Also [Do...Loop, For...Next, Function...End Function, Stop, Sub...End Sub](#)

Exp Function

Action	Returns the value e (the base of natural logarithms) raised to a power.
Syntax	Exp (<i>number</i>) where <i>number</i> is the exponent value for e .
Comments	<p>If the variable to contain the return value has a data type Integer, Currency, or Single, the return value is a single-precision value. If the variable has a date type of Long, Variant, or Double, the value returned is a double-precision number.</p> <p>The constant e is approximately 2.718282.</p>

Example This example estimates the value of a factorial of a number entered by the user. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of $5*4*3*2*1$, or the value 120.

```
Sub main
  Dim x as Single
  Dim msgtext, PI
  Dim factorial as Double
  PI=3.14159
i: x=InputBox("Enter an integer between 1 and 88: ")
  If x<=0 then
    Exit Sub
  ElseIf x>88 then
    MsgBox "The number you entered is too large. Try again."
    Goto i
  End If
  factorial=Sqr(2*PI*x)*(x^x/Exp(x))
  msgtext="The estimated factorial is: " & Format(factorial, "Scientific")
  MsgBox msgtext
End Sub
```

See Also [Abs](#), [Fix](#), [Int](#), [Log](#), [Rnd](#), [Sgn](#), [Sqr](#)

FileAttr Function

Action Returns the file mode or the operating system handle for the open file.

Syntax **FileAttr**(*filenumber%*, *returntype*) where *filenumber%* is an integer expression identifying the open file to use and *returntype* is 1=Return file mode, 2=Return operating system handle

Comments The argument *filenumber%* is the number used in the **Open** statement to open the file.

The following table lists the return values and corresponding file modes if *returntype* is 1:

Value	Mode
1	Input
2	Output
8	Append

Example This example closes an open file if it is open for Input or Output. If open for Append, it writes a range of numbers to the file. The second subprogram, CREATEFILE, creates the file and leaves it open.

```
Declare Sub createfile()
Sub main
  Dim filemode as Integer
  Dim attrib as Integer
```

```

Call createfile
attrib=1
filemode=FileAttr(1,attrib)
If filemode=1 or 2 then
  MsgBox "File was left open. Closing now."
  Close #1
Else
  For x=11 to 15
    Write #1, x
  Next x
  Close #1
End If
Kill "C:\TEMP001"
End Sub

Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
End Sub

```

See Also [GetAttr](#), [Open](#), [SetAttr](#)

FileCopy Statement

Action	Copies a file.
Syntax	FileCopy <i>source</i> \$, <i>destination</i> \$ where <i>source</i> \$ is a string expression for the name (and path) of the file to copy and <i>destination</i> \$ is a string expression for the name (and path) for the copied file.
Comments	Wildcards (* or ?) are not allowed for either the <i>source</i> \$ or <i>destination</i> \$. The <i>source</i> \$ file cannot be copied if it is opened by SBL for anything other than Read access.
Example	This example copies one file to another. Both filenames are specified by the user.

```

Sub main
  Dim oldfile, newfile
  On Error Resume Next
  oldfile= InputBox("Copy which file?")
  newfile= InputBox("Copy to?")
  FileCopy oldfile,newfile
  If Err<>0 then
    msgtext="Error during copy. Rerun program."
  Else
    msgtext="Copy successful."
  End If
  MsgBox msgtext
End Sub

```

See Also [FileAttr](#), [FileDateTime](#), [GetAttr](#), [Kill](#), [Name](#)

FileDateTime Function

Action	Returns the last modification date and time for the specified file.
Syntax	FileDateTime (<i>pathname\$</i>) where <i>pathname\$</i> is a string expression for the name of the file to query.
Comments	<i>Pathname\$</i> can contain path and disk information, but cannot include wildcards (* and ?).
Example	This example writes data to a file if it hasn't been saved within the last 2 minutes.

```

Sub main
  Dim tempfile
  Dim filetype, curtime
  Dim msgtext
  Dim acctno(100) as Single
  Dim x, l
  tempfile="C:\TEMP001"
  Open tempfile For Output As #1
  filetype=FileDateTime(tempfile)
  x=1
  l=1
  acctno(x)=0
  Do
    curtime=Time
    acctno(x)=InputBox("Enter an account number (99 to end):")
    If acctno(x)=99 then
      For l=1 to x-1
        Write #1, acctno(l)
      Next l
      Exit Do
    ElseIf (Minute(filetime)+2)<=Minute(curtime) then
      For l=1 to x
        Write #1, acctno(l)
      Next l
    End If

    x=x+1
  Loop
  Close #1
  x=1
  msgtext="Contents of C:\TEMP001 is:" & Chr(10)
  Open tempfile for Input as #1
  Do While Eof(1)<>-1
    Input #1, acctno(x)
    msgtext=msgtext & Chr(10) & acctno(x)
    x=x+1
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub

```

See Also [FileLen](#), [GetAttr](#)

FileLen Function

Action	Returns the length of the specified file.
Syntax	FileLen (<i>pathname\$</i>) where <i>pathname\$</i> is a string expression that contains the name of the file to query.
Comments	<p><i>Pathname\$</i> can contain path and disk information, but cannot include wildcards (* and ?).</p> <p>If the specified file is open, FileLen returns the length of the file before it was opened.</p>

Example This example returns the length of a file.

```
Sub main
  Dim length as Long
  Dim userfile as String
  Dim msgtext
  On Error Resume Next
  msgtext="Enter a filename:"
  userfile=InputBox(msgtext)
  length=FileLen(userfile)
  If Err<>0 then
    msgtext="Error occurred. Rerun program."
  Else
    msgtext="The length of " & userfile & " is: " & length
  End If
  MsgBox msgtext
End Sub
```

See Also [FileDateTime](#), [FileLen](#), [GetAttr](#), [Lof](#)

Fix Function

Action	Returns the integer part of a number.
Syntax	Fix (<i>number</i>) where <i>number</i> is any valid numeric expression.
Comments	<p>The return value's data type matches the type of the numeric expression. This includes Variant expressions, unless the numeric expression is a string (vartype 8) that evaluates to a number, in which case the data type for its return value is vartype 5 (double). If the numeric expression is vartype 0 (empty), the data type for the return value is vartype 3 (long).</p> <p>For both positive and negative <i>numbers</i>, Fix removes the fractional part of the expression and returns the integer part only. For example, Fix (6.2) returns 6; Fix (-6.2) returns -6.</p>

Example This example returns the integer portion of a number provided by the user.

```
Sub main
  Dim usernum
  Dim intvalue
  usernum=InputBox("Enter a number with decimal places:")
  intvalue=Fix(usernum)
  MsgBox "The integer portion of " & usernum & " is: " & intvalue
End Sub
```

See Also [Abs](#), [Cint](#), [Exp](#), [Int](#), [Log](#), [Rnd](#), [Sgn](#), [Sqr](#)

For...Next Statement

Action Repeats a series of program lines a fixed number of times.

Syntax **For** *counter* = *start* **TO** *end* [**STEP** *increment*]
 [*statementblock*]
 [**Exit For**]
 [*statementblock*]
Next [*counter*]

where **is**
counter a numeric variable for the loop counter.
start the beginning value of the counter.
end the ending value of the counter.
increment the amount by which the counter is changed each time the loop is
 run. (The default is one.)
statementblock basic functions, statements, or methods to be executed.

Comments The *start* and *end* values must be consistent with *increment*: If *end* is greater than *start*, *increment* must be positive. If *end* is less than *start*, *increment* must be negative. SBL compares the sign of (*start end*) with the sign of *increment*. If the signs are the same, and *end* does not equal *start*, the **For...Next** loop is started. If not, the loop is omitted in its entirety.

With a **For...Next** loop, the program lines following the **For** statement are executed until the **Next** statement is encountered. At this point, the **Step** amount is added to the *counter* and compared with the final value, *end*. If the beginning and ending values are the same, the loop executes once, regardless of the **Step** value. Otherwise, the **Step** value controls the loop as follows:

Step Value	Loop Execution
Positive	If <i>counter</i> is less than or equal to <i>end</i> , the Step value is added to <i>counter</i> . Control returns to the statement after the For statement and the process repeats. If <i>counter</i> is greater than <i>end</i> , the loop is exited; execution resumes with the statement following the Next statement.
Negative	The loop repeats until <i>counter</i> is less than <i>end</i> .
Zero	The loop repeats indefinitely.

Within the loop, the value of the *counter* should not be changed, as changing the *counter* will make programs more difficult to maintain and debug.

For...Next loops can be nested within one another. Each nested loop should be given a unique variable name as its *counter*. The **Next** statement for the inside loop must appear before the **Next** statement for the outside loop. The **Exit For** statement may be used as an alternative exit from **For...Next** loops.

If the variable is left out of a **Next** statement, the **Next** statement will match the most recent **For** statement. If a **Next** statement occurs prior to its corresponding **For** statement, SBL will return an error message.

Multiple consecutive **Next** statements can be merged together. If this is done, the counters must appear with the innermost counter first and the outermost counter last. For example:

```

For i = 1 To 10
    [ statementblock ]
    For j = 1 To 5
        [ statementblock ]
    Next j, i

```

Example

This example calculates the factorial of a number. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

```

Sub main
    Dim number as Integer
    Dim factorial as Double
    Dim msgtext
    number=InputBox("Enter an integer between 1 and 170:")
    If number<=0 then
        Exit Sub
    End If
    factorial=1
    For x=number to 2 step -1
        factorial=factorial*x
    Next x
    Rem If number<= 35, then its factorial is small enough
    Rem to be stored as a single-precision number

```



```
If number<35 then
    factorial=CSng(factorial)
End If
msgtext="The factorial of " & number & " is: " & factorial
MsgBox msgtext
End Sub
```

See Also [Do...Loop, Exit, While...Wend](#)

Format Function

Action Returns a formatted string of an expression based on a given format.

Syntax **Format**[\$](*expression* [,*format*]) where *expression* is the value to be formatted. It may be a number, Variant, or string and *format* is a string expression representing the format to use. Select one of the topics below for a detailed description of format strings.

Comments **Format** formats the *expression* as a number, date, time, or string depending upon the *format* argument. The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string). As with any string, you must enclose the *format* argument in quotation marks ("").

Numeric values are formatted as either numbers or date/times. If a numeric expression is supplied and the *format* argument is omitted or null, the number will be converted to a string without any special formatting.

Both numeric values and Variants may be formatted as dates. When formatting numeric values as dates, the value is interpreted according the standard Basic date encoding scheme. The base date, December 30, 1899, is represented as zero, and other dates are represented as the number of days from the base date.

Strings are formatted by transferring one character at a time from the input *expression* to the output string.

Formatting Numbers
 Formatting Dates and Times
 Formatting Strings

Formatting Numbers The predefined numeric formats with their meanings are as follows:

Format	Description
General Number	Display the number without thousand separator.
Fixed	Display the number with at least one digit to the left and at least two digits to the right of the decimal separator.

Format	Description
Standard	Display the number with thousand separator and two digits to the right of decimal separator.
Scientific	Display the number using standard scientific notation.
Currency	Display the number using a currency symbol as defined in the International section of the Control Panel. Use thousand separator and display two digits to the right of decimal separator. Enclose negative value in parentheses.
Percent	Multiply the number by 100 and display with a percent sign appended to the right; display two digits to the right of decimal separator.
True/False	Display False for 0, True for any other number.
Yes/No	Display No for 0, Yes for any other number.
On/Off	Display Off for 0, On for any other number.

For a simple numeric format, use one or more digit characters and (optionally) a decimal separator. The two format digit characters provided are zero, “0”, and number sign, “#”. A zero forces a corresponding digit to appear in the output; while a number sign causes a digit to appear in the output if it is significant (in the middle of the number or non-zero).

Number	Fmt	Result
1234.56	#	1235
1234.56	###	1234.56
1234.56	##	1234.6
1234.56	#####	1234.56
1234.56	00000.000	01234.560
0.12345	###	.12
0.12345	0.###	0.12

A comma placed between digit characters in a format causes a comma to be placed between every three digits to the left of the decimal separator.

Number	Fmt	Result
1234567.8901	#,###	1,234,567.89
1234567.8901	#,#####	1,234,567.8901



Although a comma and period are used in the format to denote separators for thousands and decimals, the output string will contain the appropriate character, based upon the current international settings for your machine.

Numbers may be scaled either by inserting one or more commas before the decimal separator or by including a percent sign in the *format* specification. Each comma preceding the decimal separator (or after all digits if no decimal separator is supplied) will scale (divide) the number by 1000. The commas will not appear in the output string. The percent sign will cause the number to be multiplied by 100. The percent sign will appear in the output string in the same position as it appears in *format*.

Number	Fmt	Result
1234567.8901	#,##	1234.57
1234567.8901	#,.,####	1.2346
1234567.8901	#,.,##	1,234.57
0.1234	#0.00%	12.34%

Characters may be inserted into the output string by being included in the *format* specification. The following characters will be automatically inserted in the output string in a location matching their position in the *format* specification:

- + \$ () space : /

Any set of characters may be inserted by enclosing them in double quotes. Any single character may be inserted by preceding it with a backslash, “\”.

Number	Fmt	Result
1234567.89	\$#,0.00	\$1,234,567.89
1234567.89	"TOTAL:" \$#,#.00	TOTAL: \$1,234,567.89
1234	\=>#,\#< =	=>1,234<=

You may wish to use the SBL **'\$CSTRINGS** metacommand or the **Chr** function if you need to embed quotation marks in a format specification. The character code for a quotation mark is 34.

Numbers may be formatted in scientific notation by including one of the following exponent strings in the *format* specification:

E- E+ e- e+

The exponent string should be preceded by one or more digit characters. The number of digit characters following the exponent string determines the number of exponent digits in the output. *Format* specifications containing an uppercase E will result in an uppercase E in the output. Those containing a lowercase e will result in a lowercase e in the output. A minus sign following the E will cause negative exponents in the output to be preceded by a minus sign. A plus sign in the *format* will cause a sign to always precede the exponent in the output.

Number	Fmt	Result
1234567.89	###.##E-00	123.46E04
1234567.89	###.##e+#	123.46e+4
0.12345	0.00E-00	1.23E-01

A numeric *format* can have up to four sections, separated by semicolons. If you use only one section, it applies to all values. If you use two sections, the first section applies to positive values and zeros, the second to negative values. If you use three sections, the first applies to positive values, the second to negative values, and the third to zeros. If you include semicolons with nothing between them, the undefined section is printed using the format of the first section. The fourth section applies to Null values. If it is omitted and the input expression results in a NULL value, **Format** will return an empty string.

Number	Fmt	Result
1234567.89	#,0.00;(#,0.00);"Zero";"NA"	1,234,567.89
-1234567.89	#,0.00;(#,0.00);"Zero";"NA"	(1,234,567.89)
0.0	#,0.00;(#,0.00);"Zero";"NA#"	Zero
0.0	#,0.00;(#,0.00);;"NA"	0.00
Null	#,0.00;(#,0.00);"Zero";"NA"	NA
Null	"The value is: "	0.00

Formatting Dates and Times

As with numeric formats, there are several predefined formats for formatting dates and times:

Format	Description
General Date	If the number has both integer and real parts, display both date and time. (e.g., 11/8/93 1:23:45 PM); if the number has only integer part, display it as a date; if the number has only fractional part, display it as time.
Long Date	Display a Long Date. Long Date is defined in the International section of the Control Panel.

Format	Description
Medium Date	Display the date using the month abbreviation and without the day of the week. (e.g, 08-Nov-93).
Short Date	Display a Short Date. Short Date is defined in the International section of the Control Panel.
Long Time	Display Long Time. Long Time is defined in the International section of the Control Panel and includes hours, minutes, and seconds.
Medium Time	Do not display seconds; display hours in 12-hour format and use the AM/PM designator.
Short Time	Do not display seconds; use 24-hour format and no AM/PM designator.

When using a user-defined format for a date, the *format* specification contains a series of tokens. Each token is replaced in the output string by its appropriate value.

A complete date may be output using the following tokens:

Token	Output
c	The date time as if the <i>format</i> was “dddd tttt”. See the definitions below.
dddd	The date including the day, month, and year according to the machine’s current Short Date setting. The default Short Date setting for the United States is m/d/yy.
dddddd	The date including the day, month, and year according to the machine’s current Long Date setting. The default Long Date setting for the United States is mmmm dd, yyyy.
tttt	The time including the hour, minute, and second using the machine’s current time settings. The default time format is h:mm:ss AM/PM.

Finer control over the output is available by including *format* tokens that deal with the individual components of the date time. These tokens are:

Token	Output
d	The day of the month as a one or two digit number (1-31).
dd	The day of the month as a two digit number (01-31).
ddd	The day of the week as a three letter abbreviation (Sun-Sat).
dddd	The day of the week without abbreviation (Sunday-Saturday).

Token	Output
w	The day of the week as a number (Sunday as 1, Saturday as 7).
ww	The week of the year as a number (1-53).
m	The month of the year or the minute of the hour as a one or two digit number. The minute will be output if the preceding token was an hour; otherwise, the month will be output.
mm	The month or the year or the minute of the hour as a two digit number. The minute will be output if the preceding token was an hour; otherwise, the month will be output.
mmm	The month of the year as a three letter abbreviation (Jan-Dec).
mmmm	The month of the year without abbreviation(January-December).
q	The quarter of the year as a number (1-4).
y	The day of the year as a number (1-366).
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (100-9999).
h	The hour as a one or two digit number (0-23).
hh	The hour as a two digit number (00-23).

Token	Output
n	The minute as a one or two digit number (0-59).
nn	The minute as a two digit number (00-59).
s	The second as a one or two digit number (0-59).
ss	The second as a two digit number (00-59).

By default, times will be displayed using a military (24-hour) clock. Several tokens are provided in date time *format* specifications to change this default. They all cause a 12 hour clock to be used. These are:

Token	Output
AM/PM	An uppercase AM with any hour before noon; an uppercase PM with any hour between noon and 11:59 PM.
am/pm	A lowercase am with any hour before noon; a lowercase pm with any hour between noon and 11:59 PM.
A/P	An uppercase A with any hour before noon; an uppercase P with any hour between noon and 11:59 PM.

Formatting Strings

Token	Output
a/p	A lowercase a with any hour before noon; a lowercase p with any hour between noon and 11:59 PM.
AMPM	The contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. Note, ampm is equivalent to AMPM.

Any set of characters may be inserted into the output by enclosing them in double quotes. Any single character may be inserted by preceding it with a backslash, “\”. See number formatting above for more details.

By default, string formatting transfers characters from left to right. The exclamation point, “!”, when added to the *format* specification causes characters to be transferred from right to left.

By default, characters being transferred will not be modified. The less than, “<”, and the greater than, “>”, characters may be used to force case conversion on the transferred characters. Less than forces output characters to be in lowercase. Greater than forces output characters to be in uppercase.

Character transfer is controlled by the at sign, “@”, and ampersand, “&”, characters in the *format* specification. These operate as follows:

Character	Interpretation
@	Output a character or a space. If there is a character in the string being formatted in the position where the @ appears in the format string, display it; otherwise, display a space in that position.
&	Output a character or nothing. If there is a character in the string being formatted in the position where the & appears, display it; otherwise, display nothing.

A *format* specification for strings can have one or two sections separated by a semicolon. If you use one section, the format applies to all string data. If you use two sections, the first section applies to string data, the second to Null values and zero-length strings.

Example

This example calculates the square root of 2 as a double-precision floating point value and displays it in scientific notation.

```
Sub main
    Dim value
    Dim msgtext
    value=Cdbl(Sqr(2))
    msgtext= "The square root of 2 is: " & Format(Value,"Scientific")
    MsgBox msgtext
End Sub
```

See Also

[Asc](#), [Ccur](#), [Cdbl](#), [Chr](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#), [Str](#)

FreeFile Function

Action	Returns the lowest unused file number.
Syntax	FreeFile
Comments	<p>The FreeFile function is used when you need to supply a file number and want to make sure that you are not choosing a file number which is already in use.</p> <p>The value returned can be used in a subsequent Open statement.</p>
Example	This example opens a file and assigns to it the next file number available.

```

Sub main
    Dim filenumber
    Dim filename as String
    filenumber=FreeFile
    filename=InputBox("Enter a file to open: ")
    On Error Resume Next
    Open filename For Input As filenumber
    If Err<>0 then
        MsgBox "Error loading file. Re-run program."
        Exit Sub
    End If
    MsgBox "File " & filename & " opened as number: " & filenumber
    Close #filenumber
    MsgBox "File now closed."
End Sub

```

See Also [Open](#)

Function ... End Function Statement

Action	Defines a function procedure.
Syntax	<p>[Static] [Private] Function <i>name</i> [([Optional] <i>parameter</i> [<i>As type</i>] ...)] [<i>As functype</i>]</p> <p><i>name</i>= <i>expression</i></p> <p>End Function</p>
where	is
<i>name</i>	a function name.
<i>parameter</i>	the argument(s) to pass to the function when it is called.
<i>type</i>	the data type for the function arguments.
<i>functype</i>	the data type for the return value.
<i>name=expression</i>	the expression that sets the return value for the function.

Comments

The purpose of a function is to produce and return a single value of a specified type. Recursion is supported.

The data type of *name* determines the type of the return value. Use a type character as part of the *name*, or use the **As** *functype* clause to specify the data type. If omitted, the default data type is **Variant**. When calling the function, you need not specify the type character.

The *parameters* are specified as a comma-separated list of variable names. The data type of a parameter may be specified by using a type character or by using the **As** clause. Record parameters are declared using an **As** clause and a *type* which has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the **Function** statement. All references to an array parameter within the body of the function must have a consistent number of dimensions.

You specify the return value for the function name using the *name=expression* assignment, where *name* is the name of the function and *expression* evaluates to a return value. If omitted, the value returned is 0 for numeric functions and an empty string ("") for string functions and *vartype* 0 (Empty) is returned for a return type of Variant. The function returns to the caller when the **End Function** statement is reached or when an **Exit Function** statement is executed.

If you declare a parameter as **Optional**, a procedure may omit its value when calling the function. Only parameters with **Variant** data types may be declared as optional, and all optional arguments must appear after all required arguments in the **Function** statement.

The **Static** keyword specifies that all the variables declared within the function will retain their values as long as the program is running, regardless of the way the variables are declared.

The **Private** keyword specifies that the function will not be accessible to functions and subprograms from other modules. Only procedures defined in the same module will have access to a **Private** function.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it will modify the variable passed by the caller. This feature should be used with great care.

Use **Sub** to define a procedure with no return value.

Example

This example declares a function that is later called by the main subprogram. The function does nothing but set its return value to 1.

```
Declare Function SBL_exfunction()  
Sub main  
  Dim y as Integer  
  Call SBL_exfunction
```

```

y=SBL_exfunction
MsgBox "The value returned by the function is: " & y
End Sub

```

```

Function SBL_exfunction()
    SBL_exfunction=1
End Function

```

See Also [Call, Dim, Global, Option Explicit, Static, Sub...End Sub](#)

FV Function

Action Returns the future value for a constant periodic stream of cash flows as in an annuity or a loan.

Syntax **FV** (*rate*, *nper*, *pmt*, *pv*, *due*)

where	is
<i>rate</i>	interest rate per period.
<i>nper</i>	total number of payment periods.
<i>pmt</i>	constant periodic payment per period.
<i>pv</i>	present value or the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
<i>due</i>	an integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments The given interest rate is assumed constant over the life of the annuity.

If payments are on a monthly schedule and the annual percentage rate on the annuity or loan is 9%, the *rate* is 0.0075 (.0075=.09/12).

Example This example finds the future value of an annuity, based on terms specified by the user.

```

Sub main
    Dim aprate, periods
    Dim payment, annuitypv
    Dim due, futurevalue
    Dim msgtext
    annuitypv=InputBox("Enter present value of the annuity: ")
    aprate=InputBox("Enter the annual percentage rate: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    periods=InputBox("Enter the total number of pay periods: ")
    payment=InputBox("Enter the initial amount paid to you: ")
    Rem Assume payments are made at end of month
    due=0
    futurevalue=FV(aprate/12,periods,-payment,-annuitypv,due)
    msgtext= "The future value is: " & Format(futurevalue, "Currency")
    MsgBox msgtext
End Sub

```

See Also [Ipmt, IRR, NPV, Pmt, Ppmt, PV, Rate](#)

Get Statement

Action	Reads data from a file opened in Random or Binary mode and puts it in a variable.
Syntax	<div><div>Get [#] <i>filenumber%</i>, [<i>recnumber&</i>], <i>varname</i></div><div><div>where <i>filenumber%</i> <i>recnumber&</i> <i>varname</i></div><div>is an integer expression identifying the open file to use. a Long expression containing the number of the record (for Random mode) or the offset of the byte (for Binary mode) at which to start reading. the name of the variable into which Get reads file data. <i>Varname</i> can be any variable except Object or Array variables (single array elements may be used).</div></div></div>
Comments	<div>For more information about how files are numbered when they're opened, see the Open statement.</div> <div><i>Recnumber&</i> is in the range 1 to 2,147,483,647. If omitted, the next record or byte is read.</div>
+	<div><i>The commas before and after the recnumber& are required, even if you do not supply a recnumber&.</i></div> <div>For Random mode, the following rules apply:<ul style="list-style-type: none">■ Blocks of data are read from the file in chunks whose size is equal to the size specified in the Len clause of the Open statement. If the size of <i>varname</i> is smaller than the record length, the additional data is discarded. If the size of <i>varname</i> is larger than the record length, an error occurs.■ For variable length String variables, Get reads two bytes of data that indicate the length of the string, then reads the data into <i>varname</i>.■ For Variant variables, Get reads two bytes of data that indicate the type of the Variant, then it reads the body of the Variant into <i>varname</i>. Note that Variants containing strings contain two bytes of data type information followed by two bytes of length followed by the body of the string.■ User defined types are read as if each member were read separately, except no padding occurs between elements.</div>

Files opened in **Binary** mode behave similarly to those opened in **Random** mode, except:

- **Get** reads variables from the disk without record padding.
- Variable length **Strings** that are not part of user defined types are not preceded by the two-byte string length. Instead, the number of bytes read is equal to the length of *varname*.

