EXTRA! Basic
Language Reference

Manual #221117 First Edition: December, 1995

U.S. GOVERNMENT RESTRICTED RIGHTS

Table of Contents

Chapter 2: Language Summary

Related Documentation

In addition to the *EXTRA! Basic Language Reference*, EXTRA! Personal Client includes the following documentation:

Manuals

- *EXTRA! Personal Client User's Guid*e, which includes information about 3270, 5250 and VT file transfer, customizing sessions, printing, and macros.
- *OLE Automation Programmer's Reference*, which describes the objects, methods, and properties of EXTRA! Personal Client. With these, you can programmatically access the product using OLE Automation. The manual introduces you to key OLE Automation concepts as they relate to EXTRA! Personal Client, and serves as a reference to the product's objects, properties, and methods.

These manuals can be purchased separately or as a set.

Online Help

EXTRA! Personal Client includes a comprehensive set of online help, including help for the EXTRA! Basic macro language and OLE Automation. If you use EXTRA! Basic to access the EXTRA! Personal Client objects, you will find the context-sensitive help in the Macro Editor especially useful. You can access this help in the following ways:

- Display a detailed online Help topic about a particular EXTRA! Basic or OLE Automation element. Highlight the function, statement, or call in the Editor workspace and press F1. The Help Index is displayed pointing to the highest-level entry pertaining to the highlighted language item or object. Choose an appropriate sub-entry, if desired, and then choose the Display button.

  -or-

  From the Macro Editor Functions and Objects browser, highlight an EXTRA! Basic or OLE Automation element and choose the question mark button.
- Click the Help button that is located in each Macro Editor dialog box,
- Activate "What's This" mode by selecting an item in a Macro Editor dialog box and then clicking the right mouse button, or

- Click on the Help Mode toolbar button, then click on an element in the Macro Editor user interface.

Let Us Know How We're Doing

Please fill out the Customer Documentation Feedback Card that comes with your EXTRA! Personal Client software package to let us know what you think about the documentation. Your comments are valuable to us.

CHAPTER 1

Language Overview

Overview: EXTRA! Basic Language

If you've worked with macros or done some Basic programming before, much of the material here will be familiar to you. If you haven't had any experience with macros or programming in Basic, you may find it helpful to consult a book on fundamental programming skills or get help from your system administrator.

You can create a macro by recording a series of actions in *EXTRA! Personal Client*, writing it from scratch using the EXTRA! Basic Language, or by recording some parts and writing others. Recording is best for simple macros, while more complex macros typically use macro instructions that cannot be recorded.

Note: Each EXTRA! Basic statement, function, and OLE automation call online Help topic includes an example. To copy the example code into your macro code, choose the Copy button in the example window, then position the cursor in the Editor workspace and choose Paste from the Edit menu. Then rework the example as needed.

Whether you record a macro or use the Macro Editor to create a macro, it's important that the commands you use are carefully worded and punctuated. Each command must conform to specific rules, or syntax. If the syntax is incorrect, your macro won't compile.

The list below will help you understand the syntax that EXTRA! Basic uses for its macros:

- A function line always starts with a variable assignment. Everything that follows the function name is enclosed in parentheses.
- You can include text strings, numbers, and other required or optional parameters within the parentheses. Separate each item with a comma.
- Text strings are surrounded by quotation marks. For example, ‘‘This is a text string.‘‘ Notice that the period within the quotation marks is part of the string.
- Comments begin with an apostrophe (‘) or the letters REM, and end at the end of the line. You can start a comment at the beginning of a line or place it at the end, but you can't embed a comment within a function line. Once a comment begins, the rest of the function line is ignored when the macro is running.

About the Editor

EXTRA! Personal Client provides the Macro Editor to write and edit macros. The Macro Editor is an editing environment in which you can type your own commands, or insert function, statement, or object templates by choosing from a Functions and Objects list. Refer to the *EXTRA! Personal Client User's Guide* for details about using the Macro Editor.

One of the nice things about using the templates provided by the Macro Editor is that they already contain the correct syntax. All you have to do is supply actual variable names or strings as required.

Using the Macro Editor, you can also debug your macro by:

- Printing your macro source code or viewing it in the Macro Editor.
- Inserting a breakpoint by clicking on the sidebar directly opposite a function or statement. (Breakpoints can be toggled on or off during the debugging process.)
- Single-stepping through your macro code.
- Watching specific variable values as your macro runs.
- Compiling your macro and checking for syntax errors.

Tip: To display a detailed online Help topic about a particular EXTRA! Basic or OLE element, highlight the function, statement, or call in the Editor workspace and press F1. The Help Index is displayed pointing to the highest-level entry pertaining to the highlighted language item or object. Choose an appropriate sub-entry, if desired, and then choose the Display button.

To start the Macro Editor

1. Choose the Macro Editor icon from the *EXTRA! Personal Client* program group in the Program Manager or the *EXTRA! Personal Client* folder in the Windows 95 Explorer.

   –or–

   Choose Macro from the Tools menu of *EXTRA! Personal Client*.

   The Macro dialog box appears.
2. To edit an existing macro, highlight the desired macro filename and choose the Edit button.

   –or–

To create a new macro, enter the name of the new macro in the Macro name text field and choose the Edit button.

The Macro Editor is displayed.

To exit the Macro Editor

1. Make sure you have saved your macro.
2. Choose Exit from the File menu.

Typographic Conventions used in the EXTRA! Basic Language Descriptions

EXTRA! Basic uses the following typographic conventions:

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

EXTRA! Basic Arguments

Arguments to subroutines and functions are listed after the subroutine 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 subroutine or function (that is, 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 subroutine 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 subroutine, 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.

EXTRA! Basic Named Arguments

When you call a subroutine or function that takes arguments, you usually supply values for those arguments by listing them in the order shown in the syntax. For example, suppose you define a function this way:

```
myfunction(id, action, value)
```

From the above syntax, you know that the function called myfunction requires three arguments: *id*, *action*, and *value*. When you call this function, you supply those arguments in the order shown. If the function contains just a few arguments, it is fairly easy to remember the order of each of the arguments. However, if a function has several arguments, and you want to be sure the values you supply are assigned to the correct arguments, use named arguments.

Named arguments are arguments identified by name rather than by position in the syntax. To use a named argument, use the following syntax:

```
namedarg := value
```

Using this syntax for myfunction:

```
myfunction id:=1, action:=‘‘get‘‘, value:=0
```

A further advantage of named arguments is that you don’t need to remember the original order as they were listed in the syntax, so the following function call is also correct:

```
myfunction action:=‘‘get‘‘, value:=0, id:=1
```

With named arguments, order is not important.

The other significant advantage to named arguments is when you call functions or subroutines that have a mix of required and optional arguments. Ordinarily, you need to use commas as placeholders in the syntax for the optional arguments that you do not use. With named arguments, however, you can specify just the arguments you want to use and their values and disregard their order in the syntax. For example, if myfunction is defined as:

```
myfunction(id, action, value, Optional counter)
```

you can use named arguments as follows:

```
myfunction id:=‘‘1‘‘, action:=‘‘get‘‘, value:=‘‘0‘‘
```

or,

```
myfunction value:=‘‘0‘‘, counter:=‘‘10‘‘,
action:=‘‘get‘‘, id:=‘‘1‘‘
```

Note: Although you can shift the order of named arguments, you cannot omit required arguments.

All EXTRA! Basic functions and statements accept named arguments. The argument names are listed in their syntax for the statement and function.

The EXTRA! Basic Line Continuation Character

Long statements can be continued across more than one line by typing a space followed by an underscore character at the end of a line and continuing the statement on the next line. (You can add a comment after the underscore.) For example:

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

How EXTRA! Basic Compares to Other Versions of Basic

If you are familiar with older versions of Basic (those that predate Windows), you will notice that EXTRA! Basic includes many new features and changes from the language you have learned. EXTRA! Basic more closely resembles other higher level languages popular today, such as C++ and Pascal.

Line Numbers and Labels

Older versions of Basic require numbers at the beginning of every line. More recent versions do not support these line numbers; in fact, they will generate error messages. In place of line numbers, EXTRA! Basic calls functions and subroutines by name.

If you want to reference a line of code, you can use a label. A label can be any combination of text and numbers. Usually, it is a single word followed by a colon, placed at the beginning of a line of code.

Subroutines and Language Modularity

EXTRA! Basic is a modular language; code is divided into subroutines and functions. The subroutines and functions you write use the EXTRA! Basic statements and functions to perform actions.

In EXTRA! Basic, the first subroutine must be named ‘‘main’‘; it cannot take any arguments or contain any parentheses. To define it, you use the Sub...End Sub statements, as follows:

```
Sub Main
   ‘body of code here
End Sub
```

The Main subroutine can then call other subroutines or functions included in an EXTRA! Basic file.

Global Variables

The placement of variable declarations determines their scope as follows:

Local: Dimensioned within 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 macro.

Global: Dimensioned outside any subroutine or function using the Global statement. The variable is accessible to any subroutine or function in any module (macro).

Data Types

Modern Basic is now a typed language. In addition to the standard data types -- numeric, string, array, and record -- EXTRA! Basic includes variants and objects.

Variables that are defined as variants can store any type of data. For example, the same variable can hold integers one time, and then, later in a procedure, it can hold strings.

Objects give you the ability to manipulate complex data supplied by an application, such as windows, forms, or OLE objects.

Dialog Box Handling

EXTRA! Basic contains extensive dialog box support to give you great flexibility in creating and running your own customized dialog boxes. You define a dialog box with dialog control statements between the Begin Dialog...End Dialog statements, and then display it using the Dialog statement (or function).

EXTRA! Basic stores information about the selections the user makes in the dialog box. When the dialog box is closed, you can view this information.

EXTRA! Basic also includes statements and functions to display other types of boxes: message boxes notify the user of an event; password boxes do not echo the user’s keystrokes on the screen; and input boxes prompt for a single line of input.

Financial Functions

EXTRA! Basic includes a list of financial functions, to calculate, for example, loan payments, internal rates of return, or future values based on 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 functions (such as finding the date for next Tuesday).

Object Handling

Windows includes OLE 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 EXTRA! Basic code to access another software application through its objects and change those objects.

Environment Control

EXTRA! Basic 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$).

How EXTRA! Basic Compares to Visual Basic and Word Basic

There are several versions of Basic with which you may be familiar: the most common are Microsoft's Visual Basic and Word Basic. EXTRA! Basic shares a substantial common core of functions and statements with these versions; however, each one has unique capabilities.

Functions and Statements Unique to EXTRA! Basic

EXTRA! Basic offers the following statements and functions not found in the standard version of VB: $CStrings, $Include, $NoCStrings, Assert, GetField$, SetField$, and With.

Control-Based Objects

EXTRA! Basic does not include a VB form of control-based objects. As a result, a VB property like ‘‘BorderStyle’‘ is not an intrinsic part of EXTRA! Basic. This does not mean that you cannot define an EXTRA! Basic object that has BorderStyle as a property. You will probably define many objects that are instrinsic to your application in the process of integration.

Dialog Box Capabilities and VBA

VB does not have a syntax to create or run dialog boxes. EXTRA! Basic has a set of functions and statements to enable the use of dialog boxes (similar to those in Word).

Microsoft offers a modified version of VB in some of its products, such as Excel. Called Visual Basic for Applications (VBA), this version does provide dialog box handling statements and functions.

Differences Between EXTRA! Basic and Word Basic

Word Basic is a version of Visual Basic that is included in Microsoft Word. Word Basic supports dialog boxes, but it does not support objects.

Dialog Box Capabilities

The dialog box capabilities in EXTRA! Basic and Word are very similar. Word does offer some statements and functions that EXTRA! Basic does not, such as DlgFilePreview. As well, EXTRA! Basic offers some features that Word does not: Button, ButtonGroup, Caption, DropComboBox, and StaticComboBox.