Example

This example opens a file for Random access, gets its contents, and closes the file again. The second subprogram, CREATEFILE, creates the C:\TEMP001 file used by the main subprogram. Declare Sub createfile()

```
Sub main
  Dim acctno as String*3
  Dim recno as Long
  Dim msgtext as String
  Call createfile
  recno=1
  newline=Chr(10)
  Open "C:\TEMP001" For Random As #1 Len=3
  msgtext="The account numbers are:" & newline
  Do Until recno=11
    Get #1,recno,acctno
    msgtext=msgtext & acctno
    recno=recno+1
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub

Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
  Close #1
End Sub
```

See Also [Open, Put, Type](#)

GetAttr Function

Action Returns the attributes of a file, directory or volume label.

Syntax **GetAttr**(*pathname\$*) where *pathname\$* is a String expression for the name of the file, directory, or label to query.

Comments *Pathname\$* may not contain wildcards (* and ?).

The file attributes returned by **GetAttr** are as follows:

Value	Meaning
0	Normal file
1	Read-only file
2	Hidden file
4	System file
8	Volume label
16	Directory
32	Archive - file has changed since last backup

Example This example tests the attributes for a file and if it is hidden, changes it to a non-hidden file.

```

Sub main
  Dim filename as String
  Dim attribs, saveattribs as Integer
  Dim answer as Integer
  Dim archno as Integer
  Dim msgtext as String
  archno=32
  On Error Resume Next
  msgtext="Enter name of a file:"
  filename=InputBox(msgtext)
  attribs=GetAttr(filename)
  If Err<>0 then
    MsgBox "Error in filename. Re-run Program."
    Exit Sub
  End If
  saveattribs=attribs
  If attribs>= archno then
    attribs=attribs-archno
  End If
  Select Case attribs
    Case 2,3,6,7
      msgtext=" File: " & filename & " is hidden." & Chr(10)
      msgtext=msgtext & Chr(10) & " Change it?"
      answer=Msgbox(msgtext,308)
      If answer=6 then
        SetAttr filename, saveattribs-2
        MsgBox "File is no longer hidden."
        Exit Sub
      End If
      MsgBox "Hidden file not changed."
    Case Else
      MsgBox "File was not hidden."
  End Select
End Sub

```

See Also [FileAttr](#), [SetAttr](#)

GetCurValues Statement

Action	Stores the current values for the dialog box associated with the specified record.
Syntax	GetCurValues <i>recordName</i> where <i>recordName</i> is a variable dimensioned as a dialog box record.
Comments	A dialog box record is defined using the Begin Dialog statement. <i>recordName</i> is a variable dimensioned as follows:

Dim *recordName* as UserDialog

where UserDialog is the dialog box name used in **Begin Dialog**.

Example This example stores the values for the dialog box MYDIALOGBOX.

```
Sub main
  Dim cchoices as String
  On Error Resume Next
  cchoices="All"+Chr$(9)+"Nothing"
  Begin Dialog UserDialog 180, 95, "SBL Dialog Box"
    ButtonGroup .ButtonGroup1
    Text 9, 3, 69, 13, "Filename:", .Text1
    ComboBox 9, 17, 111, 41, cchoices, .ComboBox1
    OkButton 131, 8, 42, 13
    CancelButton 131, 27, 42, 13
    PushButton 132, 48, 42, 13, "Help", .Push1
  End Dialog
  Dim mydialogbox As UserDialog
  Dialog mydialogbox
  If Err=102 then
    MsgBox "You pressed Cancel."
  End If
  GetCurValues mydialogbox
End Sub
```

See Also [Dim](#), [Begin Dialog](#), [Dialog Function](#), [Dialog Statement](#)

GetField Function [SBL Extension]**

Action	Returns a substring from a source string.
Syntax	GetField[\$](<i>string</i> \$, <i>field_number</i> %, <i>separator_chars</i> \$)
where	is
<i>string</i> \$	a list of fields, divided by separator characters.
<i>field_number</i> %	the number of the field to return, starting with 1.
<i>separator_chars</i> \$	the characters separating each field.

Comments	<p>Multiple separator characters may be specified. If <i>field_number</i> is greater than the number of fields in the string, an empty string (") is returned.</p> <p>**SBL offers a number of extensions that are not included in Visual Basic.</p>
Example	<p>This example finds the third value in a string, delimited by plus signs (+).</p> <pre> Sub main Dim teststring,retvalue Dim msgtext teststring="9+8+7+6+5" retvalue=GetObject(teststring,3,"+") MsgBox "The third field in: " & teststring & " is: " & retvalue End Sub </pre>
See Also	Left , Ltrim , Mid Function , Mid Statement , Right , Rtrim , SetField , StrComp , Trim

GetObject Function

Action	Returns an OLE2 object associated with the file name or the application name.
Syntax A	GetObject (<i>pathname</i>)
Syntax B	GetObject (<i>pathname</i> , <i>class</i>)
Syntax C	GetObject (, <i>class</i>) where <i>pathname</i> is the path and file name for the object to retrieve and <i>class</i> is a string containing the class of the object.
Comments	<p>Use GetObject with the Set statement to assign a variable to the object for use in a Basic procedure. The variable used must first be dimensioned as an Objectobjectclass.</p> <p>Syntax A of GetObject accesses an OLE2 object stored in a file. For example, the following two lines dimension the variable, FILEOBJECT as an Object and assign the object file "PAYABLES" to it. PAYABLES is located in the subdirectory SPREDSHT:</p> <pre> Dim FileObject As Object Set FileObject = GetObject("spredsht\payables") </pre> <p>If the application supports accessing component OLE2 objects within the file, you may append an exclamation point and a component object name to the file name, as follows:</p> <pre> Dim ComponentObject As Object Set ComponentObject = GetObject("spredsht\payables!R1C1:R13C9") </pre>

Syntax B of **GetObject** accesses an OLE2 object of a particular class that is stored in a file. *Class* uses the syntax “*appname.objtype*”, where *appname* is the name of the application that provides the object, and *objtype* is the type or class of the object. For example:

```
Dim ClassObject As Object
Set ClassObject = GetObject("\\spredsht\payables",
"\\turbosht.spreadsheet")
```

The third form of **GetObject** accesses the active OLE2 object of a particular class. For example:

```
Dim ActiveSheet As Object
SetActiveSheet = GetObject( , "\\turbosht.spreadsheet")
```

Example

This example displays a list of open files in the software application, VISIO. It uses the **GetObject** function to access VISIO. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

    'Initialize Visio
    Set visio = GetObject("visio.application")    ' find Visio
    If (visio Is Nothing) then
        MsgBox "Couldn't find Visio!"
    Exit Sub
    End If
    'Get # of open Visio files
    doccount = visio.documents.count 'OLE2 call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)    ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also [CreateObject](#), [Is](#), [Me](#), [New](#), [Nothing](#), [Object Class](#), [Typeof](#)

Global Statement

Action Declare Global variables for use in a Basic program.

Syntax **Global** *variableName* [**As** *type*] [,*variableName* [**As** *type*]] ... where *variableName* is a variable name and *type* is the data type for a variable.

Comments Global data is shared across all loaded modules. If an attempt is made to load a module which has a global variable declared which has a different data type than an existing global variable of the same name, the module load will fail.

Basic is a strongly typed language: all variables must be given a data type or they will be automatically assigned a type of **Variant**.

If the **As** clause is not used, the type of the global variable may be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single **Global** statement (although not on the same variable).

Regardless of which mechanism you use to declare a global variable, you may choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

The following data types are available:

- Arrays
- Numbers
- Records
- Strings
- Variants

Arrays The available data types for arrays are: numbers, strings, Variants and records. Arrays of arrays, dialog box records, and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

Global *variable*([*subscriptRange*, ...]) [**As** *typeName*]

where *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

Both the *startSubscript* and the *endSubscript* are valid subscripts for the array. The maximum number of subscripts which may be specified in an array definition is 60.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the **ReDim** statement must be used to specify the dimensions of the array before the array can be used.

Numbers Numeric variables can be declared using the **As** clause and one of the following numeric types: **Currency**, **Integer**, **Long**, **Single**, **Double**. Numeric variables can also be declared by including a type character as a suffix to the name.

Records Record variables are declared by using an **As** clause and a *type* which has previously been defined using the **Type** statement. The syntax to use is:

Global *variableName* **As** *typeName*

Records are made up of a collection of data elements called fields. These fields may be of any numeric, string, Variant or previously defined record type. See **Type** for details on accessing fields within a record.

You cannot use the **Global** statement to declare a dialog record.

Strings SBL supports two types of strings, fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

Global *variableName* **As String***length

Dynamic strings have no declared length, and can vary in length from 0 to 32767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

Global *variableName*\$

or **Global** *variableName* **As String**

Variants Declare variables as Variants when the type of the variable is not known at the start of, or may change during, the procedure. For example, a Variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a Variant:

Global *variableName*

or **Global***variableName* **As Variant**

Variant variables are initialized to vartype Empty.

Example This example contains two subroutines that share the variables TOTAL and ACCTNO, and the record GRECORD.

```
Type acctrecord
  acctno As Integer
End Type
```

```

Global acctno as Integer
Global total as Integer
Global grecord as acctrecord
Declare Sub createfile

Sub main
    Dim msgtext
    Dim newline as String
    newline=Chr$(10)
    Call createfile
    Open "C:\TEMP001" For Input as #1
    msgtext="The new account numbers are " & newline
    For x=1 to total
        Input #1, grecord.acctno
        msgtext=msgtext & newline & grecord.acctno
    Next x
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile
    Dim x
    x=1
    grecord.acctno=1
    Open "C:\TEMP001" For Output as #1
    Do While grecord.acctno<>0
        grecord.acctno=InputBox("Enter 0 or new account #" & x & ":")
        If grecord.acctno<>0 then
            Print #1, grecord.acctno
            x=x+1
        End If
    Loop
    total=x-1
    Close #1
End Sub

```

See Also [Const, Dim, Option Base, ReDim, Static, Type](#)

GoTo Statement

Action	Transfers program control to the label specified.
Syntax	GoTo { <i>label</i> <i>line</i> } where <i>label</i> is a name beginning in the first column of a line of code and ending with a colon (:) and <i>line</i> is the line number of a program line.
Comments	<p>A <i>label</i> has the same format as any other Basic name. Reserved words are not valid labels. Program lines are numbered automatically, beginning with 1.</p> <p>GoTo cannot be used to transfer control out of the current Function or Subprogram.</p>

Example This example displays the date for one week from the date entered by the user. If the date is invalid, the Goto statement sends program execution back to the beginning.

```
Sub main
  Dim str1 as String
  Dim nextweek
  Dim msgtext
i: str1=InputBox$("Enter a date:")
  answer=IsDate(str1)
  If answer=-1 then
    str1=CvDate(str1)
    nextweek=DateValue(str1)+7
    msgtext="One week from the date entered is:"
    msgtext=msgtext & Format(nextweek,"dddddd")
    MsgBox msgtext
  Else
    MsgBox "Invalid date or format. Try again."
    Goto i
  End If
End Sub
```

See Also [Do...Loop, For...Next, If...Then...Else, Select Case, While...Wend](#)

GroupBox Statement

Action Defines a box to enclose sets of dialog box items, such as option boxes and checkboxes.

Syntax **GroupBox** *x, y, dx, dy, text\$ [, .id]*

where	is
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.
<i>text\$</i>	a string containing the title for the top border of the group box.
<i>.id</i>	the optional string ID for the groupbox, used by the dialog statements that act on this control.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

If *text\$* is wider than *dx*, the additional characters are truncated. If *text\$* is an empty string (""), the top border of the group box will be a solid line.

Use the **GroupBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example creates a dialog box with a group box, and two buttons.

```
Sub main
Begin Dialog UserDialog 194, 76, "SBL Dialog Box"
  GroupBox 9, 8, 97, 57, "File Range"
  OptionGroup .OptionGroup2
    OptionButton 19, 16, 46, 12, "All pages", .OptionButton3
    OptionButton 19, 32, 67, 8, "Range of pages", .OptionButton4
  Text 25, 43, 20, 10, "From:", .Text6
  Text 63, 43, 14, 9, "To:", .Text7
  TextBox 79, 43, 13, 12, .TextBox4
  TextBox 47, 43, 12, 11, .TextBox5
  OkButton 135, 6, 54, 14
  CancelButton 135, 26, 54, 14
End Dialog
Dim mydialog as UserDialog
On Error Resume Next
Dialog mydialog
If Err=102 then
  MsgBox "Dialog box canceled."
End If
End Sub
```

See Also [Begin Dialog...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Hex Function

Action	Returns the hexadecimal representation of a number, as a string.
Syntax	Hex [\$](<i>number</i>) where <i>number</i> is any numeric expression that evaluates to a number.
Comments	<p>If <i>number</i> is an integer, the return string contains up to four hexadecimal digits; otherwise, the value will be converted to a Long Integer, and the string may contain up to 8 hexadecimal digits.</p> <p>To represent a hexadecimal number directly, precede the hexadecimal value with &H. For example, &H10 equals decimal 16 in hexadecimal notation.</p> <p>The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will return a Variant of vartype 8 (string).</p>
Example	<p>This example returns the hex value for a number entered by the user.</p> <pre>Sub main Dim usernum as Integer Dim hexvalue usernum=InputBox("Enter a number to convert to hexadecimal:") hexvalue=Hex(usernum) MsgBox "The HEX value is: " & hexvalue End Sub</pre>
See Also	Oct

Hour Function

Action Returns the hour of day component (0-23) of a date-time value.

Syntax **Hour**(*time*) where *time* is any numeric or string expression that can evaluate to a date and time.

Comments **Hour** accepts any type of *time* including strings and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (integer). If the value of *time* is Null, a Variant of vartype 1 (null) is returned.

Time is a double-precision value. The numbers to the left of the decimal point denote the date and the decimal value denotes the time (from 0 to .99999). Use the **TimeValue** function to obtain the correct value for a specific time.

Example This example extracts just the time (hour, minute, and second) from a file's last modification date and time.

```
Sub main
    Dim filename as String
    Dim ftime
    Dim hr, min
    Dim sec
    Dim msgtext as String
i: msgtext="Enter a filename:"
    filename=InputBox(msgtext)
    If filename="" then
        Exit Sub
    End If
    On Error Resume Next
    ftime=FileDateTime(filename)
    If Err<>0 then
        MsgBox "Error in file name. Try again."
        Goto i:
    End If

    hr=Hour(ftime)
    min=Minute(ftime)
    sec=Second(ftime)
    MsgBox "The file's time is: " & hr & ":" & min & ":" & sec
End Sub
```

See Also [Day](#), [Minute](#), [Month](#), [Now](#), [Second](#), [Time Function](#), [Time Statement](#), [Weekday](#), [Year](#)

If ... Then ... Else

Action	Executes alternative blocks of program code based on one or more expressions.
Syntax A	If <i>condition</i> Then <i>then_statement</i> [Else <i>else_statement</i>]
Syntax B	If <i>condition</i> Then <i>statement_block</i> [ElseIf <i>expression</i> Then <i>statement_block</i>] ... [Else <i>statement_block</i>] End If where is <i>condition</i> any expression that evaluates to TRUE (non-zero) or FALSE (zero). <i>then_statement</i> any valid single expression. <i>else_statement</i> any valid single expression. <i>expression</i> any expression that evaluates to TRUE (non-zero) or FALSE (zero). <i>statement_block</i> 0 or more valid expressions, separated by colons (:), or on different lines.
Comments	When multiple statements are required in either the Then or Else clauses, use the block version (Syntax B) of the If statement.
Example	<p>This example tests the attributes for a file and if it is hidden, changes it to a non-hidden file.</p> <pre> Sub main Dim filename Dim attribs, msgtext Dim answer, archno Dim saveattribs On Error Resume Next archno=32 filename=InputBox("Enter name of a file:") attribs=GetAttr(filename) If attribs=0 then MsgBox "Error in file name. Rerun program." Exit Sub End If saveattribs=attribs If attribs>= archno then attribs=attribs-archno End If Select Case attribs Case 2,3,6,7 msgtext="File:" &filename & "is hidden. Change it? (Y/N)" answer=InputBox\$(msgtext) If answer="Y" then </pre>

```

        SetAttr filename, saveattribs-2
        MsgBox "File is no longer hidden."
    End If
Case Else
    MsgBox "File was not hidden."
End Select
End Sub

```

See Also [Do...Loop, For...Next, Goto, On...Goto, Select Case, While...Wend](#)

'\$Include Metacommand [SBL Extension]** '

Action	Includes statements from the specified file.
Syntax	'\$Include: " <i>filename</i> " where <i>filename</i> is the name and location of the file to include.
Comments	<p>It is recommended (although not required) that you use a file extension of .SBH for <i>filename</i>.</p> <p>All metacommands must begin with an apostrophe (') and are recognized by the compiler only if the command starts at the beginning of a line. For compatibility with other versions of Basic, you may enclose the <i>filename</i> in single quotation marks (').</p> <p>If no directory or drive is specified, the compiler will search for <i>filename</i> on the source file search path.</p>

**SBL offers a number of extensions that are not included in Visual Basic.

Example This example includes a file containing the list of global variables, called GLOBALS.SBH. For this example to work correctly, you must create the GLOBALS.SBH file with at least the following statement: Dim gtext as String. The Option Explicit statement is included in this example to prevent SBL from automatically dimensioning the variable as a Variant.

```

Option Explicit
Sub main
    Dim msgtext as String
    '$Include: "c:\globals.sbh"
    gtext=InputBox("Enter a string for the global variable:")
    msgtext="The variable for the string "
    msgtext=msgtext & gtext & " was DIM'ed in GLOBALS.SBH."
    MsgBox msgtext
End Sub

```

See Also [\\$Cstrings, \\$NoCStrings, Rem](#)

Input Function

Action	Returns a string containing the characters read.
Syntax	Input [\$](<i>number%</i> , [#] <i>filenumber%</i>) where <i>number%</i> is the number of characters (bytes) to read from the file and <i>filenumber%</i> is an integer expression identifying the open file to use.
Comments	<p>The file pointer is advanced the number of characters read. Unlike the Input statement, Input returns all characters it reads, including carriage returns, line feeds, and leading spaces.</p> <p>The dollar sign, “\$”, in the function name is optional. If specified the return type is string. If omitted the function will return a Variant of vartype 8 (string).</p>
Example	<p>This example opens a file and prints its contents to the screen.</p> <pre> Sub main Dim fname Dim fchar() Dim x as Integer Dim msgtext Dim newline newline=Chr(10) On Error Resume Next fname=InputBox("Enter a filename to print:") If fname="" then Exit Sub End If Open fname for Input as #1 If Err<>0 then MsgBox "Error loading file. Re-run program." Exit Sub End If msgtext="The contents of " & fname & " is: " & newline & newline Redim fchar(Lof(1)) For x=1 to Lof(1) fchar(x)=Input(1,#1) msgtext=msgtext & fchar(x) Next x MsgBox msgtext Close #1 End Sub </pre>

See Also [Get, Input Statement, Line Input, Open, Write](#)

Input Statement

Action	Reads data from a sequential file and assigns the data to variables.
Syntax A	Input [#] <i>filename</i> %, variable [, variable]...
Syntax B	Input [prompt\$,] variable [, variable]...
	<p>where is</p> <p><i>filename</i>% an integer expression identifying the open file to read from</p> <p><i>variable</i> the variable(s) to contain the value(s) read from the file.</p> <p><i>prompt</i>\$ an optional string that prompts for keyboard input.</p>
Comments	<p>The <i>filename</i>% is the number used in the Open statement to open the file. The list of <i>variables</i> is separated by commas.</p> <p>If <i>filename</i>% is not specified, the user is prompted for keyboard input, either with <i>prompt</i>\$ or with a "?", if <i>prompt</i>\$ is omitted.</p>
Example	<p>This example prompts a user for an account number, opens a file, searches for the account number and displays the matching letter for that number. It uses the Input statement to increase the value of x and at the same time get the letter associated with each value. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.</p>

```

Declare Sub createfile()
Global x as Integer
Global y(100) as String
Sub main
    Dim acctno as Integer
    Dim msgtext
    Call createfile
i: acctno=InputBox("Enter an account number from 1-10:")
    If acctno<1 Or acctno>10 then
        MsgBox "Invalid account number. Try again."
        Goto i:
    End if
    x=1
    Open "C:\TEMP001" for Input as #1
    Do Until x=acctno
        Input #1, x,y(x)
    Loop
        msgtext="The letter for account number " & x & " is: " & y(x)
    Close #1
    MsgBox msgtext
    Kill "C:\TEMP001"
End Sub

Sub createfile()
' Put the numbers 1-10 and letters A-J into a file
    Dim startletter
    Open "C:\TEMP001" for Output as #1
    startletter=65
    For x=1 to 10

```

```
y(x)=Chr(startletter)
startletter=startletter+1
Next x
For x=1 to 10
  Write #1, x,y(x)
Next x
Close #1
End Sub
```

See Also [Get, Input Function, Line Input, Open, Write](#)

InputBox Function

Action Displays a dialog box containing a prompt and returns a string entered by the user.

Syntax **InputBox**[\$](*prompt\$* , [*title\$*] , [*default\$*] ,[*xpos%* , *ypos%*])

The dollar sign, “\$”, in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

- | | |
|-----------------------------|--|
| where | is |
| <i>prompt\$</i> | a string expression containing the text to show in the dialog box. |
| <i>title\$</i> | the caption to display in the dialog box’s title bar. |
| <i>default\$</i> | the string expression to display in the edit box as the default response. |
| <i>xpos%</i> , <i>ypos%</i> | numeric expressions, specified in dialog box units, that determine the position of the dialog box. |

Comments The length of *prompt\$* is restricted to 255 characters. This figure is approximate and depends on the width of the characters used. Note that a carriage return and a line-feed character must be included in *prompt\$* if a multiple-line prompt is used.

If either *prompt\$* or *default\$* is omitted, nothing is displayed.

Xpos% determines the horizontal distance between the left edge of the screen and the left border of the dialog box. *Ypos%* determines the horizontal distance from the top of the screen to the dialog box’s upper edge. If these arguments are not entered, the dialog box is centered roughly one third of the way down the screen. A horizontal dialog box unit is 1/4 of the average character width in the system font; a vertical dialog box unit is 1/8 of the height of a character in the system font.

+ *To specify the dialog box’s position, you must enter both of these arguments. If you enter one without the other, the default positioning is set.*

If the user presses Enter, or selects the OK button, **InputBox** returns the text contained in the input box. If the user selects Cancel, the **InputBox** function returns a null string ("").

Example This example uses `InputBox` to prompt for a filename and then prints the filename using `MsgBox`.

```
Sub main
    Dim filename
    Dim msgtext
    msgtext="Enter a filename:"
    filename=InputBox$(msgtext)
    MsgBox "The file name you entered is: " & filename
End Sub
```

See Also [Input Function](#), [Input Statement](#), [MsgBox Function](#), [MsgBox Statement](#), [PasswordBox](#)

InStr Function

Action Returns the position of the first occurrence of one string within another string.

Syntax A `InStr([start%,] string1$, string2$)`

Syntax B `InStr(start, string1$, string2$[, compare])`

where	is
<i>start%</i>	the position in <i>string1\$</i> to begin the search. (1=first character in string.)
<i>string1\$</i>	the string to search.
<i>string2\$</i>	the string to find.
<i>compare</i>	an integer expression for the method to use to compare the strings. (0=case-sensitive, 1=case-insensitive.)

Comments If not specified, the search starts at the beginning of the string (equivalent to a *start%* of 1). These arguments may be of any type. They will be converted to strings.

InStr returns a zero under the following conditions:

- 1 *start%* is greater than the length of *string2\$*.
- 2 *string1\$* is a null string.
- 3 *string2\$* is not found.

If either *string1\$* or *string2\$* is a null Variant, **InStr** returns a null Variant.

If *string2\$* is a null string (""), **InStr** returns the value of *start%*.

If *compare* is 0, a case-sensitive comparison based on the ANSI character set sequence is performed. If *compare* is 1, a case-insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system. If *compare* is omitted, the module level default, as specified with **Option Compare**, is used.

Example This example generates a random string of characters then uses InStr to find the position of a single character within that string.

```
Sub main
  Dim x as Integer
  Dim y
  Dim str1 as String
  Dim str2 as String
  Dim letter as String
  Dim randomvalue
  Dim upper, lower
  Dim position as Integer
  Dim msgtext, newline
  upper=Asc("z")
  lower=Asc("a")
  newline=Chr(10)
  For x=1 to 26
    Randomize
    randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
    letter=Chr(randomvalue)
    str1=str1 & letter
  'Need to waste time here for fast processors
  For y=1 to 1000
    Next y
  Next x
  str2=InputBox("Enter a letter to find")
  position=InStr(str1,str2)
  If position then
    msgtext="The position of " & str2 & " is: " & position & newline
    msgtext=msgtext & "in string: " & str1
  Else
    msgtext="The letter: " & str2 & " was not found in: " & newline
    msgtext=msgtext & str1
  End If
  MsgBox msgtext
End Sub
```

See Also [GetField](#), [Left](#), [Mid Function](#), [Mid Statement](#), [Option](#), [Right](#), [Str](#), [StrComp](#)

Int Function

Action	Returns the integer part of a <i>number</i> .
Syntax	Int (<i>number</i>) where <i>number</i> is any numeric expression.
Comments	<p>For positive <i>numbers</i>, Int removes the fractional part of the expression and returns the integer part only. For negative <i>numbers</i>, Int returns the largest integer less than or equal to the expression. For example, Int (6.2) returns 6; Int(-6.2) returns -7.</p> <p>The return type matches the type of the numeric expression. This includes Variant expressions which will return a result of the same vartype as input except vartype 8 (string) will be returned as vartype 5 (double) and vartype 0 (empty) will be returned as vartype 3 (long).</p>

Example This example uses `Int` to generate random numbers in the range between the ASCII values for lowercase a and z (97 and 122). The values are converted to letters and displayed as a string.

```
Sub main
    Dim x as Integer
    Dim y
    Dim str1 as String
    Dim letter as String
    Dim randomvalue
    Dim upper, lower
    Dim msgtext, newline
    upper=Asc("z")
    lower=Asc("a")
    newline=Chr(10)
    For x=1 to 26
        Randomize
        randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
        letter=Chr(randomvalue)
        str1=str1 & letter
        'Need to waste time here for fast processors
        For y=1 to 1500
            Next y
        Next x

        msgtext="The string is:" & newline
        msgtext=msgtext & str1
        MsgBox msgtext
    End Sub
```

See Also [Exp](#), [FixInt](#), [Log](#), [Rnd](#), [Sgn](#), [Sqr](#)

IPmt Function

Action Returns the interest portion of a payment for a given period of an annuity.

Syntax **IPmt**(*rate*, *per*, *nper*, *pv*, *fv*, *due*)

where	is
<i>rate</i>	interest rate per period.
<i>per</i>	particular payment period in the range 1 through <i>nper</i> .
<i>nper</i>	total number of payment periods.
<i>pv</i>	present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
<i>fv</i>	future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
<i>due</i>	0 if payments are due at the end of each payment period, and 1 if they are due at the beginning of the period.

Comments The given interest rate is assumed constant over the life of the annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the interest portion of a loan payment amount for payments made in last month of the first year. The loan is for \$25,000 to be paid back over 5 years at 9.5% interest.

```
Sub main
  Dim aprate, periods
  Dim payperiod
  Dim loanpv, due
  Dim loanfv, intpaid
  Dim msgtext
  aprate=.095
  payperiod=12
  periods=120
  loanpv=25000
  loanfv=0
  Rem Assume payments are made at end of month
  due=0
  intpaid=IPmt(aprate/12,payperiod,periods,-loanpv,loanfv,due)
  msgtext="For a loan of $25,000 @ 9.5% for 10 years," & Chr(10)
  msgtext=msgtext+ "the interest paid in month 12 is: "
  msgtext=msgtext + Format(intpaid, "Currency")
  MsgBox msgtext
End Sub
```

See Also [FV](#), [IRR](#), [NPV](#), [Pmt](#), [Ppmt](#), [PV](#), [Rate](#)

IRR Function

Action Returns the internal rate of return for a stream of periodic cash flows.

Syntax **IRR**(*valuearray*(), *guess*) where *valuearray*() is an array containing cash flow values and *guess* is a ballpark estimate of the value returned by **IRR**.

Comments *valuearray*() must have at least one positive value (representing a receipt) and one negative value (representing a payment). All payments and receipts must be represented in the exact sequence. The value returned by **IRR** will vary with the change in the sequence of cash flows.

In general, a *guess* value of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable estimate.

IRR is an iterative function. It improves a given guess over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

Example This example calculates an internal rate of return (expressed as an interest rate percentage) for a series of business transactions (income and costs). The first value entered must be a negative amount, or IRR generates an “Illegal Function Call” error.

```
Sub main
  Dim cashflows() as Double
```

```

Dim guess, count as Integer
Dim i as Integer
Dim int1 as Single
Dim msgtext as String
guess=.15
count=InputBox("How many cash flow amounts do you have?")
ReDim cashflows(count+1)
For i=0 to count-1
    cashflows(i)=InputBox("Enter income value for month " & i+1 & ":")
Next i
int1=IRR(cashflows(),guess)
msgtext="The IRR for your cash flow amounts is: "
msgtext=msgtext & Format(int1, "Percent")
MsgBox msgtext
End Sub

```

See Also [FV, Ipmt, NPV, Pmt, Ppmt, PV, Rate](#)

Is Operator

Action Compares two object expressions and returns -1 if they refer to the same object, 0 otherwise.

Syntax *objectExpression* **Is** *objectExpression* where objectexpression is any valid object expression.

Comments **Is** may also be used to test if an object variable has been **Set** to **Nothing**.

Example This example displays a list of open files in the software application, VISIO. It uses the Is operator to determine whether VISIO is available. To see how this example works, you need to start VISIO and open one or more documents.

```

Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

    'Initialize Visio
    Set visio = GetObject("visio.application")    ' find Visio
    If (visio Is Nothing) then
        MsgBox "Couldn't find Visio!"
    Exit Sub
    End If
    'Get # of open Visio files
    doccount = visio.documents.count 'OLE2 call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)    ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub

```

See Also [Create Object, Get Object, Me, Nothing, Object, Typeof](#)

IsDate Function

Action	Returns -1 (TRUE) if an expression is a legal date, 0 (FALSE) if it is not.
Syntax	IsDate (<i>expression</i>) where <i>expression</i> is any valid expression.
Comments	IsDate returns -1 (True) if the expression is of vartype 7 (date) or a string that may be interpreted as a date.
Example	This example adds a number to today's date value and checks to see if it is still a valid date (within the range January 1, 100AD through December 31, 9999AD).

```

Sub main
    Dim curdatevalue
    Dim yrs
    Dim msgtext
    curdatevalue=DateValue(Date$)
    yrs=InputBox("Enter a number of years to add to today's date")
    yrs=yrs*365
    curdatevalue=curdatevalue+yrs
    If IsDate(curdatevalue)=-1 then
        MsgBox "The new date is: " & Format(CVDate(curdatevalue), "dddddd")
    Else
        MsgBox "The date is not valid."
    End If
End Sub

```

See Also [CVDate](#), [IsEmpty](#), [IsNull](#), [IsNumeric](#), [VarType](#)

IsEmpty Function

Action	Returns -1 (TRUE) if a Variant has been initialized. 0 (FALSE) otherwise.
Syntax	IsEmpty (<i>expression</i>) where <i>expression</i> is any expression with a data type of Variant .
Comments	IsEmpty returns -1 (True) if the Variant is of vartype 0 (empty). Any newly defined Variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty Variant converts to zero when used in a numeric expression, or an empty string ("") in a string expression.
Example	This example prompts for a series of test scores and uses IsEmpty to determine whether the maximum allowable limit has been hit. (IsEmpty determines when to exit the Do...Loop.)

```

Sub main
    Dim arrayvar(10)
    Dim x as Integer
    Dim tscore as Single
    Dim total as Integer
    x=1

```

```

Do
    tscore=InputBox("Enter test score #" & x & ":")
    arrayvar(x)=tscore
    x=x+1
Loop Until IsEmpty(arrayvar(10))<>-1
total=x-1
msgtext="You entered: " & Chr(10)

For x=1 to total
    msgtext=msgtext & Chr(10) & arrayvar(x)
Next x
MsgBox msgtext
End Sub

```

See Also [IsDate](#), [IsNull](#), [IsNumeric](#), [VarType](#)

IsNull Function

Action	Returns -1 (TRUE) if a Variant expression contains the Null value, 0 (FALSE) otherwise.
Syntax	IsNull (<i>expression</i>) where <i>expression</i> is any expression with a data type of Variant .
Comments	Null Variants have no associated data and serve only to represent invalid or ambiguous results. Null is not the same as Empty, which indicates that a Variant has not yet been initialized.
Example	This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```

Sub main
    Dim arrayvar(10)
    Dim count as Integer
    Dim total as Integer
    Dim x as Integer
    Dim tscore as Single
    count=10
    total=0
    For x=1 to count
        tscore=InputBox("Enter test score #" & x & ":")
        If tscore<0 then
            arrayvar(x)=Null
        Else
            arrayvar(x)=tscore
            total=total+arrayvar(x)
        End If
    Next x
    Do While x<>0
        x=x-1
        If IsNull(arrayvar(x))=-1 then
            count=count-1
        End If
    End While
    MsgBox "Average: " & total/count
End Sub

```

```

Loop
msgtext="The average (excluding negative values) is: " & Chr(10)
msgtext=msgtext & Format (total/count, "##.##")
MsgBox msgtext
End Sub

```

See Also [IsDate](#), [IsEmpty](#), [IsNumeric](#), [VarType](#)

IsNumeric Function

- Action** Returns -1 (TRUE) if an expression has a data type of **Numeric**, 0 (FALSE) otherwise.
- Syntax** **IsNumeric**(*expression*) where *expression* is any valid expression.
- Comments** **IsNumeric** returns -1 (True) if the expression is of vartypes 2-6 (numeric) or a string that may be interpreted as a number.
- Example** This example uses IsNumeric to determine whether a user selected an option (1-3) or typed “Q” to quit.

```

Sub main
    Dim answer
    answer=InputBox("Enter a choice (1-3) or type Q to quit")
    If IsNumeric(answer)=-1 then
        Select Case answer
            Case 1
                MsgBox "You chose #1."
            Case 2
                MsgBox "You chose #2."
            Case 3
                MsgBox "You chose #3."
        End Select
    Else
        MsgBox "You typed Q."
    End If
End Sub

```

See Also [IsDate](#), [IsEmpty](#), [IsNull](#), [VarType](#)

Kill Statement

- Action** Deletes files from a hard disk or floppy drive.
- Syntax** **Kill** *pathname\$* where *pathname\$* is a String expression that specifies a valid DOS file specification.
- Comments** The *pathname\$* specification can contain paths and wildcards. **Kill** deletes files only, not directories. Use the **RmDir** function to delete directories.

Example This example prompts a user for an account number, opens a file, searches for the account number and displays the matching letter for that number. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram. After processing is complete, the first subroutine uses Kill to delete the file.

```
Declare Sub createfile()
Global x as Integer
Global y(100) as String

Sub main
Dim acctno as Integer
Dim msgtext
Call createfile
i: acctno=InputBox("Enter an account number from 1-10:")
If acctno<1 Or acctno>10 then
MsgBox "Invalid account number. Try again."
Goto i:
End if
x=1
Open "C:\TEMP001" for Input as #1
Do Until x=acctno
Input #1, x,y(x)
Loop
msgtext="The letter for account number " & x & " is: " & y(x)
Close #1
MsgBox msgtext
Kill "C:\TEMP001"
End Sub

Sub createfile()
' Put the numbers 1-10 and letters A-J into a file
Dim startletter
Open "C:\TEMP001" for Output as #1
startletter=65
For x=1 to 10
y(x)=Chr(startletter)
startletter=startletter+1
Next x
For x=1 to 10
Write #1, x,y(x)
Next x
Close #1
End Sub
```

See Also [FileAttr](#), [FileDateTime](#), [GetAttr](#), [RmDir](#)

LBound Function

Action Returns the lower bound of the subscript range for the specified array.

Syntax **LBound**(*arrayname* [, *dimension*]) where *arrayname* is the name of the array to use and *dimension* is the dimension to use.

- Comments** The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is used as a default.
- LBound** can be used with **UBound** to determine the length of an array.
- Example** This example resizes an array if the user enters more data than can fit in the array. It uses **LBound** and **UBound** to determine the existing size of the array and **ReDim** to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
        arrayvar(x)=InputBox("Enter test score #" & x & ":")
        x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
        count=InputBox("How many more do you have?")
        If count<>0 then
            count=count+(x-1)
            ReDim Preserve arrayvar(count)
            Goto start
        End If
    End If
    x=LBound(arrayvar,1)
    count=UBound(arrayvar,1)
    For y=x to count
        total=total+arrayvar(y)
    Next y
    MsgBox "The average of " & count & " scores is: " & Int(total/count)
End Sub
```

See Also [Dim](#), [Global](#), [Option Base](#), [ReDim](#), [Static](#), [UBound](#)

LCASE Function

- Action** Returns a copy of a string, with all uppercase letters converted to lowercase.
- Syntax** **LCASE**[\$](*string*\$) where *string*\$ is a string, or an expression containing the string to use.
- Comments** The translation is based on the country specified in the Windows Control Panel. **LCASE** accepts expressions of type String. **LCASE** accepts any type of argument and will convert the input value to a string.

The dollar sign, "\$", in the function name is optional. If specified the return type is String. If omitted the function will typically return a **Variant** of vartype 8 (string). If the value of *string\$* is NULL, a Variant of vartype 1 (Null) is returned.

Example This example converts a string entered by the user to lowercase.

```
Sub main
Dim userstr as String
userstr=InputBox$("Enter a string in upper and lowercase letters")
userstr=LCase$(userstr)
Msgbox "The string now is: " & userstr
End Sub
```

See Also [UCase](#)

Left Function

Action Returns a string of a specified length copied from the beginning of another string.

Syntax **Left**[\$](*string\$*, *length%*) where *string\$* is a string or an expression containing the string to copy and *length%* is the number of characters to copy.

Comments If the length of *string\$* is less than *length%*, **Left** returns the whole string.

Left accepts expressions of type String. **Left** accepts any type of *string\$*, including numeric values, and will convert the input value to a string.

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will typically return a **Variant** of vartype 8 (string). If the value of *string\$* is NULL, a Variant of vartype 1 (Null) is returned.

Example This example extracts a user's first name from the entire name entered.

```
Sub main
Dim username as String
Dim count as Integer
Dim firstname as String
Dim charspace
charspace=Chr(32)
username=InputBox("Enter your first and last name")
count=InStr(username,charspace)
firstname=Left(username,count)
Msgbox "Your first name is: " &firstname
End Sub
```

See Also [GetField](#), [Ltrim](#), [Mid Function](#), [Mid Statement](#), [Right](#), [Rtrim](#), [Str](#), [StrComp](#), [Trim](#)

Len Function

Action	Returns the length of a string or variable.
Syntax A	Len (<i>string\$</i>)
Syntax B	Len (<i>varname</i>)
	where <i>string\$</i> is a string or an expression that evaluates to a string and <i>varname</i> is a variable that contains a string.
Comments	<p>If the argument is a string, the number of characters in the string is returned. If the argument is a Variant variable, Len returns the number of bytes required to represent its value as a string; otherwise, the length of the built-in data type or user-defined type is returned.</p> <p>If syntax B is used, and <i>varname</i> is a Variant containing a NULL, Len will return a Null Variant.</p>
Example	<p>This example returns the length of a name entered by the user (including spaces).</p> <pre> Sub main Dim username as String username=InputBox("Enter your name") count=Len(username) MsgBox "The length of your name is: " &count End Sub </pre>
See Also	Instr

Let (Assignment Statement)

Action	Assigns an expression to a Basic variable.
Syntax	[Let] <i>variable</i> = <i>expression</i> where <i>variable</i> is the name of a variable to assign to the <i>expression</i> and <i>expression</i> is the expression to assign to the variable.
Comments	<p>The keyword Let is optional.</p> <p>The Let statement can be used to assign a value or expression to a variable with a data type of Numeric, String, Variant or Record variable. You can also use the Let statement to assign to a record field or to an element of an array.</p> <p>When assigning a value to a numeric or string variable, standard conversion rules apply.</p>

Example This example uses the Let statement for the variable sum. The subroutine finds an average of 10 golf scores.

```
Sub main
    Dim score As Integer
    Dim x, sum
    Dim msgtext
    Let sum=0
    For x=1 to 10
        score=InputBox("Enter your last ten golf scores #" & x & ":")
        sum=sum+score
    Next x
    msgtext="Your average is: " & CInt(sum/(x-1))
    MsgBox msgtext
End Sub
```

See Also [Const](#), [Lset](#), [Set](#)

Like Operator

Action Returns the value -1 (TRUE) if a string matches a pattern, 0 (FALSE) otherwise.

Syntax *string\$* **LIKE** *pattern\$* where *string\$* is any string expression. and *pattern\$* is any string expression to match to *string\$*.

Comments *pattern\$* may include the following special characters:

Character	Matches:
?	A single character
*	A set of zero or more characters
#	A single digit character (0-9)
[chars]	A single character in <i>chars</i>
[!chars]	A single character not in <i>chars</i>
[schar-echar]	A single character in range <i>schar</i> to <i>echar</i>
[!schar-echar]	A single character not in range <i>schar</i> to <i>echar</i>

Both ranges and lists may appear within a single set of square brackets. Ranges are matched according to their ANSI values. In a range, *schar* must be less than *echar*.

If either *string\$* or *pattern\$* is NULL then the result value is NULL.

The **Like** operator respects the current setting of **Option Compare**.

Example This example tests whether a letter is lowercase.

```
Sub main
  Dim userstr as String
  Dim revalue as Integer
  Dim msgtext as String
  Dim pattern
  pattern="[a-z]"
  userstr=InputBox$("Enter a letter:")
  revalue=userstr LIKE pattern
  If revalue=-1 then
    msgtext="The letter " & userstr & " is lowercase."
  Else
    msgtext="Not a lowercase letter."
  End If
  MsgBox msgtext
End Sub
```

See Also [Instr](#), [Option Compare](#), [StrComp](#)

Line Input Statement

Action	Reads a line from a sequential file into a string variable.
Syntax A	Line Input [#] <i>filenumber%</i> , <i>varname\$</i>
Syntax B	Line Input [<i>prompt\$</i> ,] <i>varname\$</i>
	<p>where is</p> <p><i>filenumber%</i> an integer expression identifying the open file to use.</p> <p><i>prompt\$</i> an optional string that can be used to prompt for keyboard input.</p> <p><i>varname\$</i> a string variable to contain the line read.</p>
Comments	<p>If specified, the <i>filenumber%</i> is the number used in the Open statement to open the file. If <i>filenumber%</i> is not provided, the line is read from the keyboard.</p> <p>If <i>prompt\$</i> is not provided, a prompt of “?” is used.</p>
Example	<p>This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.</p> <pre>Declare Sub createfile() Sub main Dim testscore as String Dim x Dim y Dim newline Call createfile Open "c:\temp001" for Input as #1 x=1 newline=Chr(10)</pre>

```

msgtext= "The contents of c:\temp001 is: " & newline
Do Until x=Lof(1)
  Line Input #1, testscore
  x=x+1
  y=Seek(1)
  If y>Lof(1) then
    x=Lof(1)
  Else
    Seek 1,y
  End If
  msgtext=msgtext & testscore & newline
Loop
MsgBox msgtext
Close #1
Kill "C:\TEMP001"
End Sub

Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
  Close #1
End Sub

```

See Also [Get](#), [Input Function](#), [Input Statement](#), [Open](#)

ListBox Statement

Action	Defines a list box of choices for a dialog box.
Syntax A	ListBox <i>x</i> , <i>y</i> , <i>dx</i> , <i>dy</i> , <i>text</i> \$, <i>field</i>
Syntax B	ListBox <i>x</i> , <i>y</i> , <i>dx</i> , <i>dy</i> , <i>stringarray</i> \$(), <i>field</i>
where	is
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the list box.
<i>text</i> \$	a string containing the selections for the list box.
<i>stringarray</i> \$	an array of dynamic strings for the selections in the list box.
<i>field</i>	the name of the dialog-record field that will hold a number for the choice made in the list box.
Comments	The <i>x</i> argument is measured in 1/4 system-font character-width units. The <i>y</i> argument is measured in 1/8 system-font character-width units. (See Begin Dialog for more information.)

The *text\$* argument must be defined, using a Dim Statement, before the Begin Dialog statement is executed. The arguments in the *text\$* string are entered as shown in the following example:

```
dimname = "listchoice"+Chr$(9)+"listchoice"+Chr$(9)+"listchoice"...
```

A number representing the selection's position in the *text\$* string is recorded in the field designated by the *field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The numbers begin at 0. If no item is selected, it is -1. The *field* argument is also used by the dialog statements that act on this control.

Use the **ListBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with list box and two buttons.

```
Sub main
  Dim ListBox1() as String
  ReDim ListBox1(0)
  ListBox1(0)="C:\"
  Begin Dialog UserDialog 133, 66, 171, 65, "SBL Dialog Box"
    Text 3, 3, 34, 9, "Directory:", .Text2
    Listbox 3, 14, 83, 39, ListBox1(), .ListBox2
    OkButton 105, 6, 54, 14
    CancelButton 105, 26, 54, 14
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also [Begin...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [OkButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Loc Function

Action Returns the current offset within an open file.

Syntax **Loc**(*filenumber%*) where *filenumber%* is an integer expression identifying the open file to query.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

For files opened in **Random** mode, **Loc** returns the number of the last record read or written. For files opened in **Append**, **Input**, or **Output** mode, **Loc** returns the current byte offset divided by 128. For files opened in **Binary** mode, **Loc** returns the offset of the last byte read or written.

Example This example creates a file of account numbers as entered by the user. When the user finishes, the example displays the offset in the file of the last entry made.

```
Sub main
  Dim filepos as Integer
  Dim acctno() as Integer
  Dim x as Integer
  x=0
  Open "c:\TEMP001" for Random as #1
  Do
    x=x+1
    Redim Preserve acctno(x)
    acctno(x)=InputBox("Enter account #" & x & " or 0 to end:")
    If acctno(x)=0 then
      Exit Do
    End If
    Put #1,, acctno(x)
  Loop
  filepos=Loc(1)
  Close #1
  MsgBox "The offset is: " & filepos
  Kill "C:\TEMP001"
End Sub
```

See Also [Eof](#), [Lof](#), [Open](#)

Lock, Unlock Statements

Action Controls access to an open file.

Syntax **Lock** [#]*filenumber%* [, [*start&*] [**To** *end&*]]

Unlock [#]*filenumber%* [, { *record&* | [*start&*] **To** *end&* }]

where **is**

filenumber% an integer expression identifying the open file.

record& number of the starting record to unlock.

start& number of the first record or byte offset to lock/unlock.

end& number of the last record or byte offset to lock/unlock.

Comments

The *filenumber%* is the number used in the **Open** statement of the file.

For **Binary** mode, *start&*, and *end&* are byte offsets. For **Random** mode, *start&*, and *end&* are record numbers. If *start&* is specified without *end&*, then only the record or byte at *start&* is locked. If **To** *end&* is specified without *start&*, then all records or bytes from record number or offset 1 to *end&* are locked.

For **Input**, **Output** and **Append** modes, *start&*, and *end&* are ignored and the whole file is locked.

Lock and **Unlock** always occur in pairs with identical parameters. All locks on open files must be removed before closing the file, or unpredictable results will occur.

Example

This example locks a file that is shared by others on a network, if the file is already in use. The second subprogram, CREATEFILE, creates the file used by the main subprogram.

```

Declare Sub createfile
Sub main
  Dim btngrp, icongrp
  Dim defgrp
  Dim answer
  Dim noaccess as Integer
  Dim msgabort
  Dim msgstop as Integer
  Dim acctname as String
  noaccess=70
  msgstop=16
  Call createfile
  On Error Resume Next
  btngrp=1
  icongrp=64
  defgrp=0
  answer=MsgBox("Open the account file?" & Chr(10), btngrp+icongrp+defgrp)
  If answer=1 then
    Open "C:\TEMP001" for Input as #1
    If Err=noaccess then
      msgabort=MsgBox("File Locked",msgstop,"Aborted")
    Else
      Lock #1
      Line Input #1, acctname
      MsgBox "The first account name is: " & acctname
      Unlock #1
    End If
    Close #1
  End If
  Kill "C:\TEMP001"
End Sub

Sub createfile()
  Rem Put the letters A-J into the file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1

```

```
For x=1 to 10
  Write #1, Chr(x+64)
Next x
Close #1
End Sub
```

See Also [Open](#)

Lof Function

Action Returns the length in bytes of an open file.

Syntax **Lof**(*filenumber%*) where *filenumber%* is an integer expression identifying the open file.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

Example This example opens a file and prints its contents to the screen.

```
Sub main
  Dim fname
  Dim fchar()
  Dim x as Integer
  Dim msgtext
  Dim newline
  newline=Chr(10)
  fname=InputBox("Enter a filename to print:")
  On Error Resume Next
  Open fname for Input as #1
  If Err<>0 then
    MsgBox "Error loading file. Re-run program."
    Exit Sub
  End If
  msgtext="The contents of " & fname & " is: " & newline & newline
  Redim fchar(Lof(1))
  For x=1 to Lof(1)
    fchar(x)=Input(1,#1)
    msgtext=msgtext & fchar(x)
  Next x

  MsgBox msgtext
  Close #1
End Sub
```

See Also [Eof](#), [FileLen](#), [Loc](#), [Open](#)

Log Function

Action	Returns the natural logarithm of a number.
Syntax	Log (<i>number</i>) where <i>number</i> is any valid numeric expression.
Comments	The return value is single-precision for an integer, currency or single-precision numeric expression, double precision for a long, Variant or double-precision numeric expression.
Example	This example uses the Log function to determine which number is larger: 999^1000 (999 to the 1000 power) or 1000^999 (1000 to the 999 power). Note that you can't use the exponent (^) operator for numbers this large.