Button vs. PushButton

Button is the original EXTRA! Basic syntax; PushButton is the Word Basic syntax. The two are interchangeable, and EXTRA! Basic supports both.

Dialog Box Units

The measurement units used in the two dialog box syntaxes are different. EXTRA! Basic supports both methods.

User Input Mechanisms

There are slight differences in some of the mechanisms for user input:

EXTRA! Basic	**Word Basic**
StaticComboBox or ComboBox (in EXTRA! Basic, these are interchangeable)	ComboBox (Word Basic supports only this syntax)
DropComboBox	N/A

EXTRA! Basic Data Types

EXTRA! 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. EXTRA! Basic supports standard Basic numeric, string, record, and array data. EXTRA! Basic also supports Dialog Box Records and Objects (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)*.

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	-1.401298e-45,
	0.0,	
	1.401298e-45	3.402823466e+38
Double	-1.797693134 862315d+308	-4.94065645841247d-308,
	0.0,	
	2.2250738585072014 d-308	1.797693134862315d+308
Currency	-922,337,203, 685,477.5808	922,337,203,685,477.5807

Numeric values are always signed.

EXTRA! Basic has no true Boolean variables. EXTRA! 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 that 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.

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 that can be included in a string. For example, the character whose ANSI value is 0 can be embedded in strings.

The 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 can 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)

Type/Name	Size of Data	Range
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 type Empty, 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 can 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 can 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.

Data Type Conversion

EXTRA! 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).

EXTRA! Basic will also automatically convert between fixed strings and dynamic strings. When converting a fixed string to dynamic, a dynamic string that has the same length and content 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.

EXTRA! Basic will automatically convert between any data type and a variant. EXTRA! 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, EXTRA! 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, 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
```

Creating a Dialog Box Without the Dialog Editor

Most, if not all, of the time you will want to use the Dialog Editor to design and generate code for your macro dialog boxes. Using the language alone, however, it is possible to code a dialog box without the Dialog Editor. This section gives you the basic procedure you will need to follow:

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.

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 action. The body of the function uses the ‘‘Dlg’‘-prefixed EXTRA! Basic 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).

3: Display the dialog box.

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.

Dialog Functions and Statements

The function you create uses ’‘Dlg’‘ dialog functions and statements to manipulate the active dialog box. This is the *only* function that can use these functions and statements. The ’‘Dlg’‘ functions and statements are:

DlgControlId Function: Returns the numeric ID of a dialog control.

DlgEnable Function: Tells whether a control is enabled or disabled.

DlgEnable Statement: Enables or disables a dialog control.

DlgFocus Function: Returns the ID of the dialog control with input focus.

DlgFocus Statement: Sets focus to a dialog control.

DlgListBoxArray Function: Returns the contents of a list box or combo box.

DlgListBoxArray Statement: Sets the contents of a list box or combo box.

DlgText Function: Returns the text associated with a dialog control.

DlgText Statement: Sets the text associated with a dialog control.

DlgValue Function: Returns the value associated with a dialog control.

DlgValue Statement: Sets the value associated with a dialog control.

DlgVisible Function: Tells whether a control is visibled or disabled.

DlgVisible Statement: Shows or hides a dialog control.

Most of these functions and statements take a 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 selected, 0 if not. Note that the IDs are case-sensitive and do not include the period that 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 above is 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

EXTRA! Basic contains three error handling statements and functions for trapping errors in your program: Err, Error, and On Error. EXTRA! Basic returns a code for many of the possible runtime errors you may encounter.

In addition to the errors trapped by EXTRA! Basic, 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 EXTRA! Basic-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.

Trappable Errors

The following table lists the runtime errors that EXTRA! Basic 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.

Code	Error Description
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
62	Input past end of file
63	Bad record number
64	Bad file name
68	Device unavailable

Code	Error Description
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. 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: numeric operators, string operators, comparison operators, and then 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.

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 Operator	**Operand**
Not (Unary Not)	Operand can be Integer or Long. The operation is performed bitwise (one's complement).
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 Long. The operation is performed bitwise. (A Imp B) is the same as ((Not A) OR B).

Object Handling

In EXTRA! Basic, 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. Next, you attach an object name (with or without properties and methods) to the variable to manipulate the object.

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	Sec(x) = 1 / Cos(x)
CoSecant	CoSec(x) = 1 / Sin(x)
CoTangent	CoTan(x) = 1 / Tan(x)
ArcSine	ArcSin(x) = Atn(x / Sqr(-x*x+1))
ArcCosine	ArcCos(x) = Atn(-x / Sqr(-x*x+1))+1.5708
ArcSecant	ArcSec(x) = Atn(x / Sqr(x*x-1))+Sgn(x-1)*1.5708
ArcCoSecant	ArcCoSec(x) = Atn(x / Sqr(x*x-1))+(Sgn(x)-1)*1.5708
ArcCoTangent	ArcTan(x) = Atn(x)+1.5708
Hyperbolic Sine	HSin(x) = (Exp(x)-Exp(-x)) / 2
Hyperbolic Cosine	HCos(x) = (Exp(x)+Exp(-x)) / 2
Hyperbolic Tangent	HTan(x) = (Exp(x)-Exp(-x)) / (Exp(x)+Exp(-x))
Hyperbolic Secant	HSec(x) = 2 / (Exp(x)+Exp(-x))
Hyperbolic CoSecant	HCoSec(x) = 2 / (Exp(x)-Exp(-x))
Hyperbolic Cotangent	HCotan(x) = (Exp(x)+Exp(-x)) / (Exp(x)-Exp(-x))
Hyperbolic ArcSine	HArcSin(x) = Log(x+Sqr(x*x+1))
Hyperbolic ArcCosine	HArcCos(x) = Log(x+Sqr(x*x-1))
Hyperbolic ArcTangent	HArcTan(x) = Log((1+x) / (1-x)) / 2

Function	Computed By
Hyperbolic ArcSecant	HArcSec(x) = Log((Sqr(-x*x+1)+1) / x)
Hyperbolic ArcCoSecant	HArcCoSec(x) = Log((Sgn(x)*Sqr(x*x+1)+1) / x)
Hyperbolic ArcCoTangent	HArcCoTan(x) = Log((x+1) / (x-1)) / 2

EXTRA! Basic Macro File Formats

The EXTRA! Basic Macro Editor can save files in three formats, and distinguishes these formats by filename extension:

- EXTRA! Basic Macro files (.EBM)
- EXTRA! Basic Header files (.EBH)
- ANSI Text files (.TXT)

EXTRA! Basic Macro files are saved in binary form, and can be saved compiled or uncompiled. The Macro Editor is required to display these files in a ‘‘readable’‘ (that is, source code) format.

EXTRA! Basic Header files are ‘‘include’‘ files that contain variables or function definitions that supplement another macro. The header file is compiled when the main macro is compiled. The **‘$Include** statement is used in a main macro to include the header.

Text files are normal ANSI text files. These files may be created and saved as in any other text editor. This format is included as a convenience.

A fourth file format is shown in the File Open dialog box of the Macro Editor. This is the .EWM format which is used by 3.x and higher versions of EXTRA! for Windows Macros. When these macros are opened in the *EXTRA! Personal Client* Macro Editor, they are automatically converted to .EBM formats. The original .EWM file is not altered, so after loading, editing, and saving an old .EWM macro in the Macro Editor the you will see both an .EBM and a .EWM file with the same name in your macro directory.

CHAPTER 2 *Language Summary*

Abs (function) — Return absolute value

rc% = **Abs(***numeric-expression***)**

Returns the absolute value (unsigned magnitude) of the specified numeric expression.

Parameter	**Description**
*rc***%**	The return value.
numeric-expression	An expression of any numeric data type

Comments

The data type of the return value is the same as the data type of the *numeric-expression* parameter. This includes variant expressions that 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).

If *numeric-expression* results in a Null, **Abs** returns a Null.

Abs (function) Example

This example finds the approximate value for a cube root. It uses Abs to determine the absolute difference between two numbers.

```
Randomize
Precision# = .0000001#
Value# = Val(InputBox$("Enter a value: "))  ' Prompt for input.
Value# = 100.0*Rnd(1) + 1.0
X1# = 0#: X2# = Value#                      ' Make first two guesses.
' Loop until difference between guesses is less
‘ than required precision.
Do Until Abs(X1# - X2#) < Precision#
   X# = (X1# + X2#) / 2#
   If X# * X# * X# - Value# < 0# Then       ' Adjust guesses.
       X1# = X#
   Else
       X2# = X#
   End If
Loop
Msg$ = "The cube root of " + LTrim$(Str$(Value#)) + " is "
Msg$ = Msg$ + LTrim$(Str$(X#))
MsgBox Msg$                                ' Display message.
```

AppActivate (statement) — Activates an application window

`AppActivate` *`string-expression`*

Activates an application window.

Parameter	**Description**
string-expression	The name (from the title bar) of the application you want to activate. Both the title bar and string-expression must be spelled identically. However, the comparison is not case sensitive.

Comments

AppActivate changes the focus to the specified window but will not automatically maximize an application if it is currently minimized. **AppActivate** can be used in combination with the **SendKeys** statement to send keystrokes to another application.

Note: If there is more than one application window named *string-expression*, EXTRA! Basic chooses randomly from the pool of matching windows.

AppActivate (statement) 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
```

Asc (function) — Return ANSI code

`rc% = Asc(string-expression$)`

Returns an integer corresponding to the ANSI code of the first character in the specified string or string expression.

Parameter	**Description**
rc%	The return value.
string-expression$	A string whose first character is evaluated

Comments

If *string-expression*$ is Null, EXTRA! generates an error.

Asc (function) 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
```

Assert (statement) — Trigger error if condition is false

```
Assert condition
```

Triggers an error if the specified *condition* parameter is false.

Parameter	**Description**
condition	Any expression that evaluates to True or False

Comments

Provides verification that a procedure is performing in the expected manner and notification if it is not.

An assertion error cannot be trapped by the **On Error** statement.

Atn (function) — Returns arctangent of number

```
rc% = Atn(numeric-expression)
```

Evaluates the ratio of two sides of a right triangle and returns the corresponding angle in radians.

Parameter	**Description**
rc	The return value.
numeric-expression	Any numeric data type that describes the ratio of the side opposite and the side adjacent to a right triangle.

The following table shows how the return parameter is converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

Comments

To convert radians to degrees, multiply radians by 180 / Pi (or 57.2957795130824) where Pi equals 3.141593.

To convert degrees to radians, multiply degrees by Pi / 180 (or .0174532925199433) where Pi equals 3.141593.

Note: The arctangent of a numeric expression is the inverse of the tangent. Do not confuse cotangent (the simple inverse of a tangent) with arctangent.

Atn (function) ExampleThe example uses Atn to calculate Pi. By definition, Atn(1) is 45 degrees; 180 degrees equals Pi radians.

```
Sub Main

   Dim Msg$, Pi                  ' Declare variables.
```

```
Pi = 4 * Atn(1)               ' Calculate Pi.
Msg$ = "Pi is equal to " + STR$(Pi)
MsgBox Msg$                   ' Display results.
```

Beep (statement) — Generate a beep

Beep

Produces a single short beeping tone through the computer's speaker.

Comments

Your computer's hardware dictates the frequency and duration of the beep produced by the **Beep** statement.

Beep (statement) Example

This example uses Beep to sound a tone in the computer speaker if Answer is less than 1 or greater than 3.

```
x$ = "Enter a value from 1 to 3."  ' Set prompt message.
Default$ = "1"                     ' Set default value.
Do
   Answer$ = InputBox$(x$,"BeepDemo", Default$)
                                   ' Get user input.
      If Len(Answer$) > 0 Then     ' Check for null.
         Value% = Val(Answer$)     ' Convert to number.
         If Value% >= 1 And Value% <= 3 Then
                                   ' Check range.
            Msg$ = "You entered a value in the proper range."
            Exit Do                ' Exit Do Loop.
         Else
            Beep                   ' Beep if not in range.
         End If
      Else
         Msg$ = "You pressed cancel."
         Exit Do                   ' Exit Do Loop.
      End If
Loop
MsgBox Msg$                        ' Display results.
```

Begin Dialog...End Dialog (statement) — Dialog box defintion

```
Begin Dialog DialogName [x, y,] dx, dy _
   [, caption$] [,.dialogfunction]
   'dialog-box-definition-statements
End Dialog
```

Initiates the dialog-box declaration for a user-defined dialog box.

Parameter	**Description**
DialogName	Identifies the current dialog box definition
x, y	Coordinate location of the dialog box's upper left corner (relative to the upper left corner of the parent window's client area). *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. Example If *x* is omitted, the dialog box is centered horizontally within the client area. If *y* is omitted, the dialog box is centered vertically within the client area.
dx, dy	*dx* and *dy* specify the width and height of the dialog box (relative to the *x* and *y* coordinates). The *dx* argument is measured in one-quarter system-font character-width units. The *dy* argument is measured in one-eighth system-font character-width units.
*caption***$**	*caption***$** defines the text displayed on the title bar of the dialog box. If *caption***$** is omitted, the default title bar (EXTRA! Basic Language) will be used.

Parameter	**Description**
.dialog function	*.dialogfunction* is the name of an EXTRA! Basic function associated with the dialog box. The Basic function you reference must have been defined or declared before the Begin Dialog statement. Refer to the *dialogfunction* section below for details.
dialog-box-definition statements	Any of the following*:* **Button**, **ButtonGroup**, **CancelButton**, **Caption**, **CheckBox**, **ComboBox**, **GroupBox**, **ListBox**, **OKButton**, **OptionButton**, **OptionGroup**, **Text**, and **TextBox**

Comments

Assumes that if only two arguments are supplied, they are the *dx* (width) and *dy* (height) arguments.

Unless the **Begin Dialog** statement is followed by at least one other dialog-box definition statement followed by the **End Dialog** statement, an error will result. Further, the dialog-box-definition-statement must include an **OkButton**, **CancelButton**, or **Button** statement. If this statement is omitted, the dialog box cannot be closed, and the procedure will halt.

To display the dialog box, create a dialog record variable with the **Dim** statement, and then display the dialog box using the **Dialog** statement. In the **Dim** statement, *DialogName* is used to identify the dialog definition.

***Dialogfunction* Parameter Details**

Declare the *dialogfunction* parameter prior to the **Begin Dialog** statement as follows:

function *dialogfunction***%**(*id***$**, *action***%**, *suppvalue***&**)
    '*function body*
end function

where

*id***$**	Identifies the control associated with the call to the dialog function. This is the same value that appears in the definition of the control.
*action***%**	An integer value from 1 to 5 identifying the reason that the dialog box was called.

1 The dialog box has been initialized. This action value is passed before the dialog box is displayed.

2 The user has chosen a command button or otherwise changed the value of a dialog box control. (If the user types in a text box or combobox, EXTRA! Basic returns an action value of 3, described next.)

3 A change has occured in a text box or combobox. This value is passed when the control loses the input focus (i.e., the user presses the TAB key or chooses another control).

4 A change in control focus has occurred. *id***$** is the value that identifies the control gaining focus, and *suppvalue***&** is the value of the control losing focus. Note that a dialog function cannot display a message box or dialog box in response to an *action***%** value of 4.

5	The user interface is idle. Immediately after the dialog box is initialized (action value 1), the dialog function is passed an action value of 5 until another action occurs. If the dialog box wants to receive this message continuously while the dialog box is idle, it should return a non-zero value. If 0 (zero) is returned, action 5 will be passed only while the user is moving the mouse. For this action, *id*$ is equal to empty string (“‘“) and *suppvalue*& is equal to the number of times action 5 was passed before.

*suppvalue***%** When *action***%** is 2 or 3, *suppvalue***&** depends on the type of the control. Refer to the following table:

Control	*suppvalue*
List box	Number of the item selected (0-based)
Check box	1 if selected, 0 if cleared, -1 if 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	Number of the item selected (0-based) for action 2, or number of characters in the associated text box for action 3
OK button	1

Cancel button	2
Push button	An internal button identifier. Not the same as the *id***$** parameter of the button control.

In most cases, the return value of the dialog function is ignored. The exceptions are the return values from *action***%** 5 (discussed above), and from *action***%** 2. If *action***%** 2 is called because the user chose the OK button, Cancel button, or a command button (as indicated by *id***$**), and the dialog function returns a non-zero value, the dialog box will not be closed.

Begin Dialog...End Dialog (statement) Example

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

```
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, "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
```

Button (statement) — Define a dialog box button

```
Button x, y, dx, dy, text$ [, .id]
```

-or-

```
PushButton x, y, dx, dy, text$ [, .id]
```

Defines a custom push button other than OK or Cancel. (These buttons are defined using the **OkButton** and **CancelButton** statements.) The **Button** statement must be used in conjunction with the **ButtonGroup** statement.

Note: Button and **PushButton** are equivalent.

Parameter	**Description**
x, y	Specifies the position of the button relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the button is centered horizontally within the dialog box. If *y* is omitted, the button is centered vertically within the dialog box.
dx, dy	Specifies the width and height of the button. *dx* is measured in one-quarter system-font character-width units. *dy* is measured in one-eighth system-font character-width units. A *dy* value of 14 typically accommodates system font text.
*text***$**	Supplies the text for the button. If the width of this string is greater than *dx*,

trailing characters are truncated.

Parameter	Description
id$	An optional identifier used by the dialog statements that act on this control.

Comments

The Button statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

Button (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, "E! Basic 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
```

ButtonGroup (statement) — Define a group of dialog box buttons

`ButtonGroup` *`.field`*

Begins the definition of the buttons when custom buttons are used.

Parameter	**Description**
.field	The name of the dialog record field that will hold the value for the button selected by the user

Comments

ButtonGroup creates a dialog-record field that will contain the value for the user's selection. If **ButtonGroup** is used, it must appear before any **Button** or **PushButton** statement. Only one **ButtonGroup** statement is allowed within a dialog box definition.

The **ButtonGroup** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

If OK and Cancel buttons are used in a dialog with the ButtonGroup statement, the ButtonGroup variable must be set to -1. If the variable remains -1, either the OK or Cancel button was pushed. If any other variable is returned, a custom button was pushed.

ButtonGroup (statement) Example

This example defines a dialog box with a group of three buttons.

```
Sub main
   Begin Dialog UserDialog 34,0,231,140, "E! Basic Dialog Box"
      ButtonGroup .bg
      PushButton 71,17,88,17, "&Button 0"
      PushButton 71,50,88,17, "&Button 1"
      PushButton 71,83,88,17, "&Button 2"
   End Dialog
   Dim mydialog as UserDialog
   Dialog mydialog
   Msgbox "Button " & mydialog.bg & " was pressed."
End Sub
```

Call (statement) — Transfer Control to a Subprogram

```
Call     subprogram-name[(argumentlist)]
```

-or-

```
subprogram-name argumentlist
```

-or-

```
Call     app-dialog (recordName)
```

-or-

```
App-dialog {recordName | dotList}
```

The Call statement is used to transfer control to a subprogram procedure, dynamic link library (DLL), or application-defined dialog box.

Parameter	**Description**
subprogram-name	A subprogram written in Basic or a dynamic link library (DLL) containing C procedures. (The C procedures must be described in a **Declare** statement or be implicit in the application.)
argumentlist	Any arguments required by the subprogram. The arguments can be either variables or expressions. For subprograms written in Basic, where arguments are passed by reference: if you pass a variable to a procedure that will in turn modify the variable's corresponding formal parameter, you can enclose the variable in parentheses to keep the formal parameter from being modified. The parentheses prompt Basic to pass a copy of the variable. Note that this method is less efficient, and should not be used unless necessary.

Parameter	Description
app-dialog	Functions associated with application-defined dialog boxes can be invoked using the third or fourth syntax variation shown above. In the third variation, the name inside the parentheses must be a variable previously declared (refer to the **Dim** statement description) as an application-defined dialog record. In the fourth variation, the dialog box name can be followed by either a dialog record variable or a comma-separated list of dialog box field settings. For example: `SearchFind .SearchFor = "abc", .Forward = 1` 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, the **ByVal** keyword is optional in the **Call**. If **ByVal** is present in the declaration, it must precede the value. If **ByVal** is not present, it is illegal in the **Call** unless the datatype specified in the declaration was **Any**. Specifying **ByVal** places the parameter's actual value on the stack instead of a far reference to the value.
recordName	The name of the dialog record variable.
dotList	A comma-separated list of dialog field settings.

Comments

The **Call** keyword is not required when calling a procedure. Note that if you use the **Call** keyword to call a procedure that requires parameters, the parameter list must be enclosed in parentheses. If you do not use the **Call** keyword, do not enclose the parameter list in parentheses.

Call (statement) Example

This example shows two ways to call the MessageBeep procedure in USER.EXE, a Microsoft Windows DLL.

```
Declare Sub MessageBeep Lib "User" (ByVal N As Integer)


Sub Main
   Msg$ = "This demonstration uses the Call statement to call "
   Msg$ = Msg$ + "a procedure in a dynamic-link library. "
   Msg$ = Msg$ + "Choose OK to call the MessageBeep procedure in "
   Msg$ = Msg$ + "USER.EXE using both forms of Call syntax."
   MsgBox Msg$                   ' Display message.
   Call MessageBeep(0)           ' Call Windows procedure.
   For I% = 1 to 50:
      MessageBeep 0              ' Second Call; Insert short delay.
   Next I%
End Sub
```

CancelButton (statement) — Define a cancel button dialog box control

`CancelButton` *`x, y, dx, dy`* `[, .`*`id`*`]`

Determines the position and size of a Cancel button.

Parameter	**Description**
x, y	Specifies the position of the Cancel button relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. The *y* argument is measured in units one-eighth the height of the system font. If *x* is omitted, the Cancel button is centered horizontally within the client area. If *y* is omitted, the button is centered vertically within the client area.
dx, dy	Specifies the width and height of the button. *dx* is measured in one-quarter system-font character-width units. The *dy* argument is measured in one-eighth system-font character-width units. A *dy* value of 14 typically accommodates system font text.
*id***$**	An optional identifier used by the dialog statements that act on this control.

Comments

The **CancelButton** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

If the **Dialog** *statement* was used to start the dialog and the CancelButton is selected, the dialog box is removed and the ‘‘Command failed’‘ error (102) is triggered. If the dialog was started by the **Dialog** *function*, the function will return 0 and the error will not occur.

CancelButton (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, "E! Basic 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
```

Caption (statement) — Assign a title to a dialog box

```
Caption text$
```

Defines title text of your dialog box.

Parameter	**Description**
*text***$**	Supplies the title text for the dialog box defined in the **Begin Dialog** structure.

Comments

The **Caption** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

If you do not specify a **Caption** for your dialog box, the default title ‘‘EXTRA! Basic Language’‘ is used.

Caption (statement) Example

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’‘.

```
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
```

CCur (function) — Return a value as currency

`rc% = CCur(expression)`

Returns a value in the form of currency.

Parameter	**Description**
rc%	The return value.
expression	Any valid string or numeric expression.

Comments

If you attempt to convert a numeric expression that is not within the acceptable range, an Overflow Error will occur. Strings that cannot be converted to currency will result in a Type Mismatch error and variants containing Nulls will result in an Illegal Use of Null error.

CCur (function) 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
```

CDbl (function) — Convert to double precision

rc = **CDbl(***numeric-expression***)**

Converts the specified numeric expression to a double-precision floating point value.

Parameter	**Description**
rc	The return value.
numeric-expression	Any numeric data type.