```

Sub main
    Dim x
    Dim y
    x=999
    y=1000
    a=y*(Log(x))
    b=x*(Log(y))
    If a>b then
        MsgBox "999^1000 is greater than 1000^999"
    Else
        MsgBox "1000^999 is greater than 999^1000"
    End If
End Sub

```

See Also [Exp](#), [FixInt](#), [Int](#), [Rnd](#), [Sgn](#), [Sqr](#)

Lset Statement

Action	Copies one string to another, or assigns a user-defined type variable to another.										
Syntax A	Lset <i>string\$</i> = <i>string-expression</i>										
Syntax B	Lset <i>variable1</i> = <i>variable2</i>										
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>string\$</i></td><td>a string or string expression to contain the copied characters.</td></tr> <tr> <td><i>string-expression</i></td><td>an expression containing the string to copy.</td></tr> <tr> <td><i>variable1</i></td><td>a variable with a user-defined type to contain the copied variable.</td></tr> <tr> <td><i>variable2</i></td><td>a variable with a user-defined type to copy.</td></tr> </table>	where	is	<i>string\$</i>	a string or string expression to contain the copied characters.	<i>string-expression</i>	an expression containing the string to copy.	<i>variable1</i>	a variable with a user-defined type to contain the copied variable.	<i>variable2</i>	a variable with a user-defined type to copy.
where	is										
<i>string\$</i>	a string or string expression to contain the copied characters.										
<i>string-expression</i>	an expression containing the string to copy.										
<i>variable1</i>	a variable with a user-defined type to contain the copied variable.										
<i>variable2</i>	a variable with a user-defined type to copy.										
Comments	If <i>string\$</i> is shorter than <i>string-expression</i> , Lset copies the leftmost character of <i>string-expression</i> into <i>string\$</i> . The number of characters copied is equal to the length of <i>string\$</i> .										

If *string* is longer than *string-expression*, all characters of *string-expression* are copied into *string\$*, filling it from left to right. All leftover characters of *string\$* are replaced with spaces.

In Syntax B, the number of characters copied is equal to the length of the shorter of *variable1* and *variable2*.

Lset cannot be used to assign variables of different user-defined types if either contains a **Variant** or a variable-length string.

Example

This example puts a user's last name into the variable LASTNAME. If the name is longer than the size of LASTNAME, then the user's name is truncated. If you have a long last name and you get lots of junk mail, you've probably seen how this works already.

```
Sub main
    Dim lastname as String
    Dim strlast as String*8
    lastname=InputBox("Enter your last name")
    Lset strlast=lastname
    msgtext="Your last name is: " &strlast
    MsgBox msgtext
End Sub
```

See Also [Rset](#)

LTrim Function

Action Returns a copy of a string with all leading space characters removed.

Syntax **LTrim**[\$](*string\$*) where *string\$* is a string or expression containing a string to copy.

Comments **LTrim** accepts any type of *string\$*, including numeric values, and will convert the input value to a string.

The dollar sign, "\$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string\$* is NULL, a Variant of vartype 1 (Null) is returned.

Example

This example trims the leading spaces from a string padded with spaces on the left.

```
Sub main
    Dim userInput as String
    Dim numsize
    Dim str1 as String*50
    Dim strsize
    strsize=50
    userInput=InputBox("Enter a string of characters:")
    numsize=Len(userInput)
    str1=Space(strsize-numsize) & userInput
    ' Str1 has a variable number of leading spaces.
    MsgBox "The string is: " &str1
```



```

str1=LTrim$(str1)
' Str1 now has no leading spaces.
MsgBox "The string now has no leading spaces: " & str1
End Sub

```

See Also [GetField](#), [Left](#), [Mid Function](#), [Mid Statement](#), [Right](#), [Rtrim](#), [Trim](#)

Me

Action	Refers to the currently used OLE2 automation object.
Syntax	Me
Comments	<p>Some Basic modules are attached to application objects and Basic subroutines are invoked when that application object encounters events. A good example is a user visible button that triggers a Basic routine when the user clicks the mouse on the button.</p> <p>Subroutines in such contexts may use the variable Me to refer to the object which triggered the event (i.e., which button was clicked). The programmer may use Me in all the same ways as any other object variable except that Me may not be Set.</p>
Example	<p>This example</p> <pre> Sub main ---TBD--- End Sub </pre>
See Also	Create Object , Get Object , New , Nothing , Object , Typeof

Mid Function

Action	Returns a portion of a string, starting at a specified location within the string.
Syntax	<p>Mid[\$](<i>string</i>\$, <i>start</i>%[, <i>length</i>%])</p> <p>where is</p> <p><i>string</i>\$ a string or expression that contains the string to change.</p> <p><i>start</i>% the starting position in <i>string</i>\$ to begin replacing characters.</p> <p><i>length</i>% the number of characters to replace.</p>
Comments	<p>Mid accepts any type of <i>string</i>\$, including numeric values, and will convert the input value to a string. If the <i>length</i>% argument is omitted, or if <i>string</i>\$ is smaller than <i>length</i>%, then Mid returns all characters in <i>string</i>\$. If <i>start</i>% is larger than <i>string</i>%, then Mid returns a null string ("").</p> <p>The index of the first character in a string is 1.</p>

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string\$* is Null, a Variant of vartype 1 (Null) is returned.

To modify a portion of a string value, see **Mid Statement**.

Example

This example uses the Mid statement to replace the last name in a user-entered string to asterisks(*)).

```
Sub main
    Dim username as String
    Dim position as Integer
    Dim count as Integer
    Dim uname as String
    Dim replacement as String
    username=InputBox("Enter your full name:")
    uname=username
    replacement="*"
    Do
        position=InStr(username," ")
        If position=0 then
            Exit Do
        End If
        username=Mid(username,position+1)
        count=count+position
    Loop
    For x=1 to Len(username)
        count=count+1
        Mid(uname,count)=replacement
    Next x
    MsgBox "Your name now is: " & uname
End Sub
```

See Also [GetField](#), [Left](#), [Len](#), [Ltrim](#), [Mid Function](#), [Right](#), [Rtrim](#), [Trim](#)

Mid Statement

Action Replaces part (or all) of one string with another, starting at a specified location.

Syntax **Mid** (*stringvar\$*, *start%* [, *length%*]) = *string\$*

where	is
<i>stringvar\$</i>	the string to change.
<i>start%</i>	an expression for the position to begin replacing characters.
<i>length%</i>	an expression for the number of characters to replace.
<i>string\$</i>	the string to place into another string.

Comments If the *length%* argument is omitted, or if *string\$* is smaller than *length%*, then **Mid** replaces all the characters from the *start%* to the end of the *string\$*. If *start%* is larger than the number of characters in the indicated *stringvar\$*, then **Mid** appends *string%* to *stringvar\$*.

The index of the first character in a string is 1.

Example This example uses the Mid function to find the last name in a string. entered by the user.

```
Sub main
  Dim username as String
  Dim position as Integer
  username=InputBox("Enter your full name:")
  Do
    position=InStr(username," ")
    If position=0 then
      Exit Do
    End If
    position=position+1
    username=Mid(username,position)
  Loop
  MsgBox "Your last name is: " & username
End Sub
```

See Also [GetField](#), [Lcase](#), [Left](#), [Len](#), [Ltrim](#), [Mid Statement](#), [Right](#), [Rtrim](#), [Trim](#)

Minute Function

- Action** Returns an integer for the minute component (0-59) of a date-time value.
- Syntax** **Minute**(*time*) where *time* is any expression that can evolve to a date-time value.
- Comments** **Minute** accepts any type of *time*, including strings, and will attempt to convert the input value to a date value.
- The return value is a **Variant** of vartype 2 (Integer). If the value of *time* is null, a Variant of vartype 1 (null) is returned.
- Example** This example extracts just the time (hour, minute, and second) from a file's last modification date and time.

```
Sub main
  Dim filename as String
  Dim ftime
  Dim hr, min
  Dim sec
  Dim msgtext as String
  i: msgtext="Enter a filename:"
  filename=InputBox(msgtext)
  If filename="" then
    Exit Sub
  End If
  On Error Resume Next
  ftime=FileDateTime(filename)
  If Err<>0 then
    MsgBox "Error in file name. Try again."
    Goto i:
  End If
```

```

hr=Hour(ftime)
min=Minute(ftime)
sec=Second(ftime)
Msgbox "The file's time is: " & hr & ":" & min & ":" & sec
End Sub

```

See Also [Day](#), [Hour](#), [Month](#), [Now](#), [Second](#), [Time Function](#), [Time Statement](#), [Weekday](#), [Year](#)

MkDir Statement

Action Creates a new directory.

Syntax **MkDir** *path\$* where *path\$* is a string expression identifying the new default directory to create.

Comments The syntax for *path\$* is:

[*drive:*] [\] *directory* [\i*directory*]

The *drive* argument is optional. If *drive* is omitted, **MkDir** makes a new directory on the current drive. The *directory* argument is any directory name.

Example This example makes a new temporary directory in C:\ and then deletes it.

```

Sub main
    Dim path as String
    On Error Resume Next
    path=CurDir(C)
    If path<>"C:\" then
        ChDir "C:\"
    End If
    MkDir "C:\TEMP01"
    If Err=75 then
        MsgBox "Directory already exists"
    Else
        MsgBox "Directory C:\TEMP01 created"
        MsgBox "Now removing directory"
        Rmdir "C:\TEMP01"
    End If
End Sub

```

See Also [ChDir](#), [ChDrive](#), [CurDir](#), [Dir](#), [Rmdir](#)

Month Function

Action	Returns an integer for the month component (1-12) of a date-time value.
Syntax	Month (<i>date</i>) where <i>date</i> is any expression that evaluates to a date-time value.
Comments	It accepts any type of <i>date</i> , including strings, and will attempt to convert the input value to a date value. The return value is a Variant of vartype 2 (integer). If the value of <i>date</i> is null, a Variant of vartype 1 (null) is returned.

Example This example finds the month (1-12) and day (1-31) values for this Thursday.

```
Sub main
    Dim x, today
    Dim msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " & Month(Today+x)&" "&Day(Today+x)
    MsgBox msgtext
End Sub
```

See Also [Date Function](#), [Date Statement](#), [Day](#), [Hour](#), [Minute](#), [Now](#), [Second](#), [Weekday](#), [Year](#)

Msgbox Function

Action	Displays a message dialog box and returns a value (1-7) indicating which button the user selected.								
Syntax	Msgbox (<i>prompt\$</i> ,[<i>buttons%</i>][, <i>title\$</i>])								
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>prompt\$</i></td><td>the text to display in a dialog box.</td></tr> <tr> <td><i>buttons%</i></td><td>an integer value for the buttons, the icon, and the default button choice to display in a dialog box.</td></tr> <tr> <td><i>title\$</i></td><td>a string expression containing the title for the message box.</td></tr> </table>	where	is	<i>prompt\$</i>	the text to display in a dialog box.	<i>buttons%</i>	an integer value for the buttons, the icon, and the default button choice to display in a dialog box.	<i>title\$</i>	a string expression containing the title for the message box.
where	is								
<i>prompt\$</i>	the text to display in a dialog box.								
<i>buttons%</i>	an integer value for the buttons, the icon, and the default button choice to display in a dialog box.								
<i>title\$</i>	a string expression containing the title for the message box.								
Comments	<i>prompt\$</i> must be no more than 1,024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character. <i>buttons%</i> is the sum of three values, one from each of the following groups:								

Group	Value	Description
1: Buttons	0	OK only
	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
2: Icons	16	Critical Message (STOP)
	32	Warning Query (?)
	48	Warning Message (!)
	64	Information Message (i)
3: Defaults	0	First button
	256	Second button
	512	Third button

If *buttons%* is omitted, MsgBox displays a single OK button.

After the user clicks a button, **Msgbox** returns a value indicating the user's choice. The return values for the MsgBox function are:

Value	Button Pressed
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Example

This example displays one of each type of message box.

```

Sub main
  Dim btngrp as Integer
  Dim icongrp as Integer
  Dim defgrp as Integer
  Dim msgtext as String
  icongrp=16
  defgrp=0
  btngrp=0
  Do Until btngrp=6
    Select Case btngrp
      Case 1, 4, 5
        defgrp=0
      Case 2
        defgrp=256
      Case 3
    
```

```

    defgrp=512
End Select
msgtext=" Icon group = " & icongrp & Chr(10)
msgtext=msgtext + " Button group = " & btngroup & Chr(10)
msgtext=msgtext + " Default group = " & defgrp & Chr(10)
msgtext=msgtext + Chr(10) + " Continue?"
answer=MsgBox(msgtext, btngroup+icongrp+defgrp)
Select Case answer
Case 2,3,7
Exit Do
End Select
If icongrp<>64 then
icongrp=icongrp+16
End If
btngroup=btngroup+1
Loop
End Sub

```

See Also [InputBox](#), [MsgBox Statement](#), [PasswordBox](#)

Msgbox Statement

Action Displays a prompt in a message dialog box.

Syntax **MsgBox** *prompt*\$, [*buttons*%][, *title*\$] where *prompt*\$ is the text to display in a dialog box, *buttons*% is an integer value for the buttons, the icon, and the default button choice to display in a dialog box, and *title*\$ is a string expression containing the title for the message box.

Comments *Prompt*\$ must be no more than 1,024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.

buttons% is the sum of three values, one from each of the following groups:

Group	Value	Description
1: Buttons	0	OK only
	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
2: Icons	16	Critical Message (STOP)
	32	Warning Query (?)
	48	Warning Message (!)
	64	Information Message (i)
3: Defaults	0	First button
	256	Second button
	512	Third button

If *buttons*% is omitted, MsgBox displays a single OK button.

Example This example finds the future value of an annuity, whose terms are defined by the user. It uses the MsgBox statement to display the result.

```
Sub main
    Dim aprate, periods
    Dim payment, annuitypv
    Dim due, futurevalue
    Dim msgtext
    annuitypv=InputBox("Enter present value of the annuity: ")
    aprate=InputBox("Enter the annual percentage rate: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    periods=InputBox("Enter the total number of pay periods: ")
    payment=InputBox("Enter the initial amount paid to you: ")

    Rem Assume payments are made at end of month
    due=0
    futurevalue=FV(aprate/12,periods,-payment,-annuitypv,due)
    msgtext="The future value is: " & Format(futurevalue, "Currency")
    MsgBox msgtext
End Sub
```

See Also [InputBox](#), [MsgBox Function](#), [PasswordBox](#)

Name Statement

Action Renames a file or moves a file from one directory to another.

Syntax **Name** *oldfilename\$* **As** *newfilename\$* where *oldfilename\$* is a string expression containing the file to rename and *newfilename\$* is a string expression containing the name for the file.

Comments A path may be part of either filename argument. If the paths are different, the file is moved to the new directory.

A file must be closed in order to be renamed. If the file *oldfilename\$* is open or if the file *newfilename\$* already exists, Basic generates an error message.

Example This example creates a temporary file, C:\TEMP001, renames the file to C:\TEMP002, then deletes them both. It calls the subprogram, CREATEFILE, to create the C:\TEMP001 file.

```
Declare Sub createfile()
Sub main
    Call createfile
    On Error Resume Next
    Name "C:\TEMP001" As "C:\TEMP002"
    MsgBox "The file has been renamed"
    MsgBox "Now deleting both files"
    Kill "TEMP001"
    Kill "TEMP002"
End Sub
```



```
Sub createfile()  
    Rem Put the numbers 1-10 into a file  
    Dim x as Integer  
    Dim y()  
    Dim startletter  
    Open "C:\TEMP001" for Output as #1  
    For x=1 to 10  
        Write #1, x  
    Next x  
    Close #1  
End Sub
```

See Also [FileAttr](#), [FileCopy](#), [GetAttr](#), [Kill](#)

New Operator

Action	Allocates and initializes a new OLE2 object of the named class.
Syntax	Set <i>objectVar</i> = New <i>className</i> Dim <i>objectVar</i> As New <i>className</i> where <i>objectVar</i> is the OLE2 object to allocate and initialize and <i>className</i> is the class to assign to the object.
Comments	In the Dim statement, New marks <i>objectVar</i> so that a new object will be allocated and initialized when <i>objectVar</i> is first used. If <i>objectVar</i> is not referenced, then no new object will be allocated.
+	An object variable that was declared with New will allocate a second object if <i>objectVar</i> is Set to Nothing and referenced again.
Example	This example Sub main ---TBD--- End Sub
See Also	Dim , Global , Set , Static

\$NoCStrings Metacommand [SBL Extension]**

Action	Tells the compiler to treat a backslash(\) inside a string as a normal character.
Syntax	'\$NoCStrings [Save]where Save means saves the current '\$CStrings setting before restoring the treatment of the backslash (\) to a normal character.
Comments	Use the '\$CStings Restore command to restore a previously saved setting. Save and Restore operate as a stack and allow the user to change the '\$CStrings setting for a range of the program without impacting the rest of the program.

Use the **'\$CStrings** metacommand to tell the compiler to treat a backslash (\) inside of a string as an Escape character.

****SBL** offers a number of extensions that are not included in Visual Basic.

Example This example displays two lines, the first time using the C language characters “\n” for a carriage return and line feed.

```
Sub main
'$CStrings
MsgBox "This is line 1\n This is line 2 (using C Strings)"
'$NoCStrings
MsgBox "This is line 1" + Chr$(13)+Chr$(10)+"This is line 2 (using Chr)"
End Sub
```

See Also [\\$Cstrings](#), [\\$Include](#), [Rem](#)

Nothing Function

Action Returns an object value that doesn't refer to an object.

Syntax **Set** *variableName* = **Nothing** where *variableName* is the name of the object variable to set to nothing.

Comments **Nothing** is the value object variables have when they do not refer to an object, either because they have not been initialized yet or because they were explicitly **Set** to **Nothing**. For example:

```
    If Not objectVar Is Nothing then
        objectVar.Close
        Set objectVar = Nothing
    End If
```

Example This example displays a list of open files in the software application VISIO. It uses the Nothing function to determine whether VISIO is available. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
Dim visio as Object
Dim doc as Object
Dim msgtext as String
Dim i as Integer, doccount as Integer

'Initialize Visio
Set visio = GetObject(,"visio.application") ' find Visio
If (visio Is Nothing) then
Msgbox "Couldn't find Visio!"
Exit Sub
End If
'Get # of open Visio files
doccount = visio.documents.count 'OLE2 call to Visio
```

```

If doccount=0 then
    msgtext="No open Visio documents."
Else
    msgtext="The open files are: " & Chr$(13)
    For i = 1 to doccount
        Set doc = visio.documents(i) ' access Visio's document method
        msgtext=msgtext & Chr$(13)& doc.name
    Next i
End If
MsgBox msgtext
End Sub

```

See Also [Is](#), [New](#)

Now Function

Action	Returns the current date and time.
Syntax	Now()
Comments	The Now function returns a Variant of vartype 7 (date) that represents the current date and time according to the setting of the computer's system date and time.
Example	This example finds the month (1-12) and day (1-31) values for this Thursday.

```

Sub main
    Dim x, today
    Dim msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " & Month(Today+x)&"/"&Day(Today+x)
    MsgBox msgtext
End Sub

```

See Also [Date Function](#), [Date Statement](#), [Day](#), [Hour](#), [Minute](#), [Month](#), [Second](#), [Time Function](#), [Time Statement](#), [Weekday](#), [Year](#)

NPV Function

Action	Returns the net present value of a investment based on a stream of periodic cash flows and a constant interest rate.
Syntax	NPV (rate, valuearray()) where <i>rate</i> is the discount rate per period and <i>valuearray()</i> is an array containing cash flow values.

Comments *Valuearray*() must have at least one positive value (representing a receipt) and one negative value (representing a payment). All payments and receipts must be represented in the exact sequence. The value returned by **NPV** will vary with the change in the sequence of cash flows.

If the discount rate is 12% per period, *rate* is the decimal equivalent, i.e. 0.12.

NPV uses future cash flows as the basis for the net present value calculation. If the first cash flow occurs at the beginning of the first period, its value should be added to the result returned by **NPV** and must not be included in *valuearray*().

Example This example finds the net present value of an investment, given a range of cash flows by the user.

```
Sub main
  Dim aprate as Single
  Dim varray() as Double
  Dim cflowper as Integer
  Dim x as Integer
  Dim netpv as Double
  cflowper=InputBox("Enter number of cash flow periods")
  ReDim varray(cflowper)
  For x= 1 to cflowper
    varray(x)=InputBox("Enter cash flow amount for period #" & x & ":")
  Next x
  aprate=InputBox("Enter discount rate: ")
  If aprate>1 then
    aprate=aprate/100
  End If
  netpv=NPV(aprate,varray())
  MsgBox "The net present value is: " & Format(netpv, "Currency")
End Sub
```

See Also [FV](#), [Ipmt](#), [IRR](#), [Pmt](#), [Ppmt](#), [PV](#), [Rate](#)

Null Function

Action Returns a **Variant** value set to NULL.

Syntax **Null**

Comments Null is used to set a Variant to the Null value explicitly, as follows:

variableName = Null

Note that Variants are initialized by Basic to the empty value, which is different from the null value.

Example This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```

Sub main
  Dim arrayvar(10)
  Dim count as Integer
  Dim total as Integer
  Dim x as Integer
  Dim tscore as Single
  count=10
  total=0
  For x=1 to count
    tscore=InputBox("Enter test score #" & x & ":")
    If tscore<0 then
      arrayvar(x)=Null
    Else
      arrayvar(x)=tscore
      total=total+arrayvar(x)
    End If
  Next x
  Do While x<>0
    x=x-1
    If IsNull(arrayvar(x))=-1 then
      count=count-1
    End If
  Loop
  msgtext="The average (excluding negative values) is: " & Chr(10)
  msgtext=msgtext & Format (total/count, "##.##")
  MsgBox msgtext
End Sub

```

See Also [IsEmpty](#), [IsNull](#), [VarType](#)

Object Class

Action	A class that provides access to OLE2 automation objects.
Syntax	Dim <i>variableName</i> As Object where <i>variableName</i> is the name of the object variable to declare.
Comments	To create a new object, first dimension a variable, using the Dim statement, then Set the variable to the return value of CreateObject or GetObject , as follows:
	<pre> Dim OLE2 As Object SetOLE2 = CreateObject("spoly.cpoly") </pre> <p>To refer to a method or property of the newly created object, use the syntax: <i>objectvar.property</i> or <i>objectvar.method</i>, as follows: <i>OLE2.reset</i></p>
Example	This example displays a list of open files in the software application VISIO. It uses the Object class to declare the variables used for accessing VISIO and its document files and methods.

```

Sub main
  Dim visio as Object
  Dim doc as Object
  Dim msgtext as String
  Dim i as Integer, doccount as Integer

```

```
'Initialize Visio
Set visio = GetObject("visio.application")      ' find Visio
If (visio Is Nothing) then
    MsgBox "Couldn't find Visio!"
Exit Sub
End If
'Get # of open Visio files
doccount = visio.documents.count 'OLE2 call to Visio
If doccount=0 then
    msgtext="No open Visio documents."
Else
    msgtext="The open files are: " & Chr$(13)
    For i = 1 to doccount
        Set doc = visio.documents(i) ' access Visio's document method
        msgtext=msgtext & Chr$(13)& doc.name
    Next i
End If
MsgBox msgtext
End Sub
```

See Also [Create Object](#), [Get Object](#), [New](#), [Nothing](#), [Typeof](#)

Oct Function

Action Returns the octal representation of a number, as a string.

Syntax **Oct**[\$](*number*) where *number* is a numeric expression for the number to convert to octal.

Comments If the numeric expression has a data type of **Integer**, the string will contain up to six octal digits; otherwise, the expression will be converted to a data type of **Long**, and the string may contain up to 11 octal digits.

To represent an octal number directly, precede the octal value with **&O**. For example, &O10 equals decimal 8 in octal notation.

The dollar sign, "\$", in the function name is optional. If specified the return data type is **String**. If omitted the function will return a **Variant** of vartype 8 (string).

Example This example prints the octal values for the numbers from 1 to 15.

```
Sub main
    Dim x,y
    Dim msgtext
    Dim nospaces
    msgtext="Octal numbers from 1 to 15:" & Chr(10)
    For x=1 to 15
        nospaces=10
        y=Oct(x)
        If Len(x)=2 then
            nospaces=nospaces-2
        End If
        msgtext=msgtext & Chr(10) & x & Space(nospaces) & y
    Next x
End Sub
```

```
Next x
MsgBox msgtext
End Sub
```

See Also [Hex](#)

OkButton Statement

Action	Determines the position and size of an OK button in a dialog box.
Syntax	OkButton <i>x</i> , <i>y</i> , <i>dx</i> , <i>dy</i> [, <i>.id</i>] where is <i>x,y</i> the position of the Cancel button relative to the upper left corner of the dialog box. <i>dx,dy</i> the width and height of the button. <i>.id</i> an optional identifier for the button.
Comments	<p>A <i>dy</i> value of 14 typically accommodates text in the system font.</p> <p><i>.id</i> is an optional identifier used by the dialog statements that act on this control.</p> <p>Use the OkButton statement only between a Begin Dialog and an End Dialog statement.</p>
Example	<p>This example defines a dialog box with a dropcombo box and the OK and Cancel buttons.</p> <pre>Sub main Dim cchoices as String On Error Resume Next cchoices="All"+Chr\$(9)+"Nothing" Begin Dialog UserDialog 180, 95, "SBL Dialog Box" ButtonGroup .ButtonGroup1 Text 9, 3, 69, 13, "Filename:".Text1 DropComboBox 9, 17, 111, 41, cchoices, .ComboBox1 OkButton 131, 8, 42, 13 CancelButton 131, 27, 42, 13 End Dialog Dim mydialogbox As UserDialog Dialog mydialogbox If Err=102 then MsgBox "You pressed Cancel." Else MsgBox "You pressed OK." End If End Sub</pre>
See Also	Begin...End Dialog , Button , ButtonGroup , CancelButton , Caption , CheckBox , ComboBox , Dialog , DropComboBox , GroupBox , ListBox , OptionButton , OptionGroup , Picture , StaticComboBox , Text , TextBox

On...Goto Statement

Action	Branch to a label in the current procedure based on the value of a numeric expression.
Syntax	ON <i>numeric-expression</i> GoTo <i>label1</i> [, <i>label2</i> , ...] where <i>numeric-expression</i> is any numeric expression that evaluates to a positive number and <i>label1</i> , <i>label2</i> are label in the current procedure to branch to if <i>numeric-expression</i> evaluates to 1, 2, etc.
Comments	If <i>numeric expression</i> evaluates to 0 or to a number greater than the number of labels following GoTo , the program continues at the next statement. If <i>numeric-expression</i> evaluates to a number less than 0 or greater than 255, an “Illegal function call” error is issued.
Example	This example sets the current system time to the user's entry. If the entry cannot be converted to a valid time value, this subroutine sets the variable to Null. It then checks the variable and if it is Null, uses the On...Goto statement to ask again.

```

Sub main
    Dim answer as Integer
    answer=InputBox("Enter a choice (1-3) or 0 to quit")
    On answer Goto c1, c2, c3
    MsgBox("You typed 0.")
    Exit Sub
c1:    MsgBox("You picked choice 1.")
    Exit Sub
c2:    MsgBox("You picked choice 2.")
    Exit Sub
c3:    MsgBox("You picked choice 3.")
    Exit Sub
End Sub

```

See Also [Goto, Select Case](#)

On Error Statement

Action	Specifies the location of an error-handling routine within the current procedure.
Syntax	ON [Local] Error { GoTo <i>label</i> [Resume Next] GoTo 0 } where <i>label</i> is a string used as a label in the current procedure to identify the lines of code that process errors.
Comments	On Error can also be used to disable an error-handling routine. Unless an On Error statement is used, any run-time error will be fatal, i.e., SBL will terminate the execution of the program.

An **On Error** statement is composed of the following parts:

Part	Definition
Local	Keyword allowed in error-handling routines at the procedure level. Used to ensure compatibility with other Variants of Basic.
GoTo label	Enables the error-handling routine that starts at label. If the designated label is not in the same procedure as the On Error statement, SBL generates an error message.
Resume Next	Designates that error handling code is handled by the statement which immediately follows the statement that caused an error. At this point, use the Error function to retrieve the error-code of the run-time error.
GoTo 0	Disables any error handler that has been enabled.

When it is referenced by an **On Error GoTo label** statement, an error-handler is enabled. Once this enabling occurs, a run-time error will result in program control switching to the error-handling routine and “activating” the error handler. The error handler remains active from the time the run-time error has been trapped until a **Resume** statement is executed in the error handler.

If another error occurs while the error handler is active, SBL will search for an error handler in the procedure which called the current procedure (if this fails, SBL will look for a handler belonging to the caller’s caller, ...). If a handler is found, the current procedure will terminate, and the error handler in the calling procedure will be activated.

It is an error (No Resume) to execute an **End Sub** or **End Function** statement while an error handler is active. The **Exit Sub** or **Exit Function** statement can be used to end the error condition and exit the current procedure.

Example

This example prompts the user for a drive and directory name and uses On Error to trap invalid entries.

```

Sub main
    Dim userdrive, userdir, msgtext
    in1: userdrive=InputBox("Enter drive:","C:")
    On Error Resume Next
    ChDrive userdrive
    If Err=68 then
        MsgBox "Invalid Drive. Try again."
        Goto in1
    End If
    in2: On Error Goto Errhdlr1
    userdir=InputBox("Enter directory path:")
    ChDir userdrive & userdir
    MsgBox "New default directory is: " & userdrive & userdir
    Exit Sub
Errhdlr1:
    Select Case Err
        Case 75

```

```

        msgtext="Path is invalid."
    Case 76
        msgtext="Path not found."
    Case 70
        msgtext="Permission denied."
    Case Else
        msgtext="Error " & Err & ": " & Error$ & "occurred."
    End Select
    MsgBox msgtext & " Try again."
    Resume ln2
End Sub

```

See Also [Erl, Err Function, Err Statement, Error Function, Error Statement, Resume](#)

Open Statement

Action	Opens a file or device for input or output.										
Syntax	Open <i>filename\$</i> [For <i>mode</i>] [Access <i>access</i>] [<i>lock</i>] As [#] <i>filenumber%</i> [Len = <i>reclen</i>]										
where	is										
<i>filename\$</i>	a string or string expression for the name of the file to open.										
<i>mode</i>	one of the following keywords: <table> <tr><td>Input</td><td>Put data into the file sequentially.</td></tr> <tr><td>Output</td><td>Read data from the file sequentially.</td></tr> <tr><td>Append</td><td>Add data to the file sequentially.</td></tr> <tr><td>Random</td><td>Get data from the file by random access.</td></tr> <tr><td>Binary</td><td>Get binary data from the file.</td></tr> </table>	Input	Put data into the file sequentially.	Output	Read data from the file sequentially.	Append	Add data to the file sequentially.	Random	Get data from the file by random access.	Binary	Get binary data from the file.
Input	Put data into the file sequentially.										
Output	Read data from the file sequentially.										
Append	Add data to the file sequentially.										
Random	Get data from the file by random access.										
Binary	Get binary data from the file.										
<i>access</i>	one of the following keywords: <table> <tr><td>Read</td><td>Read data from the file only.</td></tr> <tr><td>Write</td><td>Write data the file only.</td></tr> <tr><td>Read Write</td><td>Read or write data to the file.</td></tr> </table>	Read	Read data from the file only.	Write	Write data the file only.	Read Write	Read or write data to the file.				
Read	Read data from the file only.										
Write	Write data the file only.										
Read Write	Read or write data to the file.										
<i>lock</i>	one of the following keywords to designate access by other processes: <table> <tr><td>Shared</td><td>Read or write available on the file.</td></tr> <tr><td>Lock Read</td><td>Read data only.</td></tr> <tr><td>Lock Write</td><td>Write data only.</td></tr> <tr><td>Lock Read Write</td><td>No read or write available.</td></tr> </table>	Shared	Read or write available on the file.	Lock Read	Read data only.	Lock Write	Write data only.	Lock Read Write	No read or write available.		
Shared	Read or write available on the file.										
Lock Read	Read data only.										
Lock Write	Write data only.										
Lock Read Write	No read or write available.										
<i>filenumber%</i>	an integer or expression containing the integer to assign to the open file (between 1 and 255).										
<i>reclen</i>	the length of the records (for Random or Binary files only).										

Comments A file must be opened before any input/output operation can be performed on it.

If *filename\$* does not exist, it is created when opened in **Append**, **Binary**, **Output** or **Random** modes.

If *mode* is not specified, it defaults to **Random**.

If *access* is not specified for **Random** or **Binary** modes, *access* is attempted in the following order: **Read Write**, **Write**, **Read**.

If *lock* is not specified, *filename\$* can be opened by other processes that do not specify a *lock*, although that process cannot perform any file operations on the file while the original process still has the file open.

Use the **FreeFile** function to find the next available value for *filenumber%*.

Reclen is ignored for **Input**, **Output**, and **Append** modes.

Example This example opens a file for Random access, gets the contents of the file, and closes the file again. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```

Declare Sub createfile()
Sub main
  Dim acctno as String*3
  Dim recno as Long
  Dim msgtext as String
  Call createfile
  recno=1
  newline=Chr(10)
  Open "C:\TEMP001" For Random As #1 Len=3
  msgtext="The account numbers are:" & newline
  Do Until recno=11
    Get #1,recno,acctno
    msgtext=msgtext & acctno
    recno=recno+1
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub
Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
  Close #1
End Sub

```

See Also [Close](#), [FreeFile](#)

OptionButton Statement

Action	Defines the position and text associated with an option button in a dialog box.
Syntax	OptionButton <i>x, y, dx, dy, text\$</i> [, <i>.id</i>]
where	is
<i>x,y</i>	the position of the button relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the button.
<i>text\$</i>	a string to display next to the option button. If the width of this string is greater than <i>dx</i> , trailing characters are truncated.
<i>.id</i>	an optional identifier used by the dialog statements that act on this control.

Comments You must have at least two **OptionButton** statements in a dialog box. You use these statements in conjunction with the **OptionGroup** statement.

A *dy* value of 12 typically accommodates text in the system font.

To enable the user to select an option button by typing a character from the keyboard, precede the character in *text\$* with an ampersand (&).

Use the **OptionButton** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example creates a dialog box with a group box with two option buttons: "All pages" and "Range of pages".

```
Sub main
  Begin Dialog UserDialog 183, 70, "SBL Dialog Box"
    GroupBox 5, 4, 97, 57, "File Range"
    OptionGroup .OptionGroup2
      OptionButton 16, 12, 46, 12, "All pages", .OptionButton3
      OptionButton 16, 28, 67, 8, "Range of pages", .OptionButton4
    Text 22, 39, 20, 10, "From:", .Text6
    Text 60, 39, 14, 9, "To:", .Text7
    TextBox 76, 39, 13, 12, .TextBox4
    TextBox 44, 39, 12, 11, .TextBox5
    OkButton 125, 6, 54, 14
    CancelButton 125, 26, 54, 14
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also [Begin...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OkButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

OptionGroup Statement

Action	Groups a series of option buttons under one heading in a dialog box.
Syntax	OptionGroup <i>.field</i> where <i>.field</i> is a value for the option button selected by the user: 0 for the first option button, 1 for the second button, and so on.
Comments	<p>The OptionGroup statement is used in conjunction with OptionButton statements to set up a series of related options. The OptionGroup Statement begins the definition of the option buttons and establishes the dialog-record field that will contain the option selection.</p> <p>Use the OptionGroup statement only between a Begin Dialog and an End Dialog statement.</p>
Example	<p>This example creates a dialog box with a group box with two option buttons: "All pages" and "Range of Pages".</p> <pre> Sub main Begin Dialog UserDialog 192, 71, "SBL Dialog Box" GroupBox 7, 6, 97, 57, "File Range" OptionGroup .OptionGroup2 OptionButton 18, 14, 46, 12, "All pages", .OptionButton3 OptionButton 18, 30, 67, 8, "Range of pages", .OptionButton4 Text 24, 41, 20, 10, "From:", .Text6 Text 62, 41, 14, 9, "To:", .Text7 TextBox 78, 41, 13, 12, .TextBox4 TextBox 46, 41, 12, 11, .TextBox5 OkButton 126, 6, 54, 14 CancelButton 126, 26, 54, 14 End Dialog Dim mydialog as UserDialog On Error Resume Next Dialog mydialog If Err=102 then MsgBox "Dialog box canceled." End If End Sub </pre>
See Also	Begin...End Dialog , Button , ButtonGroup , CancelButton , Caption , CheckBox , ComboBox , Dialog , DropComboBox , GroupBox , ListBox , OkButton , OptionButton , Picture , StaticComboBox , Text , TextBox

Option Base Statement

Action	Specifies the default lower bound to use for array subscripts.
Syntax	Option Base <i>lowerBound%</i> where <i>lowerBound</i> is a number or expression containing a number for the default lower bound: either 0 or 1.
Comments	If no Option Base statement is specified, the default lower bound for array subscripts will be 0.

The **Option Base** statement is *not* allowed inside a procedure, and must precede any use of arrays in the module. Only one **Option Base** statement is allowed per module.

Example

This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
        arrayvar(x)=InputBox("Enter test score #" & x & ":")
        x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
        count=InputBox("How many more do you have?")
        If count<>0 then
            count=count+(x-1)
            ReDim Preserve arrayvar(count)
            Goto start
        End If
    End If
    x=LBound(arrayvar,1)
    count=UBound(arrayvar,1)

    For y=x to count
        total=total+arrayvar(y)
    Next y
    MsgBox "The average of " & count & " scores is: " & Int(total/count)
End Sub
```

See Also [Dim](#), [Global](#), [Lbound](#), [ReDim](#), [Static](#)

Option Compare Statement

Action	Specifies the default method for string comparisons: either case-sensitive or case-insensitive.
Syntax	Option Compare { Binary Text } where Binary means comparisons are case-sensitive (i.e., lowercase and uppercase letters are different) and Text means comparisons are not case-sensitive.

Comments **Binary** comparisons compare strings based upon the ANSI character set. **Text** comparisons are based upon the relative order of characters as determined by the country code setting for your system.

Example This example compares two strings: "JANE SMITH" and "jane smith". When Option Compare is Text, the strings are considered the same. If Option Compare is Binary, they will not be the same. Binary is the default. To see the difference, run the example once, then run it again, commenting out the Option Compare statement.

```
Option Compare Text
Sub main
    Dim strg1 as String
    Dim strg2 as String
    Dim retval as Integer
    strg1="JANE SMITH"
    strg2="jane smith"
i:
    retval=StrComp(strg1,strg2)
    If retval=0 then
        MsgBox "The strings are identical"
    Else
        MsgBox "The strings are not identical"
    Exit Sub
End If
End Sub
```

See Also [Instr](#), [StrComp](#)

Option Explicit Statement

Action Specifies that all variables in a module *must* be explicitly declared.

Syntax **Option Explicit**

Comments By default, Basic automatically declares any variables that do not appear in a Dim, Global, Redim, or Static statement. Option Explicit causes such variables to produce a "Variable Not Declared" error.

Example This example specifies that all variables must be explicitly declared, thus preventing any mistyped variable names.

```
Option Explicit
Sub main
    Dim counter As Integer
    Dim fixedstring As String*25
    Dim varstring As String
    '...(code here)...
End Sub
```

See Also [Const](#), [DefType](#), [Dim](#), [Function...End Function](#), [Global](#), [ReDim](#), [Static](#), [Sub...End Sub](#)

PasswordBox Function

Action Returns a string entered by the user without echoing it to the screen.

Syntax **PasswordBox**[\$](*prompt*\$,[*title*\$] ,[*default*\$] [,*xpos*%, *ypos*%])

where **is**
prompt\$ a string expression containing the text to show in the dialog box
title\$ the caption for the dialog box's title bar
default\$ the string expression shown in the edit box as the default response.
xpos%, *ypos*% the position of the dialog box, relative to the upper left corner of the screen.

Comments The **PasswordBox** function displays a dialog box containing a prompt. Once the user has entered text, or made the button choice being prompted for, the contents of the box are returned.

The length of *prompt*\$ is restricted to 255 characters. This figure is approximate and depends on the width of the characters used. Note that a carriage return and a line-feed character must be included in *prompt*\$ if a multiple-line prompt is used.

If either *prompt*\$ or *default*\$ is omitted, nothing is displayed.

Xpos% determines the horizontal distance between the left edge of the screen and the left border of the dialog box, measured in dialog box units. *Ypos*% determines the horizontal distance from the top of the screen to the dialog box's upper edge, also in dialog box units. If these arguments are not entered, the dialog box is centered roughly one third of the way down the screen. A horizontal dialog box unit is 1/4 of the average character width in the system font; a vertical dialog box unit is 1/8 of the height of a character in the system font.



To specify the dialog box's position, you must enter both of these arguments. If you enter one without the other, the default positioning is used.

Once the user presses Enter, or selects the OK button, **PasswordBox** returns the text contained in the password box. If the user selects Cancel, the **PasswordBox** function returns a null string ("").

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

Example

This example asks the user for a password.

```
Sub main
  Dim retvalue
  Dim a
  retvalue=PasswordBox("Enter your login password",Password)
  If retvalue<>"" then
```



```
        MsgBox "Verifying password"
    ' (continue code here)
Else
    MsgBox "Login cancelled"
End If
End Sub
```

See Also [InputBox](#), [MsgBox Function](#), [MsgBox Statement](#)

Picture Statement

Action	Defines a picture control in a dialog box.												
Syntax	Picture <i>x, y, dx, dy, filename\$, type</i> [, <i>.id</i>]												
	<table><tr><td>where</td><td>is</td></tr><tr><td><i>x,y</i></td><td>the position of the picture relative to the upper left corner of the dialog box.</td></tr><tr><td><i>dx,dy</i></td><td>the width and height of the picture.</td></tr><tr><td><i>filename\$</i></td><td>the name of the bitmap file (a file with .BMP extension) where the picture is located.</td></tr><tr><td><i>type</i></td><td>an integer for the location of the bitmap (0=<i>filename\$</i>, 3=Windows Clipboard).</td></tr><tr><td><i>.id</i></td><td>an optional identifier used by the dialog statements that act on this control.</td></tr></table>	where	is	<i>x,y</i>	the position of the picture relative to the upper left corner of the dialog box.	<i>dx,dy</i>	the width and height of the picture.	<i>filename\$</i>	the name of the bitmap file (a file with .BMP extension) where the picture is located.	<i>type</i>	an integer for the location of the bitmap (0= <i>filename\$</i> , 3=Windows Clipboard).	<i>.id</i>	an optional identifier used by the dialog statements that act on this control.
where	is												
<i>x,y</i>	the position of the picture relative to the upper left corner of the dialog box.												
<i>dx,dy</i>	the width and height of the picture.												
<i>filename\$</i>	the name of the bitmap file (a file with .BMP extension) where the picture is located.												
<i>type</i>	an integer for the location of the bitmap (0= <i>filename\$</i> , 3=Windows Clipboard).												
<i>.id</i>	an optional identifier used by the dialog statements that act on this control.												
Comments	The Picture statement can only be used between a Begin Dialog and an End Dialog statement.												
Note	<p>The picture will be scaled equally in both directions and centered if the dimensions of the picture are not proportional to <i>dx</i> and <i>dy</i>.</p> <p>If <i>type%</i> is 3, <i>filename\$</i> is ignored.</p> <p>If the picture is not available (the file <i>filename\$</i> doesn't exist, doesn't contain a bitmap, or there is no bitmap on the Clipboard), the picture control will display the picture frame and the text "(missing picture)". This behavior may be changed by adding 16 to the value of <i>type%</i>. If <i>type%</i> is 16 or 19 and the picture is not available, a runtime error occurs.</p>												
Example	<p>This example defines a dialog box with a picture, and the OK and Cancel buttons.</p> <pre>Sub main Begin Dialog UserDialog 148, 73, "SBL Dialog Box" Picture 8, 7, 46, 46, "C:\WINDOWS\ARCADE.BMP", 0 OkButton 80, 10, 54, 14 CancelButton 80, 30, 54, 14 End Dialog</pre>												

```

Dim mydialog as UserDialog
On Error Resume Next
Dialog mydialog
If Err=102 then
    MsgBox "Dialog box canceled."
End If
End Sub

```

See Also [Begin...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OkButton](#), [OptionButton](#), [OptionGroup](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Pmt Function

Action Returns a constant periodic payment amount for an annuity or a loan.

Syntax **Pmt** (*rate*, *nper*, *pv*, *fv*, *due*)

where **is**

rate interest rate per period.

nper total number of payment periods.

pv present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).

fv future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).

due an integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the monthly payment on a given loan.

```

Sub main
Dim aprate, totalpay
Dim loanpv, loanfv
Dim due, monthlypay
Dim yearllypay, msgtext
loanpv=InputBox("Enter the loan amount: ")
aprate=InputBox("Enter the loan rate percent: ")
If aprate >1 then
    aprate=aprate/100
End If
totalpay=InputBox("Enter the total number of monthly payments: ")
loanfv=0
'Assume payments are made at end of month
due=0
monthlypay=Pmt(aprate/12,totalpay,-loanpv,loanfv,due)
msgtext="The monthly payment is: " & Format(monthlypay, "Currency")
MsgBox msgtext
End Sub

```

See Also [FV](#), [Ipmt](#), [IRR](#), [NPV](#), [PV](#), [Ppmt](#), [Rate](#)

PPmt Function

Action Returns the principal portion of the payment for a given period of an annuity.

Syntax **PPmt** (*rate*, *per*, *nper*, *pv*, *fv*, *due*)

where	is
<i>rate</i>	interest rate per period.
<i>per</i>	particular payment period in the range 1 through <i>nper</i> .
<i>nper</i>	total number of payment periods.
<i>pv</i>	present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
<i>fv</i>	future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
<i>due</i>	an integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the principal portion of a loan payment amount for payments made in last month of the first year. The loan is for \$25,000 to be paid back over 5 years at 9.5% interest.

```
Sub main
  Dim aprate, periods
  Dim payperiod
  Dim loanpv, due
  Dim loanfv, principal
  Dim msgtext
  aprate=9.5/100
  payperiod=12
  periods=120
  loanpv=25000
  loanfv=0
  Rem Assume payments are made at end of month
  due=0
  principal=PPmt(aprate/12,payperiod,periods,-loanpv,loanfv,due)
  msgtext="Given a loan of $25,000 @ 9.5% for 10 years," & Chr(10)
  msgtext=msgtext & " the principal paid in month 12 is: "
  MsgBox msgtext & Format(principal, "Currency")
End Sub
```

See Also [FV](#), [Ipmt](#), [IRR](#), [NPV](#), [Pmt](#), [PV](#), [Rate](#)

Print Statement

Action Prints data to an open file or to the screen.

Syntax **Print** [*filenumber%*,] *expressionlist* [{ ; | , }] where *filenumber%* is an integer expression identifying the open file to use and *expressionlist* is a numeric, string, and Variant expression containing the list of values to print.

Comments The **Print** statement outputs data to the specified *filenumber%*. *filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information. If this argument is omitted, the **Print** statement outputs data to the screen.

If the *expressionlist* is omitted, a blank line is written to the file.

The values in *expressionlist* are separated by either a semi-colon (“;”) or a comma (“,”). A semi-colon indicates that the next value should appear immediately after the preceding one without intervening white space. A comma indicates that the next value should be positioned at the next print zone. Print zones begin every 14 spaces.

The optional [{;|,}] argument at the end of the **Print** statement determines where output for the next **Print** statement to the same output file should begin. A semi-colon will place output immediately after the output from this **Print** statement on the current line; a comma will start output at the next print zone on the current line. If neither separator is specified, a CR-LF pair will be generated and the next **Print** statement will print to the next line.

Special functions **Spc** and **Tab** can be used inside **Print** statement to insert a given number of spaces and to move the print position to a desired column.

The **Print** statement supports only elementary Basic data types. See **Input** for more information on parsing this statement.

Example This example prints the octal values for the numbers from 1 to 25.

```
Sub main
    Dim x as Integer
    Dim y
    For x=1 to 25

        y=Oct$(x)
        Print x Tab(10) y
    Next x
End Sub
```

See Also [Open](#), [Spc](#), [Tab](#), [Write](#)

PushButton Statement

Action	Defines a custom pushbutton.										
Syntax A	PushButton <i>x, y, dx, dy, text\$</i> [, <i>.id</i>]										
Syntax B	Button <i>x, y, dx, dy, text\$</i> [, <i>.id</i>]										
	<table> <tr> <td>where</td><td>is</td></tr> <tr> <td><i>x,y</i></td><td>the position of the button relative to the upper left corner of the dialog box.</td></tr> <tr> <td><i>dx,dy</i></td><td>the width and height of the button.</td></tr> <tr> <td><i>text\$</i></td><td>the name for the pushbutton. If the width of this string is greater than <i>dx</i>, trailing characters are truncated.</td></tr> <tr> <td><i>.id</i></td><td>an optional identifier used by the dialog statements that act on this control.</td></tr> </table>	where	is	<i>x,y</i>	the position of the button relative to the upper left corner of the dialog box.	<i>dx,dy</i>	the width and height of the button.	<i>text\$</i>	the name for the pushbutton. If the width of this string is greater than <i>dx</i> , trailing characters are truncated.	<i>.id</i>	an optional identifier used by the dialog statements that act on this control.
where	is										
<i>x,y</i>	the position of the button relative to the upper left corner of the dialog box.										
<i>dx,dy</i>	the width and height of the button.										
<i>text\$</i>	the name for the pushbutton. If the width of this string is greater than <i>dx</i> , trailing characters are truncated.										
<i>.id</i>	an optional identifier used by the dialog statements that act on this control.										
Comments	<p>A <i>dy</i> value of 14 typically accommodates text in the system font.</p> <p>Use this statement to create buttons other than OK and Cancel. Use this statement in conjunction with the ButtonGroup statement. The two forms of the statement (Button and PushButton) are equivalent.</p> <p>Use the Button statement only between a Begin Dialog and an End Dialog statement.</p>										
Example	<p>This example defines a dialog box with a combination list box and three buttons.</p> <pre> Sub main Dim fchoices as String fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3" Begin Dialog UserDialog 185, 94, "SBL Dialog Box" Text 9, 5, 69, 10, "Filename:", .Text1 DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1 ButtonGroup .ButtonGroup1 OkButton 113, 14, 54, 13 CancelButton 113, 33, 54, 13 PushButton 113, 57, 54, 13, "Help", .Push1 End Dialog Dim mydialog as UserDialog On Error Resume Next Dialog mydialog If Err=102 then MsgBox "Dialog box canceled." End If End Sub </pre>										

See Also [Begin Dialog...End Dialog Statement](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [DropComboBox](#), [DropListBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Put Statement

Action Writes a variable to a file opened in **Random** or **Binary** mode.

Syntax **Put** [#]*filename%*, [*recnumber&*], *varname*

where **is**
filename% an integer expression identifying the open file to use.
recnumber& a **Long** expression containing the record number or the byte offset
 at which to start writing.
varname the name of the variable containing the data to write.

Comments *Filename%* is the number assigned to the file when it was opened. See the **Open** statement for more information.

Recnumber& is in the range 1 to 2,147,483,647. If *recnumber&* is omitted, the next record or byte is written.

+ *The commas before and after recnumber% are **required**, even if no recnumber& is specified.*

Varname can be any variable except **Object**, **Application Data Type** or **Array** variables (single array elements may be used).