Comments

CDbl generates the same result as assigning *numeric-expression* to a Double variable. **CDbl** can be used to force double-precision accuracy in an expression that would ordinarily result in a currency value or an integer or single-precision number.

CDbl (function) Example

The CDbl function converts a Currency value to a Double.

```
Dim CurrVal as Double, CurrPercent, Msg$, Percent
'Declare variables.
Percent = 8.2  ' Set tax rate.
Msg$ = "Enter a currency value."
CurrVal = Val(InputBox$(Msg$))       ' Get user input.
CurrPercent = CDbl((CurrVal) * Percent * .01)       ' Calculate value.
Msg$ = STR$(Percent) + "% of $" + STR$(CurrVal) + ".00"
Msg$ = Msg$ + " is $" +  STR$(CurrPercent)
Msg$ = Msg$ + " expressed as a double-precision number."
MsgBox Msg$    ' Display results.
```

ChDir (statement) — Change the default directory

`ChDir pathname$`

Changes the default directory for the specified drive.

Parameter	**Description**
*pathname***$**	A string expression identifying the new default directory. The syntax for *pathname***$** is: `[drive:][\]directory[\directory]` and must contain fewer than 128 characters. If the drive parameter is omitted, **ChDir** changes the default directory on the current drive.

Comments

ChDir does not change the default drive. To change the default drive, use the **ChDrive** statement.

ChDir (statement) Example

This example uses ChDir to change the default directory on the current drive.

```
NL$ = Chr$(13) + Chr$(10)            ' Define newline.
CurPath$ = CurDir$                   ' Get current path.
ChDir "\"

Msg$ = "The current directory has been changed to "
Msg$ = Msg$ + CurDir$ + NL$ + NL$ + "Press OK to change back "
Msg$ = Msg$ + "to your previous default directory."

Answer% = MsgBox(Msg$)              ' Get user response.
ChDir CurPath$                      ' Change back to user default.
Msg$ = "Directory changed back to " + CurPath$ + "."
MsgBox Msg$
```

ChDrive (statement) — Change the default drive

`ChDrive` *`drivename`***`$`**

Changes the default drive.

Parameter	Description
*drivename***$**	A string expression designating the new default drive. This drive must exist, and must be within the range specified in the CONFIG.SYS file. If a Null parameter (“ “) is supplied, the default drive remains the same. If the *drivename***$** parameter is a string, **ChDrive** uses the first letter only. If the parameter is omitted, an error message is generated.

Comments

To change the current directory on a drive, use the **ChDir** statement.

ChDrive is not case sensitive.

ChDrive (statement) Example

This example changes the currently logged drive to the new drive indicated by the letter entered by the user.

```
On Error Resume Next
NL$ = Chr$(13) + Chr$(10)           ' Define newline.
CurPath$ = CurDir$                  ' Get current path.
If Err = 68 Then                    ' In case current
   Drive$ = "invalid."              ' drive is invalid
   Err = 0                          ' reset error to 0.
Else
   Drive$ = Left$(CurPath$, 2)      ' Get drive letter.
End If

Msg$ = "Your currently logged drive is " + Drive$ + NL$ + NL$
Msg$ = Msg$ + "Enter the letter of another drive to make it the "
Msg$ = Msg$ + "logged drive."
```

```
NewDrive$ = InputBox$(Msg$)         ' Prompt for new drive.
HasColon% = InStr(1, NewDrive$, ":")' Check for colon.

' If there is no colon, append one to NewDrive$.
If Not HasColon% Then NewDrive$ = Left$(NewDrive$, 1) + ":"
ChDrive NewDrive$                   ' Change drive.

Select Case Err
   Case 68                          ' Device unavailable error.
      Msg$ = "That drive is not available. No drive change"
      Msg$ = Msg$ + "was made."
   Case 71                          ' Disk not ready error.
      Msg$ = "Close the door on your drive and try again."
   Case 5                           ' Illegal function call.
      Msg$ = "You probably didn't enter a drive letter."
      Msg$ = Msg$ + "No change was made."
   Case Else
      Msg$ = "Drive changed to " + UCase$(NewDrive$)
End Select

ChDrive Drive$                       ' Change back.
MsgBox Msg$                          ' Display results.
```

CheckBox (statement) — Define a dialog check box

```
CheckBox x, y, dx, dy, text$, .field
```

Displays one or more options that can be toggled on or off, independent of the other options in the group.

Parameter	**Description**
x, *y*	Specifies the position of the check box relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the check box is centered horizontally within the client area. If *y* is omitted, the check box is centered vertically within the client area.
dx, *dy*	The combined width of the check box and the *text***$** field. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text***$** parameter (including blanks and punctuation) by 4 and add 12. *dy* is the height of the *text***$** parameter. 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 check box and the accompanying text will move downward within the dialog box.

Parameter	Description
*text***$**	Contains the text that is displayed to the right of the check box. If the width of this string is greater than *dx*, trailing characters will be truncated. To indicate an accelerator key in *text***$**, precede the accelerator key character with an ampersand (**&**).
.field	The name of the dialog-record field that will hold the current check box setting value. If the check box is selected, *field* is set to 1. If the check box is not selected, *field* is set to 0. If the check box is dimmed, *field* is set to -1. If *field* is set to any value other than -1, 0, or 1, the Macro Editor will treat it as though it is set to 1.

Comments

The **CheckBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

CheckBox (statement) Example

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

```
Sub main
   Dim ComboBox1() as String
   ReDim ComboBox1(0)
   ComboBox1(0)=Dir("C:\*.*")
   Begin Dialog UserDialog 166, 76, "E! Basic 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
```

Chr (function) — Return an ANSI character

*rc***$ = Chr**[**$**]**(***code***%)**

Returns the one-character string corresponding to the specified ANSI code.

Parameter	**Description**
*rc***$**	The return value.
*code***%**	An integer between 0 and 255 (inclusive) corresponding to an ANSI character.

Comments

Applications designed to run under Microsoft Windows use the ANSI character set. Note that ANSI character codes 0 through 31 are the same as the standard, nonprintable ASCII codes.

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

Chr (function) Example

The example uses the Chr function to create a variable that will insert a newline character as well as a variable containing letters from A through Z. Each time the line containing Chr is executed within the For...Next loop, another letter is appended to the Msg variable.

```
NL$ = Chr$(13) + Chr$(10)           ' Define newline.
For I% = 1 to 2                     ' Set 2 iterations.
   For J% = Asc("A") To Asc("Z")    ' From A through Z.
      Msg$ = Msg$ + Chr$(J%)        ' Create a string.
         Next J%
      Msg$ = Msg$ + NL$             ' Insert newline.
Next I%
MsgBox Msg$
```

CInt (function) — Convert to an integer

rc% = **CInt(***numeric-expression***)**

Converts a numeric expression to an integer by rounding down if the decimal value is less than 0.5, and rounding up if the value is greater than 0.5.

Parameter	Description
*rc***%**	The return value.
numeric-expression	Any numeric data type.

Comments

After rounding, the result number must be within the range of -32,767 to 32,767 (inclusive), or an error will occur.

CInt generates the same result as assigning *numeric-expression* to an Integer variable. You can also use the **Int** and **Fix** functions to convert a numeric expression to an integer, however they perform truncation to acheive a whole number instead of rounding.

CInt can be used to force single-precision accuracy in an expression that would ordinarily result in a currency value or a single- or double-precision number.

Strings that cannot be converted to an integer will result in a Type Mismatch error. Variants containing Nulls will result in an Illegal Use of Null error.

CInt (function) Example

The example converts an angle in radians to an angle in degrees and minutes. The CInt function is used to convert a double-precision value to an integer value.

```
Sub Main


Dim Degs, Mins, Pi, Msg$           ' Declare variables.

Dim Rads, Rads2Degs, WholeDegs, WholeMins As Double

Dim RadVal As String

Pi = 4 * Atn(1)                    ' Calculate Pi.
```

```
Rads2Degs = 180 / Pi               ' Calculate conversion factor.

Msg = "Please enter an angle in radians."

RadsVal$ = InputBox$(Msg)
Rads = Val(RadVal)                 ' Get angle in radians.
Degs = Rads * Rads2Degs            ' Convert radians to degrees.
Mins = Degs - Int(Degs)            ' Get fractional part.
WholeMins = CInt(Mins * 60)        ' Convert to between 0 and 60.
WholeDegs = Int(Degs)              ' Get whole number part.
If WholeMins = 60 Then             ' 60 minutes = 1 degree.
   WholeDegs = WholeDegs + 1
WholeMins = 0
End If

Msg$ = "Angle = " + Str$(WholeDegs) + " degrees, "
Msg$ = Msg$ + Str$(WholeMins) + " minutes."
MsgBox Msg$                        ' Display results.

End Sub
```

CLng (function) — Convert to a Long integer

```
Dim rc As Long
rc = CLng(numeric-expression)
```

Converts a numeric expression to a Long 4-byte integer by rounding down if the decimal value is less than 0.5, and rounding up if the value is greater than 0.5.

Parameter	**Description**
rc	The return value.
numeric-expression	Any numeric data type.

Comments

After rounding, the resulting number must be within the range of -2,147,483,648 to 2,147,483,647, or an error will occur.

CLng generates the same result as assigning *numeric-expression* to a Long variable.

CLng can be used to force Long integer arithmetic in an expression that would ordinarily result in a currency value or a single- or double-precision number.

Strings that cannot be converted to an integer will result in a Type Mismatch error. Variants containing Nulls will result in an Illegal Use of Null error.

CLng (function) Example

The example uses the CLng function to convert double-precision numbers to long integers. It also demonstrates how CLng rounds numbers when converting. Note that 25427.45 is rounded down to 25427, whereas 25427.55 is rounded up to 25428.

```
NL$ = Chr (13) + Chr (10)      ' Define newline.

Msg$ = "25427.45 rounds to "

Msg$ = Msg$ + STR$ (CLng(25427.45))

Msg$ = Msg$ + NL$ + "25427.55 rounds to "

Msg$ = Msg$ + STR$ (CLng(24527.55))

MsgBox Msg$                   ' Display results.
```

Close (statement) — Close a file

Close[[**#**]*filenumber%* [,[**#**]*filenumber%*...]]

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

Parameter	**Description**
filenumber%	An integer expression corresponding to a file number assigned in a previous **Open** statement. If this parameter is omitted, all open files are closed. Once a **Close** statement is executed, the association of that file with *filenumber*% ends, and the file can be reopened with the same or different file number.

Comments

When **Close** is issued, 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 to flush these operating system buffers to disk.

Close (statement) Example

This example uses the Close statement to close a file.

```
TestString$ = "The quick brown fox"   ' Create test string.
For I% = 1 To 5
   FNum% = FreeFile                   ' Determine next file number.
   FName$ = "FILEIOX" + LTrim$(Str$(FNum%)) + ".DAT"
   Open FName$ For Output As FNum%    ' Open file.
   Print #I%, TestString$             ' Write string to file.
Next I%
Close                                 ' Close all files.
Msg$ = "Several test files have been created on your disk."
Msg$ = Msg$ + "Choose OK to remove the test files."
MsgBox Msg$
Kill "FILEIOX?.DAT"                   ' Remove test file from disk.
```

ComboBox (statement) — Define a combination text and list box

ComboBox *x, y, dx, dy, text***$**, *.field*

-or-

ComboBox *x, y, dx, dy, stringarray***$**, *.field*

Creates a combination static text box (or edit box) and list box. For example, the File Open dialog box in typical Microsoft applications is a combo-box.

Parameter	**Description**
x, y	Specifies the position of the combination box relative to the upper left corner of the client area. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the combination box is centered horizontally within the client area. If *y* is omitted, the box is centered vertically within the client area.
dx, dy	The sum of the width of the combination box and the *text***$** parameter. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text***$** parameter (including blanks and punctuation) by 4 and add 12. *dy* is the height of the *text***$** parameter. 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 combination box and the accompanying text will move

downward within the dialog box.

Parameter	Description
text$	A string containing the selections for the combobox. This string must be defined, using a **Dim** statement, before the **Begin Dialog** statement is executed. The arguments in the *text*$ string are entered as follows: *dimname* = ‘‘*listchoice*’‘ + **Chr$**(9) = ‘‘*listchoice*’‘ + **Chr$**(9) + ‘‘*listchoice*’‘ + **Chr$**(9)...
stringarray	An array of dynamic strings.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box. The user's selection is recorded in the field designated by the *.field* argument when the OK button (or any button other than Cancel) is selected. The *.field* argument is also used by the dialog statements that act on this control.

Comments

The **ComboBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

ComboBox (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, "E! Basic 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
```

Command (function) — Returns a command line as a String

rc = **Command**[**$**]

Returns a string containing the command line specified when the MAIN subprogram was invoked.

Parameter	**Description**
rc	The return value.

Comments

The dollar sign (**$**) is optional. If **$** is specified, the return type is String. If omitted, the function will return a variant of vartype 8 (String).

After the MAIN subprogram returns, further calls to the **Command** function will yield an empty string.

Command (function) Example

This example uses Command to display the arguments on the command line.

```
NL$ = Chr$(13) + Chr$(10)          ' Define newline.
If Command$ = "" Then              ' If no Command$,
   Msg$ = "There is currently no command-line string."
Else                               ' put Command$ into message.
   Msg$ = "The command line string is: "
   Msg$ = Msg$ + NL$ + NL$ + "'" + Command$ + "'"
End If
MsgBox Msg$                        ' Display message.
```

Const (statement) — Declare a symbolic constant

```
 [Global] Const constantName = expression
[, constantName = expression]...
```

Declares a symbolic constant for use in an EXTRA! Basic macro.

Parameter	Description
constant Name	Supplies the name of a symbolic constant plus a suffix specifying the constant's type (**%**, **&**, **!**, **#**, **@**, or **$**). Example If no data type is specified, the type is derived from *expression*. A string will always result in a string constant, however, numeric expressions are evaluated and the constant is assigned the simplest type.
expression	The expression that is assigned to the constant. This expression can consist of literals (for example, 1.5), other declared constants, and any arithmetic or logical operator except the exponentiation operator(^). *expression* can also be a single literal string like ‘‘Invalid input. Try again.’‘ **Const**-declared symbolic constants cannot consist of concatenated strings, user-defined functions, or any of the built-in Macro Editor functions.

Comments

If a constant is declared in a Sub or Function procedure, it is considered local to that procedure. A constant declared in the Declarations section of a module is defined throughout the module in which it is declared. Constants declared outside a procedure with the Global reserved word can be used by all procedures in all modules.

Note: 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 will fail if a mismatch is found. This is an effective way to detect version mismatches between modules.

Const (statement) Example

This example uses Const to define the symbolic constant PI.

```
Const PI = 3.141592654            ' Define constant.
Msg$ = "Enter the radius of a circle in centimeters."

Radius# = Val(InputBox$(Msg$))    ' Get user input.
Radius# = Rnd(1)
Circum# = 2 * PI * Radius#        ' Calculate circumference.
Area# = PI * (Radius# ^ 2)        ' Calculate area.

Msg$ = "The circumference of the circle is "
Msg$ = Msg$ + LTrim$(Str$(Circum#)) + " cm. Its area is "
Msg$ = Msg$ + LTrim$(Str$(Area#)) + " cm."
MsgBox Msg$                       ' Display message.
```

Cos (function) — Return the Cosine of an Angle

`rc% = Cos(angle)`

Returns the cosine of an angle.

Parameter	**Description**
rc%	The return value.
angle	The angle, specified in radians. Can be either positive or negative.

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision Integer
!	Single	Single-precision floating point
&	Long	Double-precision Integer
#	Double	Double-precision floating point

Comments

The **Cos** function takes an angle and returns the ratio of the side adjacent to the angle divided by the length of the hypotenuse of the triangle. The return value will be between -1 and 1.

To convert radians to degrees, multiply radians by 180/Pi (or 57.2957795130824) where Pi equals 3.141593.

Cos (function) Example

This example uses Cos to calculate the cosine of an angle with a user-specified number of degrees.

```
Pi# = 4 * Atn(1#)                 ' Calculate Pi.


Msg$ = "Enter an angle in degrees."
```

```
Degrees# = Val(InputBox$(Msg$))   ' Get user input.
Degrees# = 45
Radians# = Degrees# * (Pi# / 180) ' Convert to radians.

Msg$ = "The cosine of a " + LTrim$(Str$(Degrees#))
Msg$ = Msg$ + " degree angle is "
Msg$ = Msg$ + LTrim$(Str$(Cos(Radians#))) + "."
MsgBox Msg$                       ' Display results.
```

CreateObject (function) — Create an OLE automation object

CreateObject(*class*)

Creates a new OLE Automation object. You can use CreateObject to run an instance of *EXTRA! Personal Client*. The objects that you can access in *EXTRA! Personal Client* are referred to as the *EXTRA! Personal Client* Objects.

Parameter	**Description**
class	A string indicating the name of the application used to create the object and the type of object. To specify the class, use the following syntax: **"***AppName.ObjectType***"** Note that you must insert a period (.) between *AppName* and *ObjectType.*

AppName	For *EXTRA! Personal Client*, the *AppName* is EXTRA.
ObjectType	For *EXTRA! Personal Client*, the only *ObjectType* that can be created is System.

Comments

If *EXTRA! Personal Client* is not already running, CreateObject starts the program; otherwise, CreateObject uses the existing instance of the program.

To access an object returned by CreateObject, use the Set statement to assign an object reference to a variable. In the example below, an object variable called extra is declared. Then, with the Set statement, the EXTRA! System object, returned by CreateObject, is assigned to the variable extra.

```
Dim extra As Object
Set extra = CreateObject("Extra.System")
```

In addition to starting *EXTRA! Personal Client*, CreateObject starts any application that supports OLE Automation. Such applications include Microsoft Excel 5.0 or later and Microsoft Project 4.0 or later. For information on accessing an application’s OLE Automation objects, see the documentation for that application.

CreateObject (function) Example

This example starts an instance of *EXTRA! Personal Client*, if one is not already running.

```
Sub Main()
  Dim extra As Object
  Set extra = CreateObject("Extra.System")
  MsgBox extra.name+" is now running."
End Sub
```

CSng (function) — Convert to single precision

rc! = **CSng(***numeric-expression***)**

Converts a numeric-expression to single-precision floating point.

Parameter	**Description**
rc!	The return value.
numeric-expression	Any numeric data type. The **CSng** function will round the resulting value, if necessary, before converting it.

Comments

numeric-expression must have a value within the range allowed for the Single data type, or an error will occur.

CSng generates the same result as assigning the numeric-expression to a Single variable.

Strings that cannot be converted to an integer will result in a Type Mismatch error. Variants containing nulls will result in an Illegal Use of Null error.

CSng (function) Example

The example uses the CSng function to convert double-precision numbers to single-precision. It also demonstrates how CSng rounds numbers when converting. Note that 75.3421115 is rounded down to 75.34211, whereas 75.3421155 is rounded up to 75.34212.

```
Sub Main

Dim Msg$, NL$                       ' Declare variables.
NL$ = Chr$(13) + Chr$(10)           ' Define newline.
Msg$ = "75.3421115 rounds to" + Str$(CSng(75.3421115))
Msg$ = Msg$ + "." + NL$ + "75.3421555 rounds to"
Msg$ = Msg$ + Str$(CSng(75.3421555)) + "."
MsgBox Msg$                         ' Display results.

End Sub
```

CStr (function) — Converts to a string

rc = **CStr(***expression***)**

Converts a value to a string.

Parameter	**Description**
rc	The return value.
expression	The value to be converted. This function accepts any type of expression.

CStr (function) 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
```

CStrings (metacommand) — Treat backslash as an escape

```
'$CSTRINGS [SAVE | RESTORE]
```

Instructs the compiler to treat a backslash character within a string (\) as an escape character (based on the C language protocol).

Comments

The supported special characters are:

Special Character	**Corresponding Switch**
Newline (Linefeed)	\n
Horizontal Tab	\t
Vertical Tab	\v
Backspace	\b
Carriage Return	\r
Formfeed	\f
Backslash	\\
Apostrophe	\'
Quotation Mark	\’‘
Null Character	\0

Using the **$CStrings** metacommand makes the instruction:

```
"Hello\r World"
```

equivalent to

```
"Hello" + Chr$(13) + "World"
```

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

Octal Code `\ddd`
Hexadecimal Code `\xddd`

For both hexadecimal and octal, fewer than three characters can be used to specify the code, provided the subsequent character is not a valid hexadecimal or octal character.

The **SAVE** option causes the current **CStrings** setting to be saved before **CStrings** are enabled. The **RESTORE** option restores a previously saved setting. **SAVE** and **RESTORE** act as a stack and allow the user to change the setting for a range of the macro without impacting the rest of the macro.

CStrings (metacommand) 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
```

CurDir (function) — Return current directory

`rc% = CurDir[$][(drivename$)]`

Returns the current default path for the specified drive.

Parameter	Description
*rc***%**	The return value.
*drivename***$**	A string expression identifying the drive whose default directory path will be returned. This drive must exist, and must be within the range specified in the CONFIG.SYS file. If a Null argument (“ “) is supplied, or if no drivename is indicated, the default directory path of the default drive is returned.

Comments

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

CurDir is not case sensitive.

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

CurDir (function) Example

This example uses CurDir to return the name of the current directory.

```
NL$ = Chr$(13) + Chr$(10)          ' Define newline.
Msg$ = "The current directory is: "
Msg$ = Msg$ + NL$ + CurDir$        ' Get current directory.
MsgBox Msg$                        ' Display message.
```

CVar (function) — Converts to a variant

`rc = CVar(expression)`

Converts a value to a variant.

Parameter	**Description**
rc	The return value.
expression	The value to be converted. This function accepts any type of expression.

Comments

CVar generates the same result as if you were to assign the expression to a variant variable.

CVar (function) 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 of type: " & VarType(answer)
End Sub
```

CVDate (function) — Converts to a variant date

```
rc = CVDate(expression)
```

Converts a value to a variant date.

Parameter	**Description**
rc	The return value.
expression	The value to be converted. This function accepts any type of string or numeric expression that can be interpreted as a date.

Comments

The **CVDate** functions returns a variant of vartype 7 (date) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, is returned if *expression* is ‘‘January 1, 1900’‘.

Note: Any fractional portion of *expression* will be interpreted as a time of day, beginning with midnight.

CVDate (function) 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
```

Date (function) — Return current date

rc% = **Date**[**$**]

Returns a string containing the current system date.

Parameter	**Description**
rc%	The return value.

Comments

The **Date** function returns a 10-character string. The form of the string is *mm-dd-yyyy*, where *mm* is the month (01-12), *dd* is the day (01-31), and *yyyy* is the year (1980-2099).

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified, the return type is String. If the dollar sign is omitted, a variant of vartype 8 (String) is returned.

Date (function) 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,"ddddd")
End Sub
```

Date (statement) — Set the current system date

```
Date[$] = expression
```

Sets the current system date.

Parameter	**Description**
expression	When this function is used with the dollar sign (**$**), *expression* must evaluate to a string of one of the following forms: *mm-dd-yy* *mm-dd-yyyy* *mm / dd / yy* *mm / dd / yyyy* where *mm* indicates a month (01-12), *dd* indicates a day (01-31), and *yy* or *yyyy* indicates a year (1980-2099). If the dollar sign is omitted, *expression* can be a string containing a valid date, a variant of vartype 7 (Date), or a variant of vartype 8 (String).