For **Random** mode, the following apply:

- Blocks of data are written to the file in chunks whose size is equal to the size specified in the **Len** clause of the **Open** statement. If the size of *varname* is smaller than the record length, the record is padded to the correct record size. If the size of variable is larger than the record length, an error occurs.
- For variable length String variables, **Put** writes two bytes of data that indicate the length of the string, then writes the string data.
- For Variant variables, **Put** writes two bytes of data that indicate the type of the Variant, then it writes the body of the Variant into the variable. Note that Variants containing strings contain two bytes of type information, followed by two bytes of length, followed by the body of the string.
- User defined types are written as if each member were written separately, except no padding occurs between elements.

Files opened in **Binary** mode behave similarly to those opened in **Random** mode except:

- **Put** writes variables to the disk without record padding.
- Variable length **Strings** that are not part of user defined types are not preceded by the two byte string length.

Example

This example opens a file for Random access, puts the values 1-10 in it, prints the contents, and closes the file again.

```
Sub main
' Put the numbers 1-10 into a file
Dim x, y
Open "C:\TEMP001" as #1
For x=1 to 10
    Put #1,x, x
Next x
msgtext="The contents of the file is:" & Chr(10)
For x=1 to 10
    Get #1,x, y
    msgtext=msgtext & y & Chr(10)
Next x
Close #1
MsgBox msgtext
Kill "C:\TEMP001"
End Sub
```

See Also [Close](#), [Get](#), [Open](#), [Write](#)

PV Function

Action	Returns the present value of a constant periodic stream of cash flows as in an annuity or a loan.
Syntax	PV (<i>rate</i> , <i>nper</i> , <i>pmt</i> , <i>fv</i> , <i>due</i>)
where	is
<i>rate</i>	interest rate per period.
<i>nper</i>	total number of payment periods.
<i>pmt</i>	constant periodic payment per period.
<i>fv</i>	future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).
<i>due</i>	an integer value for when the payments are due (0=end of each period, 1= beginning of the period).
Comments	<i>Rate</i> is assumed constant over the life of the annuity. If payments are on a monthly schedule, then <i>rate</i> will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the present value of a 10-year \$25,000 annuity that will pay \$1,000 a year at 9.5%.

```
Sub main
  Dim aprate, periods
  Dim payment, annuityfv
  Dim due, presentvalue
  Dim msgtext
  aprate=9.5
  periods=120
  payment=1000
  annuityfv=25000
  Rem Assume payments are made at end of month
  due=0
  presentvalue=PV(aprate/12,periods,-payment, annuityfv,due)
  msgtext= "The present value for a 10-year $25,000 annuity @ 9.5%"
  msgtext=msgtext & " with a periodic payment of $1,000 is: "
  msgtext=msgtext & Format(presentvalue, "Currency")
  MsgBox msgtext
End Sub
```

See Also [FV](#), [Ipmt](#), [IRR](#), [NPV](#), [Pmt](#), [Ppmt](#), [Rate](#)

Randomize Statement

Action Seeds the random-number generator.

Syntax **Randomize** [*number%*] where *number%* is an integer value between -32768 and 32767.

Comments If no *number%* argument is given, Basic uses the **Timer** function to initialize the random number generator.

Example This example generates a random string of characters using the Randomize statement and Rnd function. The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub main
  Dim x as Integer
  Dim y
  Dim str1 as String
  Dim str2 as String
  Dim letter as String
  Dim randomvalue
  Dim upper, lower
  Dim msgtext
  upper=Asc("z")
  lower=Asc("a")
  newline=Chr(10)
  For x=1 to 26
    Randomize
    randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
    letter=Chr(randomvalue)
    str1=str1 & letter
```



```

For y = 1 to 1500
    Next y
Next x
msgtext=str1
MsgBox msgtext
End Sub

```

See Also [Rnd](#), [Timer](#)

Rate Function

Action Returns the interest rate per period for an annuity or a loan.

Syntax **Rate** (*nper*, *pmt*, *pv*, *fv*, *due*, *guess*)

where	is
<i>nper</i>	total number of payment periods.
<i>pmt</i>	constant periodic payment per period.
<i>pv</i>	present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
<i>fv</i>	future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).
<i>due</i>	an integer value for when the payments are due (0=end of each period, 1= beginning of the period)
<i>guess</i>	a ballpark estimate for the rate returned.

Comments In general, a guess of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable value for *guess*.

Rate is an iterative function: it improves the given value of *guess* over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

Example This example finds the interest rate on a 10 year \$25,000 annuity, that pays \$100 per month.

```

Sub main
    Dim aprate
    Dim periods
    Dim payment, annuitypv
    Dim annuityfv, due
    Dim guess
    Dim msgtext as String
    periods=120
    payment=100
    annuitypv=0
    annuityfv=25000
    guess=.1
    Rem Assume payments are made at end of month
    due=0
    aprate=Rate(periods,-payment,annuitypv,annuityfv, due, guess)

```

```

aprate=(aprate*12)
msgtext= "The percentage rate for a 10-year $25,000 annuity "
msgtext=msgtext & "that pays $100/month has "
msgtext=msgtext & "a rate of: " & Format(aprate, "Percent")
MsgBox msgtext
End Sub

```

See Also [FV](#), [Ipmt](#), [IRR](#), [NPV](#), [Pmt](#), [Ppmt](#), [PV](#)

ReDim Statement

Action Changes the upper and lower bounds of a dynamic array's dimensions.

Syntax **ReDim** [**Preserve**] *variableName* (*subscriptRange*, ...) [**As** [**New**] *type*] , ...

where	is
<i>variableName</i>	the variable array name to redimension.
<i>subscriptRange</i>	the new upper and lower bounds for the array.
<i>type</i>	the type for the data elements in the array.

Comments **ReDim** re-allocates memory for the dynamic array to support the specified dimensions, and may optionally re-initialize the array elements. **ReDim** cannot be used at the module level; it must be used inside of a procedure.

The **Preserve** option is used to change the last dimension in the array while maintaining its contents. If **Preserve** is not specified, the contents of the array are re-initialized. Numbers will be set to zero (0). Strings and Variants will be set to empty ("").

The *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

A dynamic array is normally created by using **Dim** to declare an array without a specified *subscriptRange*. The maximum number of dimensions for a dynamic array created in this fashion is 8. If you need more than 8 dimensions, you may use the **ReDim** statement inside of a procedure to declare an array which has not previously been declared using **Dim** or **Global**. In this case, the maximum number of dimensions allowed is 60.

The available data types for arrays are: numbers, strings, Variants, records and objects. Arrays of arrays, dialog box records, and objects are not supported.

If the **As** clause is not used, the type of the variable may be specified by using a type character as a suffix to the name. The two different type-specification methods can be intermixed in a single **ReDim** statement (although not on the same variable).

The **ReDim** statement cannot be used to change the number of dimensions of a dynamic array once the array has been given dimensions. It can only change the upper and lower bounds of the dimensions of the array. The **LBound** and **UBound** functions can be used to query the current bounds of an array variable's dimensions.

Care should be taken to avoid **ReDim**'ing an array in a procedure that has received a reference to an element in the array in an argument; the result is unpredictable.

Example

This example finds the net present value for a series of cash flows. The array variable that holds the cash flow amounts is initially a dynamic array that is redimensioned after the user enters the number of cash flow periods they have.

```
Sub main
  Dim aprate as Single
  Dim varray() as Double
  Dim cflowper as Integer
  Dim x as Integer
  Dim netpv as Double
  cflowper=InputBox("Enter number of cash flow periods:")
  ReDim varray(cflowper)
  For x= 1 to cflowper
    varray(x)=InputBox("Enter cash flow amount for period #" &x &":")
  Next x
  aprate=InputBox ("Enter discount rate:")
  If aprate>1 then
    aprate=aprate/100
  End If
  netpv=NPV(aprate,varray())
  MsgBox "The Net Present Value is: " & Format(netpv,"Currency")
End Sub
```

See Also [Dim](#), [Global](#), [Option Base](#), [Static](#)

Rem Statement

Action	Identifies a line of code as a comment in a Basic program.
Syntax	Rem <i>comment</i> where <i>comment</i> is the text of the comment.
Comments	Everything from Rem to the end of the line is ignored. The single quote (') can also be used to initiate a comment. Metacommands (e.g., \$CSTRINGS) must be preceded by the single quote comment form.
Example	This example defines a dialog box with a combination list box and two buttons. The Rem statements describe each block of definition code.

```

Sub main
  Dim fchoices as String
  fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
  Begin Dialog UserDialog 185, 94, "SBL Dialog Box"
Rem The next two lines create the combo box
  Text 9, 5, 69, 10, "Filename:", .Text1
  DropComboBox 9, 17, 88, 71, fchoices, .ComboBox1
Rem The next two lines create the command buttons
  OkButton 113, 14, 54, 13
  CancelButton 113, 33, 54, 13
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub

```

Reset Statement

Action	Closes all open disk files and writes any data in the operating system buffers to disk.
Syntax	Reset
Example	This example creates a file, puts the numbers 1-10 in it, then attempts to Get past the end of the file. The On Error statement traps the error and execution goes to the Debugger code which uses Reset to close the file before exiting.

```

Sub main
' Put the numbers 1-10 into a file
  Dim x as Integer
  Dim y as Integer
  On Error Goto Debugger
  Open "C:\TEMP001" as #1 Len=2
  For x=1 to 10
    Put #1,x, x
  Next x
  Close #1
  msgtext="The contents of the file is:" & Chr(10)
  Open "C:\TEMP001" as #1 Len=2
  For x=1 to 10
    Get #1,x, y
    msgtext=msgtext & Chr(10) & y
  Next x
  MsgBox msgtext
done:
  Close #1
  Kill "C:\TEMP001"
  Exit Sub

Debugger:
  MsgBox "Error " & Err & " occurred. Closing open file."
Reset
  Resume done
End Sub

```

See Also [Close](#)

Resume Statement

Action	Halts an error-handling routine.
Syntax A	Resume Next
Syntax B	Resume <i>label</i>
Syntax C	Resume [0]
	where <i>label</i> is the label that identifies the statement to go to after handling an error.
Comments	<p>When the Resume Next statement is used, control is passed to the statement which immediately follows the statement in which the error occurred.</p> <p>When the Resume [0] statement is used, control is passed to the statement in which the error occurred.</p> <p>The location of the error handler which has caught the error determines where execution will resume. If an error is trapped in the same procedure as the error handler, program execution will resume with the statement that caused the error. If an error is located in a different procedure from the error handler, program control reverts to the statement that last called out the procedure containing the error handler.</p>
Example	<p>This example prints an error message if an error occurs during an attempt to open a file. The Resume statement jumps back into the program code at the label, done. From here, the program exits.</p> <pre> Sub main Dim msgtext, userfile On Error GoTo Debugger msgtext="Enter the filename to use:" userfile=InputBox\$(msgtext) Open userfile For Input As #1 MsgBox "File opened for input." 'etc.... Close #1 done: Exit Sub Debugger: msgtext="Error number " & Err & " occurred at line: " & Erl MsgBox msgtext Resume done End Sub </pre>

See Also [Erl](#), [Err Function](#), [Err Statement](#), [Error](#), [Error Function](#), [On Error](#), [Trappable Errors](#)

Right Function

Action	Returns a string copied from the rightmost characters in a specified string.
Syntax	Right [\$](<i>string</i> , <i>length%</i>) where <i>string</i> is any type of expression that contains the string to copy and <i>length%</i> is an integer for the number of characters to copy from <i>expression</i> .
Comments	<p>If the length of <i>expression</i> is less than <i>length%</i>, Right returns the whole string.</p> <p>Right accepts any type of <i>expression</i>, including numeric values, and will convert the input value to a string.</p> <p>The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will typically return a Variant of vartype 8 (string). If the value of <i>expression</i> is NULL, a Variant of vartype 1 (Null) is returned.</p>
Example	<p>This example checks for the extension .BMP in a filename entered by a user and activates the Paintbrush application if the file is found. Note this uses the Option Compare statement to accept either uppercase or lowercase letters for the filename extension.</p> <pre> Option Compare Text Sub main Dim filename as String Dim x filename=InputBox("Enter a .BMP file and path: ") extension=Right(filename,3) If extension="BMP" then x=Shell("PBRUSH.EXE",1) Sendkeys "%FO" & filename & "{Enter}", 1 Else MsgBox "File not found or extension not .BMP." End If End Sub </pre>
See Also	GetField , Instr , Left , Len , Ltrim , Mid Function , Mid Statement , Rtrim , Trim

Rmdir Statement

Action	Removes a directory.
Syntax	Rmdir <i>path\$</i> where <i>path\$</i> is a string expression identifying the directory to remove.
Comments	<p>The syntax for <i>path\$</i> is:</p> <p style="text-align: center;">[<i>drive:</i>] [\] <i>directory</i> [<i>directory</i>]</p> <p>The <i>drive</i> argument is optional. The <i>directory</i> argument is a directory name.</p>

The directory to be removed must be empty, except for the working (.) and parent (..) directories.

Example This example makes a new temporary directory in C:\ and then deletes it.

```
Sub main
  Dim path as String
  On Error Resume Next
  path=CurDir(C)
  If path<>"C:\" then
    ChDir "C:\"
  End If
  Mkdir "C:\TEMP01"
  If Err=75 then
    MsgBox "Directory already exists"
  Else
    MsgBox "Directory C:\TEMP01 created"
    MsgBox "Now removing directory"
    Rmdir "C:\TEMP01"
  End If
End Sub
```

See Also [ChDir](#), [ChDrive](#), [CurDir](#), [Dir](#), [Mkdir](#)

Rnd Function

Action	Returns a single precision random number between 0 and 1.
Syntax	Rnd [(<i>number!</i>)] where <i>number!</i> is a numeric expression to specify how to generate the random numbers. (<0=use the number specified, >0=use the next number in the sequence, 0=use the number most recently generated.)
Comments	If <i>number!</i> is omitted, Rnd uses the next number in the sequence to generate a random number. The same sequence of random numbers is generated whenever Rnd is run, unless the random number generator is re-initialized by the Randomize statement.
Example	This example generates a random string of characters within a range. The Rnd function is used to set the range between lowercase "a" and "z". The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub main
  Dim x as Integer
  Dim y
  Dim str1 as String
  Dim str2 as String
  Dim letter as String
  Dim randomvalue
  Dim upper, lower
  Dim msgtext
  upper=Asc("z")
  lower=Asc("a")
  newline=Chr(10)
```

```

For x=1 to 26
  Randomize
  randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
  letter=Chr(randomvalue)
  str1=str1 & letter
  For y = 1 to 1500
    Next y
  Next x
  msgtext=str1
  MsgBox msgtext
End Sub

```

See Also [Exp](#), [FixInt](#), [Int](#), [Log](#), [Randomize](#), [Sgn](#), [Sqr](#)

Rset Statement

Action	Right-aligns one string inside another string.
Syntax	Rset <i>string\$</i> = <i>string-expression</i> where <i>string\$</i> is the string to contain the right-aligned characters and <i>string-expression</i> is the string containing the characters to put into <i>string\$</i> .
Comments	<p>If <i>string\$</i> is longer than <i>string-expression</i>, the leftmost characters of <i>string\$</i> are replaced with spaces.</p> <p>If <i>string\$</i> is shorter than <i>string-expression</i>, only the leftmost characters of <i>string-expression</i> are copied.</p> <p>Rset cannot be used to assign variables of different user-defined types.</p>
Example	This example uses Rset to right align an amount entered by the user in a field that is 15 characters long. It then pads the extra spaces with asterisks (*) and adds a dollar sign (\$) and decimal places (if necessary).

```

Sub main
  Dim amount as String*15
  Dim x
  Dim msgtext
  Dim replacement
  replacement=""
  amount=InputBox("Enter an amount:")
  position=InStr(amount,".")
  If Right(amount,3)<>".00" then
    amount=Rtrim(amount) & ".00"
  End If
  Rset amount="$" & Rtrim(amount)
  length=15-Len(Ltrim(amount))
  For x=1 to length
    Mid(amount,x)=replacement
  Next x
  MsgBox "Formatted amount: " & amount
End Sub

```

See Also [Lset](#)

RTrim Function

Action	Copies a string and removes any trailing spaces.
Syntax	RTrim [\$](<i>string</i> \$) where <i>string</i> \$ is an expression that evaluates to a string.
Comments	<p>RTrim accepts any type of <i>string</i> including numeric values and will convert the input value to a string.</p> <p>The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted the function will typically return a Variant of vartype 8 (string). If the value of <i>string</i> is NULL, a Variant of vartype 1 (Null) is returned.</p>
Example	<p>This example asks for an amount and then right aligns it in a field that is 15 characters long. It uses Rtrim to trim any trailing spaces in the amount string, if the number entered by the user is less than 15 digits.</p>

```

Sub main
  Dim amount as String*15
  Dim x
  Dim msgtext
  Dim replacement
  replacement="X"
  amount=InputBox("Enter an amount:")
  position=InStr(amount, ".")
  If position=0 then
    amount=Rtrim(amount) & ".00"
  End If
  Rset amount="$" & Rtrim(amount)
  length=15-Len(Ltrim(amount))
  For x=1 to length
    Mid(amount,x)=replacement
  Next x
  MsgBox "Formatted amount: " & amount
End Sub

```

See Also [GetField](#), [Left](#), [Len](#), [Ltrim](#), [Mid Function](#), [Mid Statement](#), [Right](#), [Trim](#)

Second Function

Action	Returns the second component (0-59) of a date-time value.
Syntax	Second (<i>time</i>) where <i>time</i> is an expression containing a date time value.
Comments	<p>Second accepts any type of <i>time</i> including strings and will attempt to convert the input value to a date value.</p> <p>The return value is a Variant of vartype 2 (integer). If the value of <i>time</i> is NULL, a Variant of vartype 1 (Null) is returned.</p>

Example This example displays the last saved date and time for a file whose name is entered by the user.

```
Sub main
  Dim filename as String
  Dim ftime
  Dim hr, min
  Dim sec
  Dim msgtext as String
i: msgtext="Enter a filename:"
  filename=InputBox(msgtext)
  If filename="" then
    Exit Sub
  End If
  On Error Resume Next
  ftime=FileDateTime(filename)
  If Err<>0 then
    MsgBox "Error in file name. Try again."
    Goto i:
  End If
  hr=Hour(ftime)
  min=Minute(ftime)
  sec=Second(ftime)
  MsgBox "The file's time is: " & hr & ":" & min & ":" & sec
End Sub
```

See Also [Day, Hour, Minute, Month, Now, Time Function, Time Statement, Weekday, Year](#)

Seek Function

Action Returns the current file position for an open file.

Syntax **Seek**(*filenumber%*) where *filenumber%* is an integer expression identifying an open file to query.

Comments *Filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information.

For files opened in **Random** mode, **Seek** returns the number of the next record to be read or written. For all other modes, **Seek** returns the file offset for the next operation. The first byte in the file is at offset 1, the second byte is at offset 2, etc. The return value is a **Long**.

Example This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, CREATEFILE, creates the file "C:\TEMP001" used by the main subprogram.

```
Declare Sub createfile
Sub main
  Dim testscore as String
  Dim x
  Dim y
  Dim newline
```

```

Call createfile
Open "C:\TEMP001" for Input as #1
x=1
newline=Chr(10)
msgtext= "The test scores are: " & newline
Do Until x=Lof(1)
    Line Input #1, testscore
    x=x+1
    y=Seek(1)
    If y>Lof(1) then
        x=Lof(1)
    Else
        Seek 1,y
    End If
    msgtext=msgtext & newline & testscore
Loop
MsgBox msgtext
Close #1
Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 10-100 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=10 to 100 step 10

        Write #1, x
    Next x
    Close #1
End Sub

```

See Also [Get, Open, Put, Seek Statement](#)

Seek Statement

Action	Sets the position within an open file for the next read or write operation.
Syntax	Seek [#] <i>filenumber%</i> , <i>position&</i> where <i>filenumber%</i> is an integer expression identifying an open file to query <i>position&</i> is a numeric expression for the starting position of the next read or write operation (record number or byte offset).
Comments	<p>The Seek statement. If you write to a file after seeking beyond the end of the file, the file's length is extended. Basic will return an error message if a Seek operation is attempted which specifies a negative or zero position.</p> <p><i>Filenumber%</i> is an integer expression identifying the open file to Seek in. See the Open statement for more details.</p> <p>For files opened in Random mode, <i>position&</i> is a record number; for all other modes, <i>position&</i> is a byte offset. <i>Position&</i> is in the range 1 to 2,147,483,647. The first byte or record in the file is at position 1, the second is at position 2, etc.</p>

Syntax

```
Select Case testexpression
[Case expressionlist
    [statement_block] ]
[Case expressionlist
    [statement_block] ]
.
```

```

[Case Else
    [statement_block] ]
End Select

where      is
testexpression  any expression containing a variable to test.
expressionlist  one or more expressions that contain a possible value for
                testexpression .
statement_block the statements to execute if testexpression equals expressionlist.

```

Comments

When there is a match between *testexpression* and one of the values in *expressionlist*, the *statement_block* following the **Case** clause is executed. When the next **Case** clause is reached, execution control goes to the statement following the **End Select** statement.

The *expressionlist(s)* may be a comma-separated list of expressions of the following forms:

```

expression
expression To expression
Is comparison_operator expression

```

The type of each *expression* must be compatible with the type of *testexpression*.

Note that when the **To** keyword is used to specify a range of values, the smaller value must appear first. The *comparison_operator* used with the **Is** keyword is one of: <, >, =, <=, >=, <>.

Each *statement_block* can contain any number of statements on any number of lines.

Example

This example tests the attributes for a file and if it is hidden, changes it to a non-hidden file.

```

Sub main
    Dim filename as String
    Dim attribs, saveattribs as Integer
    Dim answer as Integer
    Dim archno as Integer
    Dim msgtext as String
    archno=32
    On Error Resume Next
    msgtext="Enter name of a file:"
    filename=InputBox(msgtext)
    attribs=GetAttr(filename)
    If Err<>0 then
        MsgBox "Error in filename. Re-run Program."
        Exit Sub
    End If
    saveattribs=attribs
    If attribs>= archno then
        attribs=attribs-archno
    End If

```

```

Select Case attribs
Case 2,3,6,7
  msgtext=" File: " &filename & " is hidden." & Chr(10)
  msgtext=msgtext & Chr(10) & " Change it?"
  answer=Msgbox(msgtext,308)
  If answer=6 then
    SetAttr filename, saveattribs-2
    MsgBox "File is no longer hidden."
    Exit Sub
  End If
  MsgBox "Hidden file not changed."
Case Else
  MsgBox "File was not hidden."
End Select
End Sub

```

See Also [If...Then...Else, On...Goto, Option Compare](#)

SendKeys Statement

Action Send keystrokes to an active Windows application.

Syntax **SendKeys** *string\$* [, *wait%*] where *string\$* is an expression containing the characters to send and *wait%* is a numeric expression to determine whether to wait until all keys are processed before continuing program execution (-1=wait, 0=don't wait).

Comments The keystrokes are represented by characters of *string*.

The default value for *wait* is 0 (FALSE).

To specify an ordinary character, enter this character in the *string*. For example, to send character 'a' use "a" as *string*. Several characters may be combined in one string: *string* "abc" means send 'a', 'b', and 'c'.

To specify that Shift, Alt, or Control keys should be pressed simultaneously with a character, prefix the character with

+	to specify Shift
%	to specify Alt
^	to specify Control.

Parentheses may be used to specify that the Shift, Alt, or Control key should be pressed with a group of characters. For example, "%(abc)" is equivalent to "%a%b%c".

Since '+', '%', '^', '(' and ')' characters have special meaning to **SendKeys**, they must be enclosed in braces if they need to be sent with **SendKeys**. For example *string* "{%}" specifies a percent character '%'.

The other characters that need to be enclosed in braces are '~' which stands for a newline or “Enter” if used by itself and braces themselves: use {{ } to send '{' and { } } to send '}'. Brackets '[' and ']' do not have special meaning to **SendKeys** but may have special meaning in other applications, therefore, they need to be enclosed inside braces as well.

To specify that a key needs to be sent several times, enclose the character in braces and specify the number of keys sent after a space: for example, use {X 20} to send 20 'X' characters.

To send one of the nonprintable keys use a special keyword inside braces:

Key	Keyword
Backspace	{BACKSPACE} or {BKSP} or {BS}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Delete	{DELETE} or {DEL}
Down Arrow	{DOWN}
End	{END}
Enter	{ENTER}
Esc	{ESCAPE} or {ESC}
Help	{HELP}
Home	{HOME}
Insert	{INSERT}
Left Arrow	{LEFT}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Right Arrow	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Up Arrow	{UP}

To send one of function keys F1-F15, simply enclose the name of the key inside braces. For example, to send F5 use “{F5}”.

Note that special keywords can be used in combination with +, %, and ^. For example: % {TAB} means Alt-Tab. Also, you can send several special keys in the same way as you would send several normal keys: {UP 25} sends 25 Up arrows.

SendKeys can send keystrokes only to the currently active application. Therefore, you have to use the **AppActivate** statement to activate an application before sending keys (unless it is already active).

SendKeys cannot be used to send keys to an application which was not designed to run under Windows.

Example This example starts the Windows Terminal application and dials a phone number entered by the user.

```
Sub main
  Dim phonenumber, msgtext
  Dim x
  phonenumber=InputBox("Type telephone number to call:")
  x=Shell("Terminal.exe",1)
  SendKeys "%PD" & phonenumber & "{Enter}",1
  msgtext="Dialing..."
  MsgBox msgtext
End Sub
```

See Also [AppActivate](#), [DoEvents](#), [Shell](#)

Set Statement

Action Assigns a variable to an OLE2 object.

Syntax **Set** *variableName* = *expression* where *variableName* is an object variable or a Variant variable and *expression* is a function, an object member, or **Nothing**.