Comments

If *expression* 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 chosen in the International section of the Windows Control Panel to recognize the 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.

Date (statement) Example

This example changes the system date to a date entered by the user.

```
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
```

DateSerial (function) — Returns a serial date value

`rc = DateSerial(year%, month%, day%)`

Returns the serial date equivalent to the specified date.

Parameter	**Description**
rc	The return value.
*year***%**	A number between 100 and 9999, inclusive, or a numeric expression that evaluates to a number between 100 and 9999, inclusive. Values between 0 and 99, inclusive, are interpreted as the years 1900 through 1999, inclusive. For all other years, use the four-digit number (i.e., 1800).
*month***%**	A number between 1 and 12, inclusive, or a numeric expression that evaluates to a number between 1 and 12, inclusive.
*day***%**	A number between 1 and 31, inclusive, or a numeric expression that evaluates to a number between 1 and 31, inclusive.

Comments

The **DateSerial** function returns a variant of vartype 7 (date) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Negative numbers are interpreted as dates prior to December 30, 1899.

DateSerial (function) 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
```

DateValue (function) — Returns a date value

`rc = DateValue(string-expression$)`

Returns a date value for the specified string.

Parameter	Description
rc	The return value.
string-expression$	A string representing a date from January 1, 100 through December 31, 9999, inclusive. Valid string-expression$ arguments include: 01/01/95 01/01/1995 Jan 1, 1995 January 1, 1995

Comments

DateValue interprets the order for month, day, and year according to the Short Date settings in the International section of the Microsoft Windows Control Panel. **DateValue** can also interpret dates that contain month names (either complete or abbreviated).

The **DateValue** function returns a variant of vartype 7 (date) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Negative numbers are interpreted as dates prior to December 30, 1899.

DateValue (function) 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 your date is: " & Format(nextweek,"dddddd")
      MsgBox msgtext
   Else
      MsgBox "Invalid date or format. Try again."
      Goto I
   End If
End Sub
```

Day (function) — Return the day of the month as an integer

```
rc = Day(expression)
```

Returns an integer (between 1 and 31, inclusive) representing the day of the month.

Parameter	**Description**
rc	The return value. This will be a variant of vartype 2 (Integer). If the value of expression is null, a variant of vartype 1 (null) is returned.
expresssion	A date value from -657434 (January 1, 100) through 2958465 (December 31, 9999). A value of 2, for example, represents January 1, 1900.

Comments

If expression contains a decimal point, numbers to the right of the decimal point will be interpreted as a time value. A negative expression is interpreted as a date prior to December 30, 1899.

Day (function) 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
```

Declare (statement) — Provide a forward declaration of a procedure

```
Declare Sub name [libSpecification] [(parameter[As
type][,...])]
```

-or-

```
Declare Function name [libSpecification]
[(parameter[As type])] [As functype]
```

The **Declare** statement has two uses—forward declaration of a procedure whose definition is located in the current module, and declaration of a procedure located in an external Windows DLL or external Basic module.

Parameter	**Description**
name	Indicates the Sub or Function being declared.

libSpecification	Supplies the Dynamic Link Library (DLL) that contains *name*.
	
	Use the following format if *libSpecification* is located in a Dynamic Link Library (DLL) named *libName*:
	
	`Lib libName [Alias ["]ordinal["]]`
	
	-or-
	
	`Lib libName [Alias aliasname]`
	
	The *ordinal* parameter specifies the ordinal number of the procedure within the external DLL. If *ordinal* is not supplied, the DLL function is accessed by name, causing the module to load more slowly. Use the *ordinal* parameter whenever possible.

Parameter	Description
parameter	Supplies a comma-separated list of parameter names. The data type of a particular parameter can be specified using a type declaration character or the **As** clause. Record parameters are declared using the **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by 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.

Parameter	Description
type	**Any** can be used as a type specification, and permits a call to the procedure to pass a value of any data type. When **Any** is used, type checking of the actual argument used in calls to the procedure is disabled (although other arguments not declared as type **Any** are 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** can also be used in the call. The external DLL procedure must determine the type and size of the passed value.

Comments

A forward declaration is needed only when a procedure in the current module is referenced before it is used (in which case, **Lib** and **Alias** clauses are not used).

A Sub procedure does not return a value. A Function procedure does return a value, and can be used in an expression. Function names must end with a type declaration character specifying the return value of the function.

When an empty string (“ “) is passed **ByVal** to an external procedure, the external procedure will receive a valid (non-Null) pointer to a 0 character. To send a Null pointer, declare the procedure argument as

```
ByVal As Any
```

and call the procedure with the argument:

0&

Declare (statement) Example

The example uses the Declare statement to declare a reference to an external function (GetProfileString) in the Windows Kernel DLL.

```
Declare Function GetProfileString Lib "Kernel" (ByVal Sname$, _
  ByVal Kname$, ByVal Def$, ByVal Ret$, ByVal Size%) As Integer
' Declare statement above must appear on one line in actual macro
'   source code.

Sub Main
   NL$ = Chr$(13) + Chr$(10)     ' Define newline,
   SName$ = "Intl"               ' WIN.INI section name,
   KName$ = "Country"            ' WIN.INI country code.
   Ret$ = String$(255, 0)        ' Initialize return string.

   ' Call Windows Kernel.DLL
   Success% = GetProfileString(SName$, KName$, "", Ret$, 255)
   If Success% Then              ' Evaluate results.

      Msg$ = "Your WIN.INI currently shows " + NL$ + NL$
      Msg$ = Msg$ + "'" + KName$ + "' = "
      Msg$ = Msg$ + RTrim$(Ret$)
   Else
      Msg$ = "There is no country code in your WIN.INI file."
   End If
   MsgBox Msg$                   ' Display message.

End Sub
```

Deftype (statements) — Declare a default data type

```
DefCur varTypeLetters
DefInt varTypeLetters
DefLng varTypeLetters
DefSng varTypeLetters
DefDbl varTypeLetters
DefStr varTypeLetters
DefVar varTypeLetters
```

Specifies the default data type of any variable that begins with the letter or letters specified in *varTypeLetters.*

Parameter	**Description**
varType Letters	A comma-separated list or range of letters. A letter range should be specified using the syntax: *beginletter-endletter* *beginletter* and *endletter* specify the range of variables for which a default data type will be assigned.

Comments

The parameter *varTypeLetters* is not case sensitive.

The **Def*type*** statement only affects the module in which it is specified. It must precede any variable definition within the module.

The letter range a-z is considered a special case. It indicates all alpha characters, including any international characters from the extended portion of the ANSI character set (128-255). Once the range a-z has been specified, you cannot redefine any subranges using any of the **Def*type*** statements. Similarly, once a range has been specified, an error will occur if you redefine any of the letters in that range using the **Def*type*** statement. You can always explicitly define the data type of any variable, defined or not, using the **Dim** or **Global** statement or a type declaration character.

Note: A type-declaration character (**%**, **&**, **!**, **#**, or **$**) always takes precedence over a **Def*type*** statement. **Def*type*** statements do not affect elements of user-defined types.

Deftype (statements) Example

This example uses the Deftype statements to set the default data types of variables in two ranges. Variables beginning with the letters A-K will be Integer; variables beginning with L-Z will be String.

```
DefInt A-K

DefStr L-Z


Sub Main


NL = Chr$(13) + Chr$(10)        ' Define newline.

IntVal = 2.3455


Msg = "Notice that none of the string variables in this demo have "

Msg = Msg + "the $ type-declaration character appended to their "

Msg = Msg + "names." + NL + NL + "The DefStr statement defines "

Msg = Msg + "the default for variables starting with letters 'L' "

Msg = Msg + "through 'Z' to be of the string data type."

MsgBox Msg


Msg = "Notice that the variable 'IntVal' has been defined as an "

Msg = Msg + "integer. Even though it is assigned a real value, it "

Msg = Msg + "behaves as an integer, i.e., IntVal = "


Msg = Msg +  LTrim$(Str$(IntVal)) + " instead of 2.3455."

MsgBox Msg                      ' Display message.

End Sub
```

Dialog (function) — Displays a dialog box and returns user button selection

rc = **Dialog(***recordName***)**

Displays a dialog box and returns a value corresponding to the button the user selected.

Parameter	**Description**
rc	The return value. The **Dialog** function returns -1 if the user selects the OK button; 0 if the user selects the Cancel button; and a number greater than 0 if the user selects a command button. (1 corresponds to the first command button, 2 to the second, and so forth.)
recordName	The record containing information about the controls provided in the dialog box.

Comments

The dialog box *recordName* record must have been declared using the **Dim** statement.

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

Dialog (function) 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, "E! Basic 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
```

Dialog (statement) — Display a dialog box

Dialog *recordName*

Displays a dialog box. The data for the controls of the dialog box come from the dialog box record *recordName*.

Parameter	**Description**
recordName	The record containing information about the controls provided in the dialog box.

Comments

You must first declare the dialog box *dialogName* using the **Dim** statement. If the user exits the dialog box by pressing the Cancel button, a run-time error will be triggered. This error can be trapped using **On Error**.

Dialog (statement) 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, "E! Basic 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
```

Dim (statement) — Declare a variable for use in an EXTRA! Basic program

```
Dim [Shared] variableName [As [New ]type] _
[,variableName [As [New ]type]] ...
```

Declares variables for use in an EXTRA! Basic program.

Parameter	**Description**
variable Name	Indicates the name of the variable you want to declare.
type	The available data types are: number, string, and record.

Comments

If the **As** clause is not used, specify the variable type 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 for the same variable).

The Shared keyword is included for backward compatibility with older versions of EXTRA! Basic. It is not allowed in **Dim** statements within a procedure.

EXTRA! Basic automatically declares a variable on first use without explicit declaration with the **Dim** statement. If an unDimmed variable is supplied with a type suffix, it is automatically declared to be a local variable of that type. If no type suffix is specified, the variable is automatically declared to be a local variable of type Double.

It is considered good programming practice to actually declare all variables. It is also recommended that you place all procedure-level **Dim** statements at the beginning of the procedure.

Dim (statement) 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
```

Dir (function) — Return a filename

```
rc% = Dir[$] [(filespec$ [, attrib%])]
```

Returns the first filename that matches the specified pattern.

Parameter	**Description**
*rc***%**	The return value.
*filespec***$**	A string expression identifying a path or filename. This argument may include a drive specification. It may also include the wildcard characters ‘‘**?**’‘ and ‘‘*****’‘.
*attrib***%**	An integer expression specifying the filenames that will make up the list to be evaluated. Valid *attrib***%** values are: 0 returns normal files 2 returns normal and hidden files 4 returns normal and system files 8 returns the volume label 16 returns directory names These values can be combined by adding. For example, to list hidden and system files, in addition to normal files, set *attrib***%** to 6 (2 + 4 = 6). The default value for *attrib***%** is 0.

Comments

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified, the return type is String. If it is omitted, the function will return a variant of vartype 8 (String).

Dir returns the first filename that matches the *filespec***$** parameter. To retrieve additional filenames that match *filespec***$**, call the **Dir** function again, omitting the *filespec***$** parameter. If no file is found, an empty string (‘‘ ‘‘) is returned; subsequent calls to the **Dir** function must include the *filespec***$** parameter.

Dir (function) Example

This example uses Dir to determine all the files based on the extension selected by the user. If no extension is chosen, the default is all files in the current directory are listed.

```
NL$ = Chr$(13) + Chr$(10): TB$ = Chr$(9)  ' Define newline, tab.
Msg$ = "Enter a file specification." + NL$ + "(example = C:\*.sys)"

Filespec$ = InputBox$(Msg$)               ' Get filename.

   Match$ = Dir$(Filespec$)               ' Find first match.
   If Len(Match$) Then                    ' If match found,

      Do
         MatchNum% = MatchNum% + 1        ' increment match count.
         Msg2$ = Msg2$ + NL$ + TB$ + Match$  ' Make a list.
         If Len(Msg2$) >= 180 Then        ' Handle list too long
            Match$ = "And more - - -"     ' for MsgBox.
            Msg2$ = Msg2$ + NL$ + TB$ + Match$
            Exit Do                       ' Quit Do Loop.
         End If
         Match$ = Dir$                    ' Find next match.

      Loop Until Len(Match$) = 0          ' Continue.
      If MatchNum% > 1 Then               ' More than one file.
         Msg1$ = "The files that match your file specification are:"

      Else                                ' One match only.
         Msg1$ = "The file that matches your file specification is:"
      End If
      Msg$ = Msg1$ + Msg2$                ' Put message together.
```

```
Else
   Msg$ = "The file you specified could not be found."
End If
MsgBox Msg$                            ' Display message
```

DlgControlID (function) — Translate a string ID into a numeric ID

rc = **DlgControlID(***Id***$)**

Takes a string dialog box control ID and returns the numeric value for that control.

Parameter	Description
rc	The return value.
*Id***$**	The string ID associated with the control.

Comments

Numeric IDs correspond to the position of a control within the dialog box definition. The first control is assigned ID 0 (zero), the second is assigned ID 1, and so forth.

DlgControlID (function) 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$(), .ListBox1
            Text  8, 108, 61, 9, "List Files of &Type:"
            DropListBox  7, 120, 78, 30, filetypes, .DropListBox1
            Text  98, 7, 43, 10, "&Directories:"
            Text  98, 20, 46, 8, "c:\\windows"
            ListBox  99, 34, 66, 66, "", .ListBox2
            Text  98, 108, 44, 8, "Dri&ves:"
            DropListBox  98, 120, 68, 12, "", .DropListBox2
            OKButton  177, 6, 50, 14
            CancelButton  177, 24, 50, 14
            PushButton 177, 42, 50, 14, "&Help"
            '$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$, action, suppvalue)
   Select Case action
     Case 1
       str1$="*.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
```

DlgEnable (function) — Determine whether or not a dialog is enabled

rc = **DlgEnable(***Id***)**

Determines whether or not a dialog box is enabled.

Parameter	**Description**
rc	The return value. The DlgEnable function returns -1 if the dialog control identified by Id is enabled, or 0 if it is enabled.
id	The dialog box control. This must be a numeric argument. If you want to convert a string argument to the correct id, use DlgControlID.

DlgEnable (function) Example

This example displays a dialog box with one check box, 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 check box 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, "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
```

DlgEnable (statement) — Enable or disable a dialog box

`DlgEnable` `Id [, mode]`

Enables the dialog control identified by id if mode is 1, and disables it if mode is 0.

Parameter	Description
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on it's string identifier.
mode	A numeric expression equaling 1 or 0 as shown in the following chart:

Value	Action
1	Enables the dialog control indicated by *id*.
0	Disables the dialog control indicated by *id*.

If *mode* is omitted, the **DlgEnable** statement toggles the state of the dialog control identified by the *id* parameter.

DlgEnable (statement) Example

This example displays a dialog box with two check boxes, one labeled Either, the other labeled Or. If the user clicks on Either, the Or option is filled with gray shading. Likewise, if Or is selected, Either has gray shading. This example uses the DlgEnable statement to toggle the state of the buttons.

```
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  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$, action, suppvalue)
   Select Case action
      Case 2                       'button or control value changed
       If DlgControlId(identifier$) = 2 Then
         DlgEnable 3
       Else
         DlgEnable 2
         End If
   End Select
End Function
```

DlgFocus (function) — Return id of control with focus

`rc = DlgFocus$()`

Returns the id of the dialog box control that currently has input focus. A control has focus when it is active and responds to keyboard input.

Parameter	Description
rc	The return value.

DlgFocus (function) Example

This example displays a dialog box with a check box, 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 check box, the focus goes to the OK button.

```
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, suppvalue)
   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
```

DlgFocus (statement) — Set focus to specified control

```
DlgFocus id
```

Sets the focus to the dialog box control identified by the *id* parameter.

Parameter	**Description**
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.

DlgFocus (statement) Example

This example displays a dialog box with a check box, 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 check box, the focus goes to the OK button.

```
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, suppvalue)
```

```
      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
```

DlgListBoxArray (function) — Return the number of elements in a list or combo box

rc = **DlgListBoxArray (***id* [, *array***$**]**)**

Returns the number of elements in the specified list box or combo box. The DlgListBoxArray function fills the array with the list box or combo box entries.

Parameter	**Description**
rc	The return value.
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
*array***$**	A one-dimensional array of dynamic strings.

Comments

Provided *array***$** is dynamic, its size will be altered to accommodate the number of strings in the control. If *array***$** is not dynamic, and is not large enough to accommodate all of the strings in the control, an error occurs. If *array***$** is omitted, the function returns the number of entries in the control.

DlgListBoxArray (function) Example

This example displays a dialog box with a check box, labeled ‘‘Display List’‘, and an empty list box. If the user clicks the check box, 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, suppvalue)

Sub Main

   Dim button as integer

   Dim identifier$

   Dim action as Integer

   Dim suppvalue as Integer

   Begin Dialog newdlg 186, 92, "DlgListBoxArray Example", _
```

```
      .FileDlgFunction
      '$CStrings Save
      OKButton  130, 6, 50, 14
      CancelButton  130, 23, 50, 14
      ListBox  19, 26, 74, 59, "", .ListBox1
      CheckBox  12, 4, 86, 13, "Display List", .CheckBox1
      '$CStrings Restore
   End Dialog
   Dim dlg As newdlg
   button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
Dim myarray$(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 Select
End Function
```

DlgListBoxArray (statement) — Fill the specified list or combo box with array elements

`DlgListBoxArray` *`id`*`,` *`array`***`$`**

Fills the list box or combo box identified by the *id* parameter with the strings from the array identified by the *array***$** parameter.

Parameter	**Description**
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
*array***$**	A one-dimensional array of dynamic strings.

DlgListBoxArray (statement) 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
      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$, action, suppvalue)
   Select Case action
     Case 1
       str1$="*.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
```

DlgSetPicture (statement) — Change the picture in a picture control

DlgSetPicture *id*, *filename***$**, *type*

Changes the picture currently displayed in a picture control to a different picture.

Parameter	Description
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
*filename***$**	The name of a bitmap file (.BMP) containing the new picture. (If *type* is 3, this parameter is ignored.)
type	Indicated the type of the bitmap as shown in the chart below:

Value	Action
0	Use bitmap contained in *filename***$**
3	Copy bitmap from Windows Clipboard

Comments

If the picture is not available (either the file filename**$** does not exist or it does not contain a bitmap, or there is no bitmap in the Clipboard), the picture control will display the picture frame and the text ‘‘(missing picture).’‘ To trigger a runtime error instead, add 16 to the type value (that is, a type value of either 16 or 19).

DlgSetPicture (statement) Example

This example displays a picture in a dialog box and changes the picture if the user clicks the check box 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 Select
End Function
```

DlgText (function) — Return the text associated with a dialog box control

```
rc = DlgText$(id)
```

Returns the text associated with the control identified by the *id* parameter.

Parameter	**Description**
rc	The return value.
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.

Comments

If the control is a text box or a combo box, **DlgText$** returns the text that appears in the text box. If the control is a list box, the function returns the item that is currently selected. If the control is a command button, option button, option group, or check box, the function returns its label.

DlgText (function) Example

This example displays a dialog box similar to File Open. It uses DlgText to determine which 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$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
       '$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$, action, suppvalue)
   Select Case action
     Case 1
       str1$="*.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
```

DlgText (statement) — Change the text associated with a control

DlgText$ *id*, *text***$**

Changes the text associated with the control identified by the *id* parameter.

Parameter	Description
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
*text***$**	A string or string expression.

Comments

If the control identified by the *id* parameter is a text box or combo box, **DlgText** sets the text that apprears to *text***$**. If the control is a command button, option button, option group, or check box, the **DlgText** statement sets *text***$** to the control's label.

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

DlgText (statement) 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, 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
      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$, action, suppvalue)
   Select Case action
     Case 1
       str1$="*.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
```

DlgValue (function) — Return value of a control state

```
rc = DlgValue(id)
```

Returns the numeric value associated with the current state of the specified control.

Parameter	**Description**
rc	The return value.
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.

Comments

Refer to the following chart:

For control type	**A DlgValue return value of**	**Means**
Checkbox	1	Checkbox is selected
	0	Checkbox is cleared
	-1	Checkbox is filled with gray shading
Option group	0	First option button is selected
	1	Second option button is selected
	2...	Third option button is selected and so forth
List box Combo box	0	First item is selected
	1	Second item is selected
	2...	Third item is selected and so forth

If *id* names a text box, command button, or option button control, an error occurs.

DlgValue (function) Example

This example changes the picture in the dialog box if the check box 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
```

DlgValue (statement) — Change the value of a control

```
DlgValue id, value%
```

Changes the state value associated with the specified control.

Parameter	Description
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
value	The integer value indicating the state to which you want to set the control. Refer to the chart below.

Comments

For control type	Set *value*% to	To
Checkbox	1	Select the checkbox
	0	Clear the checkbox
	-1	Fill the checkbox with gray shading
Option group	0	Select the first option button
	1	Select the second option button
	2...	Select the third option button and so forth
List box Combo box	0	Select the first item
	1	Select the second item
	2...	Select the third item and so forth

If *id* names a text box, command button, or option button control, an error occurs.

DlgValue (statement) 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

            Else

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

DlgVisible (function) — Determine whether a control is visible or hidden

rc = **DlgVisible(***id***)**

Returns -1 if the specified dialog control is visible and 0 if it is hidden.

Parameter	**Description**
rc	The return value.
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.

DlgVisible (function) Example

This example displays Option 2 in the Group box if the user clicks the check box 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
```

DlgVisible (statement) — Change a control from visible to hidden or hidden to visible

`DlgVisible` `id[,mode]`

Changes the specified control to be either visible or hidden.

Parameter	**Description**
id	The dialog box control. This must be a numeric argument. Use DlgControlID to get the numeric id based on its string identifier.
mode	A value or numeric expression that equals 1 if you want the control to be visible or 0 if you want the control to be hidden. Omit *mode* to toggle the control to hidden if it is currently visible or visible if it is currently hidden.

DlgVisible (statement) Example

This example displays Option 2 in the Group box if the user clicks the check box. 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
```

DoEvents (statement) — Yield to Windows

`DoEvents`

Yields execution to Windows for processing operating system events.

Comments

DoEvents does not return until Windows has finished processing all events in the queue and all keys sent by SendKeys statement.

DoEvents should not be used if other tasks can interact with the running program in unforeseen ways. Since EXTRA! Basic yields control to the operating system at regular intervals, DoEvents should only be used to force EXTRA! Basic to allow other applications to run at a known point in the program.

DoEvents (statement) Example

This example activates the Windows Terminal application, dials the number and then allows the operating system to process events.

```
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
```

Do While...Loop/Do Until...Loop (statements) — Repetitive action control

```
Do [{While | Until}condition]
   [statementblock]
   [Exit Do]
   [statementblock]
Loop
```

-or-

```
Do
   [statementblock]
   [Exit Do]
   [statementblock]
Loop [{While | Until} condition]
```

Tests for the specified condition after executing a block of statements and repeats the loop if the condition remains true.

Parameter	**Description**
condition	Any Boolean expression that EXTRA! Basic can determine to be true (non zero) or false (zero).
statement block	Contains the program lines that are repeated as long as a **While** condition is true or until an **Until** condition is false.

Comments

The first version of the syntax above may not execute the loop if the condition is false. The second version of the syntax will always execute the loop at least once.

When an **Exit Do** statement is executed, control is transferred to the statement that follows the **Loop** statement. When used within a nested loop, an **Exit Do** statement moves control out of the immediate loop.