Comments The following example shows the syntax for the **Set** statement:

```
Dim OLE2 As Object
Set OLE2 = CreateObject("spoly.cpoly")
OLE2.reset
```

+ *If you omit the keyword **Set** when assigning an object variable, Basic will try to copy the default member of one object to the default member of another. This usually results in a runtime error:*

```
' Incorrect code - tries to copy default member!
OLE2 = GetObject(,"spoly.cpoly")
```

Example This example displays a list of open files in the software application, VISIO. It uses the Set statement to assign VISIO and its document files to object variables. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
  Dim visio as Object
  Dim doc as Object
  Dim msgtext as String
  Dim i as Integer, doccount as Integer
```



```

'Initialize Visio
Set visio = GetObject("visio.application") ' find Visio
If (visio Is Nothing) then
    MsgBox "Couldn't find Visio!"
Exit Sub
End If
'Get # of open Visio files
doccount = visio.documents.count 'OLE2 call to Visio
If doccount=0 then
    msgtext="No open Visio documents."
Else
    msgtext="The open files are: " & Chr$(13)
    For i = 1 to doccount
        Set doc = visio.documents(i) ' access Visio's document method
        msgtext=msgtext & Chr$(13)& doc.name
    Next i
End If
MsgBox msgtext
End Sub

```

See Also [CreateObject](#), [Is](#), [Me](#), [New](#), [Nothing](#), [Object Class](#), [Typeof](#)

SetAttr Statement

Action Sets the attributes for a file.

Syntax **SetAttr** *pathname\$*, *attributes%* where *pathname\$* is a string expression containing the filename to modify and *attributes %* is an integer containing the new attributes for the file.

Comments Wildcards are not allowed in *pathname\$*. If the file is open, you can modify its attributes, but only if it is opened for **Read** access. Here is a description of attributes that can be modified:

Value	Meaning
0	Normal file
1	Read-only file
2	Hidden file
4	System file
32	Archive - file has changed since last backup

Example This example tests the attributes for a file and if it is hidden, changes it to a normal (not hidden) file.

```

Sub main
    Dim filename as String
    Dim attribs, saveattribs as Integer
    Dim answer as Integer

```

```

Dim archno as Integer
Dim msgtext as String
archno=32
On Error Resume Next
msgtext="Enter name of a file:"
filename=InputBox(msgtext)
attribs=GetAttr(filename)
If Err<>0 then
    MsgBox "Error in filename. Re-run Program."
    Exit Sub
End If
saveattribs=attribs
If attribs>= archno then
    attribs=attribs-archno
End If
Select Case attribs
Case 2,3,6,7
    msgtext=" File: " & filename & " is hidden." & Chr(10)
    msgtext=msgtext & Chr(10) & " Change it?"
    answer=Msgbox(msgtext,308)
    If answer=6 then
        SetAttr filename, saveattribs-2
        MsgBox "File is no longer hidden."
        Exit Sub
    End If
    MsgBox "Hidden file not changed."
Case Else
    MsgBox "File was not hidden."
End Select
End Sub

```

See Also [FileAttr, GetAttr](#)

SetField Function [SBL Extension]**

Action	Replaces a field within a string and returns the modified string.
Syntax	SetField [\$](<i>string</i> \$, <i>field_number</i> %, <i>field</i> \$, <i>separator_chars</i> \$)
where	is
<i>string</i> \$	A string consisting of a series of fields, separated by <i>separator_char</i> \$.
<i>field_number</i> %	An integer for the field to replace within <i>string</i> \$.
<i>field</i> \$	An expression containing the new value for the field.
<i>separator_char</i> \$	A string containing the character(s) used to separate the fields in <i>string</i> \$.
Comments	<i>separator_char</i> \$ may contain multiple separator characters, although the first one will be used as the separator character.

The *field_number%* starts with 1. If *field_number%* is greater than the number of fields in the string, the returned string will be extended with separator characters to produce a string with the proper number of fields.

It is legal for the new *field\$* value to be a different size than the old value.

**SBL offers a number of extensions that are not included in Visual Basic.

Example

This example extracts the last name from a full name entered by the user.

```
Sub main
  Dim username as String
  Dim position as Integer
  username=InputBox("Enter your full name:")
  Do
    position=InStr(username," ")
    If position=0 then
      Exit Do
    End If
    username=SetField(username,1," "," ")
    username=Ltrim(username)
  Loop
  MsgBox "Your last name is: " & username
End Sub
```

See Also [GetField](#)

Sgn Function

Action	Returns a value indicating the sign of a number.
Syntax	Sgn (<i>number</i>) where <i>number</i> is an expression for the number to use.
Comments	The value that the Sgn function returns depends on the sign of <i>number</i> . For <i>numbers</i> > 0, Sgn (<i>number</i>) returns 1. For <i>numbers</i> = 0, Sgn (<i>number</i>) returns 0. For <i>numbers</i> < 0, Sgn (<i>number</i>) returns -1.

Example

This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative or zero.

```
Sub main
  Dim profit as Single
  Dim expenses
  Dim sales
  expenses=InputBox("Enter total expenses: ")
  sales=InputBox("Enter total sales: ")
  profit=Val(sales)-Val(expenses)
  If Sgn(profit)=1 then
```

```

        MsgBox "Yeah! We turned a profit!"
    ElseIf Sgn(profit)=0 then
        MsgBox "Okay. We broke even."
    Else
        MsgBox "Uh, oh. We lost money."
    End If
End Sub

```

See Also [Exp](#), [FixInt](#), [Int](#), [Log](#), [Rnd](#), [Sqr](#)

Shell Function

Action Starts a Windows application and returns its task ID.

Syntax **Shell**(*pathname\$*, [*windowstyle%*]) where *pathname\$* is the name of the program to execute and *windowstyle%* is an integer value for the style of the program's window (1-7).

Comments **Shell** runs an executable program. *Pathname\$* may be the name of any valid .COM, .EXE., .BAT, or .PIF file. Arguments or command line switches may be included. If *pathname\$* is not a valid executable file name, or if **Shell** cannot start the program, an error message occurs.

Windowstyle% is one of the following values:

Value	Window Style
1	Normal window with focus
2	Minimized with focus
3	Maximized with focus
4	Normal window without focus
7	Minimized without focus

If *windowstyle%* is not specified, the default of *windowstyle%* = 1 is assumed (normal window with focus).

Shell returns the task ID for the program, a unique number that identifies the running program.

Example This example activates the Terminal application and dials a number entered by the user.

```

Sub main
    Dim phonenumber, msgtext
    Dim x
    phonenumber=InputBox("Type telephone number to call:")
    x=Shell("Terminal.exe",1)

    SendKeys "%PD" & phonenumber & "{Enter}",1
    msgtext="Dialing..."
    MsgBox msgtext
End Sub

```

See Also [AppActivate](#), [Command](#), [SendKeys](#)

Sin Function

Action	Returns the sine of an angle specified in radians.
Syntax	Sin (<i>number</i>) where <i>number</i> is an expression containing the angle in radians.
Comments	The return value will be between -1 and 1. The return value is single-precision if the angle is an integer, currency or single-precision value, double precision for a long, Variant or double-precision value. The angle is specified in radians, and can be either positive or negative.

To convert degrees to radians, multiply by (PI/180). The value of PI is 3.14159.

Example	This example finds the height of the building, given the length of a roof and the roof pitch.
----------------	---

```
Sub main
  Dim height, rooflength
  Dim pitch
  Dim msgtext
  Const PI=3.14159
  Const conversion= PI/180
  pitch=InputBox("Enter the roof pitch in degrees:")
  pitch=pitch*conversion
  rooflength=InputBox("Enter the length of the roof in feet:")
  height=Sin(pitch)*rooflength
  msgtext="The height of the building is "
  msgtext=msgtext & Format(height, "##.##") & " feet."
  MsgBox msgtext
End Sub
```

See Also [Atn, Cos, Tan, Derived Trigonometric Functions](#)

Space Function

Action	Returns a string of spaces.
Syntax	Space [\$](<i>number</i>) where <i>number</i> is a numeric expression for the number of spaces to return.
Comments	<i>number</i> can be any numeric data type, but will be rounded to an integer. <i>number</i> must be between 0 and 32,767. The dollar sign, "\$", in the function name is optional. If specified the return type is String. If omitted, the function will return a Variant of vartype 8 (String).

Example This example prints the octal numbers from 1 to 15 as a two-column list and uses Space to separate the columns.

```
Sub main
  Dim x,y
  Dim msgtext
  Dim nospaces
  msgtext="Octal numbers from 1 to 15:" & Chr(10)
  For x=1 to 15
    nospaces=10
    y=Oct(x)
    If Len(x)=2 then
      nospaces=nospaces-2
    End If
    msgtext=msgtext & Chr(10) & x & Space(nospaces) & y
  Next x
  MsgBox msgtext
End Sub
```

See Also [Spc, String](#)

Spc Function

Action Prints a number of spaces.

Syntax **Spc** (*n*) where *n* is an integer for the number of spaces to output.

Comments The **Spc** function can be used only inside **Print** statement.

When the **Print** statement is used, the **Spc** function will use the following rules for determining the number of spaces to output:

- 1 If *n* is less than the total line width, **Spc** outputs *n* spaces.
- 2 If *n* is greater than the total line width, **Spc** outputs *n* Modwidth spaces.
- 3 If the difference between the current print position and the output line width (call this difference *x*) is less than *n* or *n* Modwidth, then **Spc** skips to the next line and outputs *n* - *x* spaces.

To set the width of a print line, use the **Width** statement.

Example This example puts five spaces and the string "ABCD" to a file. The five spaces are derived by taking 15 MOD 10, or the remainder of dividing 15 by 10.

```
Sub main
  Dim str1 as String
  Dim x as String*10
  str1="ABCD"
  Open "C:\TEMP001" For Output As #1
  Width #1, 10
  Print #1, Spc(15); str1
  Close #1
  Open "C:\TEMP001" as #1 Len=12
  Get #1, 1,x
```

```
Msgbox "The contents of the file is: " & x
Close #1
Kill "C:\TEMP001"
End Sub
```

See Also [Print, Space, Tab, Width](#)

Sqr Function

Action	Returns the square root of a number.
Syntax	Sqr (<i>number</i>) where <i>number</i> is an expression containing the number to use.
Comments	The return value is single-precision for an integer, currency or single-precision numeric expression, double precision for a long, Variant or double-precision numeric expression.
Example	This example calculates the square root of 2 as a double-precision floating point value and displays it in scientific notation.

```
Sub main
    Dim value as Double
    Dim msgtext
    value=Cdbl(Sqr(2))
    msgtext= "The square root of 2 is: " & Format(Value,"Scientific")
    MsgBox msgtext
End Sub
```

See Also [Exp, FixInt, Int, Log, Rnd, Sgn](#)

Static Statement

Action	Declares variables and allocate storage space.
Syntax	Static <i>variableName</i> [As <i>type</i>] [, <i>variableName</i> [As <i>type</i>]] ... where <i>variableName</i> is the name of the variable to declare and <i>type</i> is the data type of the variable.
Comments	Variables declared with the Static statement retain their value as long as the program is running. The syntax of Static is exactly the same as the syntax of the Dim statement. All variables of a procedure can be made static by using the Static keyword in a definition of that procedure See Function or Sub for more information.
Example	This example puts account numbers to a file using the record variable GRECORD and then prints them again.

```
Type acctrecord
    acctno as Integer
End Type
```

```

Sub main
  Static grecord as acctrecord
  Dim x
  Dim total
  x=1
  grecord.acctno=1
  On Error Resume Next
  Open "C:\TEMP001" For Output as #1
  Do While grecord.acctno<>0
i:   grecord.acctno=InputBox("Enter 0 or new account #" & x & ":")
    If Err<>0 then
      MsgBox "Error occurred. Try again."
      Err=0
      Goto i
    End If
    If grecord.acctno<>0 then
      Print #1, grecord.acctno
      x=x+1
    End If
  Loop
  Close #1
  total=x-1
  msgtext="The account numbers are: " & Chr(10)
  Open "C:\TEMP001" For Input as #1
  For x=1 to total
    Input #1, grecord.acctno
    msgtext=msgtext & Chr(10) & grecord.acctno
  Next x
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub

```

See Also [Dim, Function...End Function, Global, Option Base, ReDim, Sub...End Sub](#)

StaticComboBox Statement

Action	Creates a combination of a list of choices and a text box.
Syntax A	StaticComboBox <i>x, y, dx, dy, text\$, .field</i>
Syntax B	StaticComboBox <i>x, y, dx, dy, stringarray\$(), .field</i>
where	is
<i>x,y</i>	the upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the combo box in which the user enters or selects text.
<i>text\$</i>	a string containing the selections for the combo box.
<i>stringarray\$</i>	an array of dynamic strings for the selections in the combo box.
<i>.field</i>	the name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The **StaticComboBox** statement is equivalent to the **ComboBox** or **DropComboBox** statement, but the list box of **StaticComboBox** always stays visible. All dialog functions and statements that apply to the **ComboBox** apply to the **StaticComboBox** as well.

The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text\$* argument must be defined, using a **Dim Statement**, before the **Begin Dialog** statement is executed. The arguments in the *text\$* string are entered as shown in the following example:

```
dimname = "listchoice"+Chr$(9)+"listchoice"+Chr$(9)+"listchoice"...
```

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

Use the **StaticComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a static combo box labeled “Installed Drivers” and the OK and Cancel buttons.

```
Sub main
    Dim cchoices as String
    cchoices="MIDI Mapper"+Chr$(9)+"Timer"
    Begin Dialog UserDialog 182, 116, "SBL Dialog Box"
        StaticComboBox 7, 20, 87, 49, cchoices, .StaticComboBox1
        Text 6, 3, 83, 10, "Installed Drivers", .Text1
        OkButton 118, 12, 54, 14
        CancelButton 118, 34, 54, 14
    End Dialog
    Dim mydialogbox As UserDialog
    Dialog mydialogbox
    If Err=102 then
        MsgBox "You pressed Cancel."
    Else
        MsgBox "You pressed OK."
    End If
End Sub
```

See Also [Begin Dialog...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#), [TextBox](#)

Stop Statement

Action	Halts program execution.
Syntax	Stop
Comments	Stop statements can be placed anywhere in a program to suspend its execution. Although the Stop statement halts program execution, it does not close files or clear variables.
Example	This example stops program execution at the user's request.

```
Sub main
  Dim str1
  str1=InputBox("Stop program execution? (Y/N):")
  If str1="Y" or str1="y" then
    Stop
  End If
  MsgBox "Program complete."
End Sub
```

Str Function

Action	Returns a string representation of a number.
Syntax	Str[\$](<i>number</i>) where <i>number</i> is the number to represent as a string.
Comments	<p>The precision in the returned string is single-precision for an integer or single-precision numeric expression, double precision for a long or double-precision numeric expression, and currency precision for currency. Variants return the precision of their underlying vartype.</p> <p>The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted, the function will return a Variant of vartype 8 (String).</p>
Example	This example prompts for two numbers, adds them, then shows them as a concatenated string.

```
Sub main
  Dim x as Integer
  Dim y as Integer
  Dim str1 as String
  Dim value1 as Integer
  x=InputBox("Enter a value for x: ")
  y=InputBox("Enter a value for y: ")
  MsgBox "The sum of these numbers is: " & x+y
  str1=Str(x) & Str(y)
  MsgBox "The concatenated string for these numbers is: " & str1
End Sub
```

See Also [Format](#), [Val](#)

StrComp Function

Action Compares two strings and returns an integer specifying the result of the comparison.

Syntax **StrComp**(*string1*\$, *string2*\$ [, *compare*%])

where **is**
string1\$ any expression containing the first string to compare.
string2\$ the second string to compare.
compare% an integer for the method of comparison (0=case-sensitive, 1=case-insensitive).

Comments **StrComp** returns one of the following values:

Value	Meaning
-1	<i>string1</i> \$ < <i>string2</i> \$
0	<i>string1</i> \$ = <i>string2</i> \$
>1	<i>string1</i> \$ > <i>string2</i> \$
Null	<i>string1</i> \$ = Null or <i>string2</i> \$ = Null

If *compare*% is 0, a case sensitive comparison based on the ANSI character set sequence is performed. If *compare*% is 1, a case insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system. If omitted, the module level default, as specified with **Option Compare** is used.

The *string1* and *string2* arguments are both passed as Variants. Therefore, any type of expression is supported. Numbers will be automatically converted to strings.

Example This example compares a user-entered string to the string "Smith".

```
Option Compare Text
Sub main
    Dim lastname as String
    Dim smith as String
    Dim x as Integer
    smith="Smith"
    lastname=InputBox("Type your last name")
    x=StrComp(lastname,smith,1)
    If x=0 then
        MsgBox "You typed 'Smith' or 'smith'."
    Else
        MsgBox "You typed: " & lastname & " not 'Smith'."
    End If
End Sub
```

See Also [Instr](#), [Option Compare](#)

String FunctionString

Action Returns a string consisting of a repeated character.

Syntax A **String**[\$](*number*, *Character*%)

Syntax B **String**[\$] (*number*, *string-expression*\$)

where	is
<i>number</i>	the length of the string to be returned.
<i>Character</i> %	a numeric expression that contains an integer for the decimal ANSI code of the character to use.
<i>string-expression</i> \$	a string argument, the first character of which becomes the repeated character.

Comments *number* must be between 0 and 32,767.

Character% must evaluate to an integer between 0 and 255.

The dollar sign, "\$", in the function name is optional. If specified the return type is string. If omitted, the function returns a **Variant** of vartype 8 (String).

Example This example places asterisks (*) in front of a string that is printed as a payment amount.

```
Sub main
    Dim str1 as String
    Dim size as Integer
i: str1=InputBox("Enter an amount up to 999,999.99: ")
    If Instr(str1,".")=0 then
        str1=str1+ ".00"
    End If
    If Len(str1)>10 then
        MsgBox "Amount too large. Try again."
        Goto i
    End If
    size=10-Len(str1)
    'Print amount in a space on a check allotted for 10 characters
    str1=String(size,Asc("")) & str1
    MsgBox "The amount is: $" & str1
End Sub
```

See Also [Space](#), [Str](#)

Sub ... End Sub Statement

Action	Defines a subprogram procedure.
Syntax	[Static] [Private] Sub <i>name</i> [([Optional] <i>parameter</i> [As <i>type</i>] , ...)] End Sub

where	is
<i>name</i>	the name of the subprogram.
<i>parameter</i>	a comma-separated list of parameter names.
<i>type</i>	a data type for <i>parameter</i>

Comments	A call to a subprogram stands alone as a separate statement. (See the Call statement). Recursion is supported.
-----------------	---

The data type of a parameter may be specified by using a type character or by using the **As** clause. Record parameters are declared by using an **As** clause and a *type* which has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the **Sub** statement. All references to an array within the body of the subprogram must have a consistent number of dimensions.

If a *parameter* is declared as **Optional**, its value may be omitted when the function is called. Only Variant parameters may be declared as optional, and all optional parameters must appear after all required parameters in the **Sub** statement.

The procedure returns to the caller when the **End Sub** statement is reached or when an **Exit Sub** statement is executed.

The **Static** keyword specifies that all the variables declared within the subprogram will retain their values as long as the program is running, regardless of the way the variables are declared.

The **Private** keyword specifies that the procedures will not be accessible to functions and subprograms from other modules. Only procedures defined in the same module will have access to a **Private** subprogram.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it will modify the variable passed by the caller.

The **MAIN** subprogram has a special meaning. In many implementations of Basic, **MAIN** will be called when the module is “run”. The **MAIN** subprogram is not allowed to take arguments.

Use **Function** to define a procedure which has a return value.

Example This example is a subroutine that uses the Sub...End Sub function.

```
Sub main
    MsgBox "Hello, World."
End Sub
```

See Also [Call, Dim, Function...End Function, Global, Option Explicit, Static](#)

Tab Function

Action Moves the current print position to the column specified.

Syntax **Tab** (*n*) where *n* is the new print position to use.

Comments The **Tab** function can be used only inside **Print** statement. The leftmost print position is position number 1.

When the **Print** statement is used, the **Tab** function will use the following rules for determining the next print position:

- 1 If *n* is less than the total line width, the new print position is *n*.
- 2 If *n* is greater than the total line width, the new print position is *n Mod width* .
- 3 If the current print position is greater than *n* or *n Mod width*, Tab skips to the next line and sets the print position to *n* or *n Mod width*.

To set the width of a print line, use the **Width** statement.

Example This example prints the octal values for the numbers from 1 to 25. It uses Tab to put five character spaces between the values.

```
Sub main
    Dim x as Integer
    Dim y
    For x=1 to 25
        y=Oct$(x)
        Print x Tab(10) y
    Next x
End Sub
```

See Also [Print, Space, Spc, Width](#)

Tan Function

Action Returns the tangent of an angle in radians.

Syntax **Tan**(*number*) where *number* is an expression containing the angle in radians.

Comments *number* is specified in radians, and can be either positive or negative.

The return value is single-precision if the angle is an integer, currency or single-precision value, double precision for a long, Variant or double-precision value.

To convert degrees to radians, multiply by PI/180. The value of PI is 3.14159.

Example This example finds the height of the exterior wall of a building, given its roof pitch and the length of the building.

```
Sub main
  Dim bldglen, wallht
  Dim pitch
  Dim msgtext
  Const PI=3.14159
  Const conversion= PI/180
  On Error Resume Next
  pitch=InputBox("Enter the roof pitch in degrees:")
  pitch=pitch*conversion
  bldglen=InputBox("Enter the length of the building in feet:")
  wallht=Tan(pitch)*(bldglen/2)
  msgtext="The height of the building is: " & Format(wallht, "##.00")
  MsgBox msgtext
End Sub
```

See Also [Atn, Cos, Sin, Derived Trigonometric Functions](#)

Text Statement

Action Places line(s) of text in a dialog box.

Syntax **Text** *x, y, dx, dy, text\$* [, *.id*]

where **is**

x,y the upper left corner coordinates of the text area, relative to the upper left corner of the dialog box.

dx,dy the width and height of the text area.

text\$ a string containing the text to appear in the text area defined by *x,y*.

.id an optional identifier used by the dialog statements that act on this control.

Comments If the width of *text\$* is greater than *dx*, the spillover characters wrap to the next line. This will continue as long as the height of the text area established by *dy* is not exceeded. Excess characters are truncated.

By preceding an underlined character in *text\$* with an ampersand (&), you enable a user to press the underlined character on the keyboard and position the cursor in the combo or text box defined in the statement immediately following the **Text** statement.

Use the **Text** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example defines a dialog box with a combination list and text box and three buttons.

```
Sub main
  Dim ComboBox1() as String
  ReDim ComboBox1(0)
  ComboBox1(0)=Dir("C:\*.*)
  Begin Dialog UserDialog 166, 142, "SBL Dialog Box"
    Text 9, 3, 69, 13, "Filename:", .Text1
    DropComboBox 9, 14, 81, 119, ComboBox1(), .ComboBox1
    OkButton 101, 6, 54, 14
    CancelButton 101, 26, 54, 14
    PushButton 101, 52, 54, 14, "Help", .Push1
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

See Also [Begin Dialog...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [TextBox](#)

TextBox Statement

Action Creates a text box in a dialog box.

Syntax **TextBox** [**NoEcho**] *x*, *y*, *dx*, *dy*, *.field*

where	is
<i>x,y</i>	the upper left corner coordinates of the text box, relative to the upper left corner of the dialog box.
<i>dx,dy</i>	the width and height of the text box area.
<i>.field</i>	the name of the dialog record field to hold the text string.

Comments A *dy* value of 12 will usually accommodate text in the system font.

When the user selects the OK button, or any pushbutton other than cancel, the text string entered in the text box will be recorded in *.field*.

The **NoEcho** keyword is often used for passwords; it displays all characters entered as asterisks (*).

Use the **TextBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Example This example creates a dialog box with a group box, and two buttons.

```
Sub main
Begin Dialog UserDialog 194, 76, "SBL Dialog Box"
  GroupBox 9, 8, 97, 57, "File Range"
  OptionGroup .OptionGroup2
    OptionButton 19, 16, 46, 12, "All pages", .OptionButton3
    OptionButton 19, 32, 67, 8, "Range of pages", .OptionButton4
  Text 25, 43, 20, 10, "From:", .Text6
  Text 63, 43, 14, 9, "To:", .Text7
  TextBox 79, 43, 13, 12, .TextBox4
  TextBox 47, 43, 12, 11, .TextBox5
  OkButton 135, 6, 54, 14
  CancelButton 135, 26, 54, 14
End Dialog
Dim mydialog as UserDialog
On Error Resume Next
Dialog mydialog
If Err=102 then
  MsgBox "Dialog box canceled."
End If
End Sub
```

See Also [Begin Dialog...End Dialog](#), [Button](#), [ButtonGroup](#), [CancelButton](#), [Caption](#), [CheckBox](#), [ComboBox](#), [Dialog](#), [DropComboBox](#), [GroupBox](#), [ListBox](#), [OKButton](#), [OptionButton](#), [OptionGroup](#), [Picture](#), [StaticComboBox](#), [Text](#)

Time Function

Action	Returns a string representing the current time.
Syntax	Time[\$]
Comments	<p>The Time function returns an eight character string. The format of the string is “<i>hh:mm:ss</i>” where <i>hh</i> is the hour, <i>mm</i> is the minutes and <i>ss</i> is the seconds. The hour is specified in military style, and ranges from 0 to 23.</p> <p>The dollar sign, “\$”, in the function name is optional. If specified, the return type is String. If omitted, the function will return a Variant of vartype 8 (String).</p>
Example	This example writes data to a file if it hasn't been saved within the last 2 minutes.

```
Sub main
Dim tempfile
Dim filetype, curtime
Dim msgtext
Dim acctno(100) as Single
Dim x, l
tempfile="C:\TEMP001"
Open tempfile For Output As #1
filetime=FileDateTime(tempfile)
x=1
l=1
acctno(x)=0
```

```

Do
  curtime=Time
  acctno(x)=InputBox("Enter an account number (99 to end):")
  If acctno(x)=99 then
    For l=1 to x-1
      Write #1, acctno(l)
    Next l
  Exit Do
  ElseIf (Minute(filetime)+2)<=Minute(curtime) then
    For l=1 to x
      Write #1, acctno(l)
    Next l
  End If
  x=x+1
Loop
Close #1
x=1
msgtext="Contents of C:\TEMP001 is:" & Chr(10)
Open tempfile for Input as #1
Do While Eof(1)<>-1
  Input #1, acctno(x)
  msgtext=msgtext & Chr(10) & acctno(x)
  x=x+1
Loop
MsgBox msgtext
Close #1
Kill "C:\TEMP001"
End Sub

```

See Also [Date Function](#), [Date Statement](#), [Time Statement](#), [Timer](#), [TimeSerial](#), [TimeValue](#)

Time Statement

Action	Sets the system time.						
Syntax	Time [\$] = <i>expression</i> where <i>expression</i> is an expression that evaluates to a valid time.						
Comments	When Time (with the dollar sign "\$") is used, the <i>expression</i> must evaluate to a string of one of the following forms: <table data-bbox="519 1438 1250 1535"> <tr> <td><i>hh</i></td><td>Set the time to <i>hh</i> hours 0 minutes and 0 seconds</td></tr> <tr> <td><i>hh:mm</i></td><td>Set the time to <i>hh</i> hours <i>mm</i> minutes and 0 seconds.</td></tr> <tr> <td><i>hh:mm:ss</i></td><td>Set the time to <i>hh</i> hours <i>mm</i> minutes and <i>ss</i> seconds</td></tr> </table>	<i>hh</i>	Set the time to <i>hh</i> hours 0 minutes and 0 seconds	<i>hh:mm</i>	Set the time to <i>hh</i> hours <i>mm</i> minutes and 0 seconds.	<i>hh:mm:ss</i>	Set the time to <i>hh</i> hours <i>mm</i> minutes and <i>ss</i> seconds
<i>hh</i>	Set the time to <i>hh</i> hours 0 minutes and 0 seconds						
<i>hh:mm</i>	Set the time to <i>hh</i> hours <i>mm</i> minutes and 0 seconds.						
<i>hh:mm:ss</i>	Set the time to <i>hh</i> hours <i>mm</i> minutes and <i>ss</i> seconds						

Time uses a 24-hour clock. Thus, 6:00 P.M. must be entered as 18:00:00.

If the dollar sign '\$' is omitted, *expression* can be a string containing a valid date, a **Variant** of vartype 7 (date) or 8 (string).

If *expression* is not already a Variant of vartype 7 (date), **Time** attempts to convert it to a valid time. It recognizes time separator characters defined in the International section of the Windows Control Panel. **Time** (without the \$) accepts both 12 and 24 hour clocks.

Example This example changes the time on the system clock.

```
Sub main
    Dim newtime as String
    Dim answer as String
    On Error Resume Next
i: newtime=InputBox("What time is it?")
    answer=InputBox("Is this AM or PM?")
    If answer="PM" or answer="pm" then
        newtime=newtime & "PM"
    End If
    Time=newtime
    If Err<>0 then
        MsgBox "Invalid time. Try again."
        Err=0
        Goto i
    End If
End Sub
```

See Also [Date Function](#), [Date Statement](#), [Time Function](#), [TimeSerial](#), [TimeValue](#)

Timer Function

Action	Returns the number of seconds that have elapsed since midnight.
Syntax	Timer
Comments	The Timer function can be used in conjunction with the Randomize statement to seed the random number generator.
Example	This example uses Timer Function to find a Megabucks number.

```
Sub main
    Dim msgtext
    Dim value(9)
    Dim nextvalue
    Dim x
    Dim y
    msgtext="Your Megabucks numbers are: "
    For x=1 to 8
        Do
            value(x)=Timer
            value(x)=value(x)*100
            value(x)=Str(value(x))
            value(x)=Val(Right(value(x),2))
        Loop Until value(x)>1 and value(x)<36
        For y=1 to 1500
            Next y
        Next x
    Next x
```

```

For y=1 to 8
  For x= 1 to 8
    If y<>x then
      If value(y)=value(x) then
        value(x)=value(x)+1
      End If
    End If
  Next x
Next y
For x=1 to 8
  msgtext=msgtext & value(x) & " "
Next x
MsgBox msgtext
End Sub

```

See Also [Randomize](#)

TimeSerial Function

Action	Returns a time as a Variant of type 7 (date/time) for a specific hour, minute, and second.
Syntax	TimeSerial (<i>hour%</i> , <i>minute%</i> , <i>second%</i>) where <i>hour%</i> is a numeric expression for an hour (0-23), <i>minute%</i> is a numeric expression for a minute (0-59) and <i>second%</i> is a numeric expression for a second (0-59).
Comments	You also can specify relative times for each argument by using a numeric expression representing the number of hours, minutes, or seconds before or after a certain time.

Example This example displays the current time using Time Serial.

```

Sub main
  Dim y
  Dim msgtext
  Dim nowhr
  Dim nowmin
  Dim nowsec
  nowhr=Hour(Now)
  nowmin=Minute(Now)
  nowsec=Second(Now)
  y=TimeSerial(nowhr,nowmin,nowsec)
  msgtext="The time is: " & y
  MsgBox msgtext
End Sub

```

See Also [DateSerial](#), [Date Value](#), [Hour](#), [Minute](#), [Now](#), [Second](#), [TimeValue](#)

TimeValue Function

Action	Returns a time value for a specified string.
Syntax	TimeValue (<i>time\$</i>) where <i>time\$</i> is a string representing a valid date time value.
Comments	The TimeValue function returns a Variant of vartype 7 (date/time) that represents a time between 0:00:00 and 23:59:59, or 12:00:00 A.M. and 11:59:59 P.M., inclusive.
Example	This example writes a variable to a disk file based on a comparison of its last saved time and the current time. Note that all the variables used for the TimeValue function are dimensioned as Double, so that calculations based on their values will work properly.

```

Sub main
    Dim tempfile
    Dim ftime
    Dim filetime as Double
    Dim curtime as Double
    Dim minutes as Double
    Dim acctno(100) as Integer
    Dim x, l
    tempfile="C:\TEMP001"
    Open tempfile For Output As 1
    ftime=FileDateTime(tempfile)
    filetime=TimeValue(ftime)
    minutes= TimeValue("00:02:00")
    x=1
    l=1
    acctno(x)=0
    Do
        curtime= TimeValue(Time)
        acctno(x)=InputBox("Enter an account number (99 to end):")
        If acctno(x)=99 then
            For l=l to x-1
                Write #1, acctno(l)
            Next l
            Exit Do
        ElseIf filetime+minutes<=curtime then
            For l=l to x
                Write #1, acctno(l)
            Next l
        End If
        x=x+1
    Loop
    Close #1
    x=1
    msgtext="You entered:" & Chr(10)
    Open tempfile for Input as #1
    Do While Eof(1)<>-1
        Input #1, acctno(x)
        msgtext=msgtext & Chr(10) & acctno(x)
        x=x+1
    Loop
    MsgBox msgtext

```

```

Close #1
Kill "C:\TEMP001"
End Sub

```

See Also [DateSerial](#), [Date Value](#), [Hour](#), [Minute](#), [Now](#), [Second](#), [TimeSerial](#)

Trim Function

Action	Returns a copy of a string after removing all leading and trailing spaces.
Syntax	Trim [\$](<i>string</i>) where <i>string</i> is an expression containing the string to trim.
Comments	<p>Trim accepts expressions of type String. Trim accepts any type of <i>string</i> including numeric values and will convert the input value to a string.</p> <p>The dollar sign, "\$", in the function name is optional. If specified, the return type is String. If omitted, the function typically returns a Variant of vartype 8 (String). If the value of <i>string</i> is NULL, a Variant of vartype 1 (Null) is returned.</p>
Example	<p>This example removes leading and trailing spaces from a string entered by the user.</p>

```

Sub main
    Dim userstr as String
    userstr=InputBox("Enter a string with leading/trailing spaces")
    MsgBox "The string is: " & Trim(userstr) & " with nothing after it."
End Sub

```

See Also [GetField](#), [Left](#), [Len](#), [Ltrim](#), [Mid Function](#), [Mid Statement](#), [Right](#), [RTrim](#)

Type Statement

Action	Declares a user-defined type.
Syntax	<pre> Type userType field1 As type1 field2 As type2 ... End Type where <i>userType</i> is <i>field1</i>, <i>field2</i> the name of a field in the user-defined type. <i>type1</i>, <i>type2</i> a data type: Integer, Long, Single, Double, Currency, String, String*<i>length</i>, Variant, or another user-defined type. </pre>

Comments The user-defined type declared by **Type** can then be used in the **Dim** statement to declare a record variable. A user-defined type is sometimes referred to as a *record type* or a *structure type*.

field may not be an array. However, arrays of records are allowed.

The **Type** statement is not valid inside of a procedure definition. To access the fields of a record, use notation of the form:

recordName.fieldName

To access the fields of an array of records, use notation of the form:

arrayName(index).fieldName

Example This example shows a Type and Dim statement for a record. You must define a record type before you can declare a record variable. The subroutine then references a field within the record.

```

Type Testrecord
  Custno As Integer
  Custname As String
End Type
Sub main
  Dim myrecord As Testrecord
  i: myrecord.custname=InputBox("Enter a customer name:")
  If myrecord.custname="" then
    Exit Sub
  End If
  answer=InputBox("Is the name: " & myrecord.custname & " correct? (Y/N)")

  If answer="Y" or answer="y" then
    MsgBox "Thank you."
  Else
    MsgBox "Try again."
    Goto i
  End If
End Sub

```

See Also [Deftype](#), [Dim](#)

Typeof Function

Action	Returns a value indicating whether an object is of a given class (-1=TRUE, 0=FALSE).
Syntax	If Typeof <i>objectVariable</i> Is <i>className</i> then . . . where <i>objectVariable</i> is the object to test and <i>className</i> is the class to compare the object to.
Comments	Typeof may only be used in an If statement and may not be combined with other boolean operators. That is, Typeof may only be used exactly as shown in the syntax above.

To test if an object does *not* belong to a class, use the following code structure:

```
If Typeof objectVariable Is className Then
Else
    Rem Perform some action.
End If
```

Example This example .

```
Sub main
---TBD---
End Sub
```

See Also [CreateObject](#), [GetObject](#), [Is](#), [Me](#), [New](#), [Nothing](#), [Object Class](#)

UBound Function

Action Returns the upper bound of the subscript range for the specified array.

Syntax **UBound**(*arrayname* [, *dimension*]) where *arrayname* is the name of the array to use and *dimension* is the dimension to use.

Comments The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is used as a default.

LBound can be used with **UBound** to determine the length of an array.

Example This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
        arrayvar(x)=InputBox("Enter test score #" & x & ":")
        x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
        count=InputBox("How many more do you have?")
        If count<>0 then
            count=count+(x-1)
            ReDim Preserve arrayvar(count)
            Goto start
```



```

End If
End If
x=LBound(arrayvar,1)
count=UBound(arrayvar,1)
For y=x to count
    total=total+arrayvar(y)
Next y
MsgBox "The average of the " & count & " scores is: " & Int(total/count)
End Sub

```

See Also [Dim](#), [Global](#), [Lbound](#), [Option Base](#), [ReDim](#), [Static](#)

UCase Function

Action	Returns a copy of a string after converting all lowercase letters to uppercase.
Syntax	UCase [\$](<i>string</i>) where <i>string</i> is an expression that evalutes to a string.
Comments	<p>The translation is based on the country specified in the Windows Control Panel.</p> <p>UCase accepts expressions of type string. UCase accepts any type of argument and will convert the input value to a string.</p> <p>The dollar sign, “\$”, in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a Variant of vartype 8 (String). If the value of <i>string</i> is Null, a Variant of vartype 1 (Null) is returned.</p>
Example	<p>This example converts a filename entered by a user to all uppercase letters.</p> <pre> Option Base 1 Sub main Dim filename as String filename=InputBox("Enter a filename: ") filename=UCase(filename) MsgBox "The filename in uppercase is: " & filename End Sub </pre>

See Also [Asc](#), [LCase](#)

Val Function

Action	Returns the numeric value of the first number found in the specified string.
Syntax	Val (<i>string</i> \$) where <i>string</i> \$ is a string expression containing a number.
Comments	Spaces in the source string are ignored. If no number is found, Val returns 0.

Example This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative or zero.

```
Sub main
    Dim profit as Single
    Dim expenses
    Dim sales
    expenses=InputBox("Enter total expenses: ")
    sales=InputBox("Enter total sales: ")
    profit=Val(sales)-Val(expenses)
    If Sgn(profit)=1 then
        MsgBox "Yeah! We turned a profit!"
    ElseIf Sgn(profit)=0 then
        MsgBox "Okay. We broke even."
    Else
        MsgBox "Uh, oh. We lost money."
    End If
End Sub
```

See Also [Ccur](#), [Cdbl](#), [Cint](#), [Clng](#), [Csng](#), [Cstr](#), [Cvar](#), [CVDate](#), [Format](#), [Str](#)

VarType Function

Action Returns the Variant type of the specified Variant variable (0-9).

Syntax **VarType**(*varname*) where *varname* is the **Variant** variable to use.

Comments The value returned by **VarType** is one of the following:

Ordinal	Representation
0	(Empty)
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	String
9	Object

Example This example returns the type of a variant.

```
Sub main
  Dim x
  Dim myarray(8)
  Dim retval
  Dim retstr
  myarray(1)=Null
  myarray(2)=0
  myarray(3)=39000
  myarray(4)=CSng(10^20)
  myarray(5)=10^300
  myarray(6)=CCur(10.25)
  myarray(7)=Now
  myarray(8)="Five"
  For x=0 to 8
    retval=Vartype(myarray(x))
    Select Case retval
      Case 0
        retstr=" (Empty)"
      Case 1
        retstr=" (Null)"
      Case 2
        retstr=" (Integer)"
      Case 3
        retstr=" (Long)"
      Case 4
        retstr=" (Single)"
      Case 5
        retstr=" (Double)"
      Case 6
        retstr=" (Currency)"
      Case 7
        retstr=" (Date)"
      Case 8
        retstr=" (String)"
    End Select

    If retval=1 then
      myarray(x)="[null]"
    ElseIf retval=0 then
      myarray(x)="[empty]"
    End If
    MsgBox "The variant type for " & myarray(x) & " is: " & retval & retstr
  Next x
End Sub
```

See Also [IsDate](#), [IsEmpty](#), [IsNull](#), [IsNumeric](#)

Weekday Function

Action	Returns the day of the week for the specified date-time value.
Syntax	Weekday (<i>date</i>) where <i>date</i> is an expression containing a date time value.
Comments	<p>The Weekday function returns an integer between 1 and 7, inclusive (1=Sunday, 7=Saturday).</p> <p>Weekday accepts any expression, including strings, and attempts to convert the input value to a date value.</p> <p>The return value is a Variant of vartype 2 (Integer). If the value of <i>date</i> is NULL, a Variant of vartype 1 (Null) is returned.</p>
Example	<p>This example finds the day of the week on which New Year's Day will fall in the year 2000.</p> <pre> Sub main Dim newyearsday Dim daynumber Dim msgtext Dim newday as Variant Const newyear=2000 Const newmonth=1 Let newday=1 newyearsday=DateSerial(newyear,newmonth,newday) daynumber=Weekday(newyearsday) msgtext="New Year's day 2000 falls on a " & Format(daynumber, "dddd") MsgBox msgtext End Sub </pre>

See Also [Date Function](#), [Date Statement](#), [Day](#), [Hour](#), [Minute](#), [Month](#), [Now](#), [Second](#), [Year](#)

While ... Wend

Action	Controls a repetitive action.
Syntax	<p>While <i>condition</i></p> <p style="padding-left: 100px;"><i>statementblock</i></p> <p>Wend</p> <p>where <i>condition</i> is an expression that evaluates to True (non-zero) or False (zero) and <i>statementblock</i> is a series of statements to execute if <i>condition</i> is True.</p>
Comments	<p>The <i>statementblock</i> statements are until <i>condition</i> becomes 0 (False).</p> <p>The While statement is included in SBL for compatibility with older versions of Basic. The Do statement is a more general and powerful flow control statement.</p>

Example The uses While...Wend to loop through the C:\TEMP00? files. These files are created by the subroutine CREATEFILES.

```

Declare Sub createfiles
Sub main
  Dim custfile as String
  Dim aline as String
  Dim pattern as String
  Dim count as Integer
  Call createfiles
  Chdir "C:\"
  custfile=Dir$("TEMP00?")
  pattern="***" + "Overdue" + "***"
  While custfile <> ""
    Open custfile for input as #1
    On Error goto atEOF
    Do
      Line Input #1, aline
      If aline Like pattern Then
        count=count+1
      End If
    Loop
  nxfile:
    On Error GoTo 0
    Close #1
    custfile = Dir$
  Wend
  If count<>0 then
    MsgBox "Number of overdue accounts: " & count
  Else
    MsgBox "No accounts overdue"
  End If
  Kill "C:\TEMP001"
  Kill "C:\TEMP002"
Exit Sub
atEOF:
  Resume nxfile
End Sub

Sub createfiles()
  Dim odue as String
  Dim ontime as String
  Dim x
  Open "C:\TEMP001" for OUTPUT as #1
  odue="***" + "Overdue" + "***"
  ontime="***" + "On-Time" + "***"
  For x=1 to 3
    Write #1, odue
  Next x
  For x=4 to 6
    Write #1, ontime
  Next x
  Close #1
  Open "C:\TEMP002" for Output as #1
  Write #1, odue
  Close #1
End Sub

```

See Also [Do...Loop](#)

Width Statement

Action	Sets the output line width for an open file.
Syntax	Width [#] <i>filename</i> %, <i>width</i> % where <i>filename</i> % is an integer expression for the open file to use and <i>width</i> % is an integer expression for the width of the line (0 to 255).
Comments	<p><i>Filename</i>% is the number assigned to the file when it is opened. See the Open statement for more information.</p> <p>A value of zero (0) for <i>width</i>% indicates there is no line length limit. The default <i>width</i>% for a file is zero (0).</p>
Example	<p>This example puts five spaces and the string "ABCD" to a file. The five spaces are derived by taking 15 MOD 10, or the remainder of dividing 15 by 10.</p> <pre> Sub main Dim str1 as String Dim x as String*10 str1="ABCD" Open "C:\TEMP001" For Output As #1 Width #1, 10 Print #1, Spc(15); str1 Close #1 Open "C:\TEMP001" as #1 Len=12 Get #1, 1,x MsgBox "The contents of the file is: " & x Close #1 Kill "C:\TEMP001" End Sub </pre>
See Also	Open , Print

With Statement [SBL Extension]**

Action	Executes a series of statements on a specified variable.
Syntax	<p>With <i>variable</i></p> <p style="padding-left: 40px;"><i>statement_block</i></p> <p>End With</p> <p>where <i>variable</i> is the variable to be changed by the statements in <i>statement_block</i> and <i>statement_block</i> the statements to execute.</p>
Comments	<p><i>Variable</i> may be an object or a user defined type. The With statements can be nested.</p> <p>**SBL offers a number of extensions that are not included in Visual Basic.</p>

Example This example creates a user-defined record type, `custrecord` and uses the `With` statement to fill in values for the record fields, for the record called “John”.

```
Type custrecord
  name as String
  ss as String
  salary as Single
  dob as Variant
  street as String
  apt as Variant
  city as String
  state as String
End Type
Sub main
  Dim John as custrecord
  Dim msgtext
  John.name="John"
  With John
    .ss="037-67-2947"
    .salary=60000
    .dob=#10-09-65#
    .street="15 Chester St."
    .apt=28
    .city="Cambridge"
    .state="MA"
  End With
  msgtext=Chr(10) & "Name:" & Space(5) & John.name & Chr(10)
  msgtext=msgtext & "SS#: " & Space(6) & John.ss & Chr(10)
  msgtext=msgtext & "D.O.B.:" & Space(4) & John.dob
  MsgBox "Done with: " & Chr(10) & msgtext
End Sub
```

See Also [Type...End Type](#)

Write Statement

Action	Writes data to an open sequential file.
Syntax	<p>Write [#] <i>filenumber%</i> [,<i>expressionlist</i>]</p> <p>where <i>filenumber%</i> is an integer expression for the open file to use and <i>expressionlist</i> is one or more values to write to the file.</p>
Comments	<p>The file must be opened in Output or Append mode. <i>Filenumber%</i> is the number assigned to the file when it is opened. See the Open statement for more information.</p> <p>If <i>expressionlist</i> is omitted, the Write statement writes a blank line to the file. (See Input for more information.)</p>
Example	This example writes a variable to a disk file based on a comparison of its last saved time and the current time.

```
Sub main
  Dim tempfile
  Dim filetype, curtime
  Dim msgtext
  Dim acctno(100) as Single
  Dim x, l
  tempfile="C:\TEMP001"
  Open tempfile For Output As #1
  filetype=FileDateTime(tempfile)
  x=1
  l=1
  acctno(x)=0
  Do
    curtime=Time
    acctno(x)=InputBox("Enter an account number (99 to end):")
    If acctno(x)=99 then
      If x=1 then Exit Sub
      For l=1 to x-1
        Write #1, acctno(l)
      Next l
      Exit Do
    ElseIf (Minute(filetime)+2)<=Minute(curtime) then
      For l=1 to x-1
        Write #1, acctno(l)
      Next l
    End If
    x=x+1
  Loop
  Close #1
  x=1
  msgtext="Contents of C:\TEMP001 is:" & Chr(10)
  Open tempfile for Input as #1
  Do While Eof(1)<>-1
    Input #1, acctno(x)
    msgtext=msgtext & Chr(10) & acctno(x)
    x=x+1
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub
```

See Also [Close, Open, Print, Put](#)

Year Function

Action Returns the year component (1-12) of a date-time value.

Syntax **Year**(*date*)

where *date* is an expression that can evaluate to a date time value.

- Comments** The **Year** function returns an integer between 100 and 9999, inclusive.
- Year** accepts any type of *date*, including strings, and will attempt to convert the input value to a date value.
- The return value is a **Variant** of vartype 2 (Integer). If the value of *date* is NULL, a Variant of vartype 1 (Null) is returned.
- Example** This example returns the year for today.
- ```
Sub main
 Dim nowyear
 nowyear=Year(Now)
 MsgBox "The current year is: " &nowyear
End Sub
```
- See Also**      [Date Function](#), [Date Statement](#), [Day](#), [Hour](#), [Minute](#), [Month](#), [Now](#), [Time Function](#), [Second](#), [Weekday](#)

# Glossary

---

**Call by reference** In a Basic script, arguments passed by reference to a procedure may be modified by the procedure. Procedures written in Basic are defined to receive their arguments by reference. If you call such a procedure and pass it a variable, and if the procedure modifies its corresponding formal parameter, it will modify the variable. Passing an expression by reference is acceptable in Basic; if the called procedure modifies its corresponding parameter, a temporary value will be modified, with no apparent effect on the caller.

**Call by value** In a Basic script, when an argument is passed by value to a procedure, the called procedure receives a copy of the argument. If the called procedure modifies its corresponding formal parameter, it will not affect the caller. Procedures written in other languages such as C may receive their arguments by value.

**Comment** In a Basic script, a comment is text that documents a program. Comments have no effect on the program (except for metacommands). In Basic, a comment begins with a single quotation mark ('), and continues to the end of the line. If the first character in a comment is a dollar sign (\$), the comment will be interpreted as a metacommand. Lines beginning with the keyword Rem are also interpreted as comments.

**Dialog control** An item in a dialog box, such as a list box, combo box, or command button.

**Function** In a Basic script, a procedure which returns a value. In Basic, the return value is specified by assigning a value to the name of the function, as if the function were a variable.

**Label** In a Basic script, a label identifies a position in the program at which to continue execution, usually as a result of executing a GoTo statement. To be recognized as a label, a name must begin in the first column, and must be immediately followed by a colon (:). Reserved words are not valid labels.

**Metacommand** In a Basic script, a command that gives the compiler instructions on how to build the program. In Basic, metacommands are specified in comments that begin with a dollar sign (\$).

**Name** In a Basic script, a name must start with a letter (A through Z). The remainder of a name can also contain numeric digits (0 through 9) or an underscore (\_). A name cannot exceed 40 characters in length. 'Type characters' are not considered part of a name.

**Precedence order** In a Basic script, the system SBL uses to determine which operators in an expression to evaluate first, second, and so on. Operators with a higher precedence are evaluated before those with lower precedence. Operators with equal precedence are evaluated from left to right. The default precedence order, from high to low, is numeric, string, comparison, logical.

**Procedure** In a Basic script, a series of SBL statements and functions executed as a unit. Both subprograms (Sub) and functions (Function) are called procedures.

**SBL** The acronym for the Softbridge Basic Language (SBL).

**Subprogram** In a Basic script, a procedure that does not return a value.

**Type character** In a Basic script, a special character used as a suffix to the name of a function, variable, or constant. The character defines the data type of the variable or function. The characters are:

- Dynamic String.....\$
- Integer..... %
- Long integer .....&
- Single: single precision floating point..... !
- Double: double precision floating point...#
- Currency exact fixed point .....@

**Vartype** In a Basic script, the internal tag used to identify the type of value currently assigned to a variant. One of the following:

- Empty .....0
- Null.....1
- Integer.....2
- Long .....3
- Single.....4
- Double .....5
- Currency .....6
- Date .....7
- String .....8
- Object .....9