Do While...Loop/Do Until...Loop (statements) Example

The example creates an infinite Do...Loop that can be exited only if the user enters a number in a range.

```
Sub Main

Dim Reply$                                       ' Declare variable.

Do
```

```
   Reply$ = InputBox$("Enter a number greater than 1 / less than 9.")
   If (Val(Reply) > 1) And (Val(Reply) < 9) Then   ' Check range.
      Exit Do                                       ' Exit Do Loop.
   End If
Loop
End Sub
```

DropComboBox (statement) — Create combination drop-down list box and text box

```
DropComboBox x, y, dx, dy, text$, .field
```

-or-

```
DropComboBox x, y, dx, dy, stringarray$(), .field
```

Creates a combination static text box (or edit box) and drop-down list box.

Parameter	**Description**
x, y	Specifies the position of the drop-down combo box relative to the upper left corner of the client area. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the drop-down combo box is centered horizontally within the client area. If *y* is omitted, the box is centered vertically within the client area.
dx, dy	The sum of the width of the drop-down combo box and the *text***$** parameter. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text***$** parameter (including blanks and punctuation) by 4 and add 12. *dy* is the height of the *text***$** parameter. 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 drop-down combo box and the accompanying text will move downward within the dialog box.

Parameter	Description
text$	A string containing the selections for the drop-down combo box. This string must be defined, using a **Dim** statement, before the **Begin Dialog** statement is executed. The arguments in the *text*$ string are entered as follows: *dimname* = ‘‘*listchoice*’‘ + **Chr$**(9) = ‘‘*listchoice*’‘ + **Chr$**(9) + ‘‘*listchoice*’‘ + **Chr$**(9)...
stringarray	An array of dynamic strings.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the drop-down combo box. The user's selection is recorded in the field designated by the *.field* argument when the OK button (or any button other than Cancel) is selected. The *.field* argument is also used by the dialog statements that act on this control.

Comments

The **DropComboBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

DropComboBox (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, "E! Basic 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
```

DropListBox (statement) — Create a drop-down list box

```
        DropListBox x, y, dx, dy, text$, .field
```

-or-

```
        DropListBox x, y, dx, dy, stringarray$(), .field
```

Creates a drop-down list of choices from which users can select.

Parameter	Description
x, y	*x* and *y* specify the position of the drop-down list box relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the drop-down list box is centered horizontally within the client area. If *y* is omitted, the drop-down list box is centered vertically within the client area.
dx, dy	The combined width of the drop-down list box and the *text*$ field. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text*$ parameter (including blanks and punctuation) by 4 and add 12. The *dy* argument is the height of the *text*$ parameter. 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 drop-down list box and the accompanying text will move downward within the dialog box.

Parameter	Description
*text***$**	A string containing the selections for the drop-down list box. This string must be defined using a **Dim** statement, before the **Begin Dialog** statement is executed.
stringarray	An array of dynamic strings.
.field	The *.field* parameter is the name of the dialog-record field that will hold the drop-down list box selection. When the user selects OK (or selects the customized button created using the Button statement), a number representing the selection's position in the *text***$** string is recorded in the *.field* field. *listchoice* numbers begin at 0. If no item is selected, *.field* is set to -1.

Comments

The **DropListBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

Note: Unlike a **ListBox**, a **DropListBox** may overlap other controls or fall outside the dialog box dimensions when dropped down.

DropListBox (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, "E! Basic 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
```

Environ (function) — Return an environment string

`rc% = Environ[$](environment-string$)`

-or-

`rc% = Environ$(n%)`

Returns a string from the operating system's environment table.

Parameter	**Description**
*rc***%**	The return value. The returned value will be in the form: `Keyword = value`.
*environment-string***$**	The name of a keyword in the operating system environment variable table. The value associated with the keyword is returned. If this argument is not entered in uppercase, **Environ$** will return a Null string.
*n***%**	*n***%** represents the *n*th string in the operating system environment variable table. *n***%** may be any numeric expression, however it will be rounded to a whole number by **Environ$** before being evaluated.

Comments

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified the return type is String. If it is omitted, the **Environ** function will return a variant of vartype 8 (String).

A Null string will be returned if the specified argument cannot be found.

Environ (function) Example

This example uses Environ to supply the entry number and length of the PATH statement from the environment-string table.

```
Indx% = 1                                    ' Initialize Index to 1.
Do
   EnvString$ = Environ$(Indx%)              ' Get environment variable.
   If Left$(EnvString$, 5) = "PATH=" Then    ' Check if PATH.
```

```
            PathLen% = Len(Environ$("PATH"))    ' Get length.
            Msg$ = "Your PATH statement is entry number "
            Msg$ = Msg$ + LTrim$(Str$(Indx%))
            Msg$ = Msg$ + " in the environment-string table. "
            Msg$ = Msg$ + "Your PATH statement uses "
            Msg$ = Msg$ + LTrim$(Str$(PathLen%)) + " characters."
            Exit Do
        Else
            Indx% = Indx% + 1                   ' Not PATH entry,
                                                ' so increment.
        End If
    Loop Until EnvString$ = ""
    If PathLen% Then
        Print Msg$                              ' Display message.
    Else
        MsgBox "No PATH statement exists."
        Print Msg$
    End If
```

Eof (function) — Create end-of-file condition

`rc% = Eof (filenumber%)`

Returns a value indicating whether the end of a file has been reached.

Parameter	**Description**
rc%	The return value.
filenumber%	An integer or integer expression corresponding to a file number assigned in a previous **Open** statement.

Comments

The **Eof** Function returns a (-1) if the end-of-file condition is true for the specified file.

Eof (function) Example

This example uses the Eof function to detect the end of a sequential file.

```
Dim FileNum, Handle, Mode$, Msg$       ' Declare variables.
FileNum = FreeFile                     ' Get available file number.
Open "TESTFILE" For Append As FileNum  ' Create sample data file.
Handle = FileAttr(FileNum,2)           ' Get file handle.
Select Case FileAttr(FileNum,1)        ' Determine mode.
   Case 1 : Mode = "Input"
   Case 2 : Mode = "Output"
   Case 8 : Mode = "Append"
End Select
Close FileNum                          ' Close file.

Msg$ = "The file using DOS file handle " + STR$(Handle)
Msg$  = Msg$ + " was opened as an " + Mode + " file."
MsgBox Msg$                            ' Display message.
Kill "TESTFILE"                        ' Delete zero-length file.
```

Erase (statement) — Reinitialize a fixed array or free a dynamic array

```
Erase array1[, array2]...
```

Reinitializes the contents of a fixed array and deallocates or frees the storage associated with a dynamic array.

Parameter	**Description**
array	The name of the array (or arrays) to be erased. Refer to the chart below for details about how Erase affects fixed array elements.

Comments

A fixed array of	**Produces this Erase effect**
numeric elements	Each element is set to zero
variable length strings	Each string is set to a length of zero
fixed length strings	Each string is zero-filled
variant elements	Each element is set to Empty
user-defined element types	Members of each element are cleared as if the members were array elements (e.g., numeric elements are set to zero)
object elements	Each elements is set to the special value Nothing

Dynamic array consideration: After using the **Erase** function to deallocate storage space for a dynamic array, your macro must redeclare the array with the **ReDim** statement.

Erase (statement) Example

This example prompts for a list of item numbers to put into an array and clears array if the user wants to start over.

```
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
```

Erl (function) — Return line number location of an error

```
rc% = Erl
```

Returns the line number where an error occurred.

Parameter	**Description**
*rc***%**	The return value.

Comments

Use **Erl** in an error-handling routine to investigate trapped errors and outline corrective action. Using the **Resume** or **On Error** statements will reset the **Erl** value to zero (0).

Note: An **On Error GoTo** error handler that calls another procedure may inadvertantly reset the Erl value to zero. To make sure that the **Erl** value does not get reset, you should assign it to a variable.

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

If your procedure does not have line numbers, or if an error occurs before your procedure is executed, **Erl** returns a zero.

Erl (function) 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
```

Err (function) — Return error code

```
rc% = Err
```

Returns the run-time error code for the last error.

Parameter	**Description**
*rc***%**	The return value.

Comments

Use **Err** in an error-handling routine to return an integer run-time error code identifying an error. Using the **Resume** or On Error statements will reset the **Err** value to zero (0).

Note: An On Error GoTo error handler that calls another procedure may inadvertantly reset the **Err** value to zero. To make sure that the **Err** value does not get reset, you should assign it to a variable.

The value returned by the **Err** function can be set directly using the **Err** statement. The value of **Err** can be set indirectly through the **Error** statement.

Err (function) 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
```

Err (statement) — Transfer application error information

`Err` `=` *`n%`*

Sends application-specific error information from one procedure to another.

Parameter	**Description**
*n***%**	Must be a 0 (indicating that no run-time error has been trapped) or an integer expression indicating a run-time error code (with a value between 1 and 32,767 inclusive).

Comments

Set **Err** to a nonzero integer value corresponding to an application-specific error code. To determine which codes are not used by EXTRA! Basic (and therefore available for use by your application) use the **Error$** function and / or the **Error** statement (or work backwards from error code 32,767).

Using the **Resume** or **On Error** statements will reset the **Err** value to zero (0).

Err (statement) Example

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.

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

Error (statement) — Simulate an error condition

```
Error errorcode%
```

Simulates the occurrence of an EXTRA! Basic or user-defined error.

Parameter	**Description**
*errorcode***%**	Represents a specific error code and must be an integer between 1 and 32,767. If an *errorcode***%** is an EXTRA! Basic error code, the **Error** statement will simulate an occurrence of that error and set **Err** to the value of *errorcode***%**.

Comments

User-defined error codes should use values greater than those used for standard EXTRA! Basic error codes. To determine which codes are not used by EXTRA! Basic (and therefore available for use by your application) use the **Error$** function and / or the **Error** statement (or work backwards from error code 32,767).

If an **Error** statement is executed, but no error-handling routine is enabled, EXTRA! Basic produces an error message and halts program execution. If an **Error** statement specifies an error code not used by EXTRA! Basic, the message

```
User-defined error
```

is displayed.

Error (statement) Example

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.

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

Error (function) — Return error message

*rc***$** = **Error**[**$**] [**(***errorcode***%)**]

Returns the error message that corresponds to the specified error code.

Parameter	**Description**
*rc***%**	The return value.
*errorcode***%**	Represents a specific error code and must be an integer between 1 and 32,767.

Comments

If this argument is omitted, EXTRA! Basic returns the error message for the run-time error that occurred most recently.

If no error message matches the error code, “ “ is returned.

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified, the return type is String. If it is omitted, **Error** will return a variant of vartype 8 (String).

Error (function) 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
```

Exit (statement) — Transfer control to the next statement

```
Exit {Do | For | Function | Sub}
```

Transfers control of the program execution to the next statement block.

Statement	Description
Exit Do	Provides an exit from a **Do...Loop** statement by transferring control to the statement following the Loop statement. If **Exit Do** is used within a nested **Do...Loop** statement, control returns to the loop that is one nested level above the loop in which it occurs.
Exit For	Provides an exit from a **For...Next** statement by transferring control to the statement following the **Next** statement. If **Exit For** is used within a nested **For...Next** statement, control returns to the loop that is one nested level above the loop in which it occurs.
Exit Function	Exits immediately. Program execution continues with the statement following the **Exit Function** call.
Exit Sub	Exits immediately. Program execution continues with the statement following the **Exit Sub** procedure call.

Exit (statement) Example

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.

```
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
```

Exp (function) — Return the antilogarithm

`rc% = Exp(numeric-expression)`

Returns the value *e* (the base of natural logarithms) raised to the *numeric-expression* power.

Parameter	**Description**
rc%	The return value.
numeric-expression	Any numeric data type. The **Exp** function will round the resulting value, if necessary, before converting it.

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision Integer
#	Double	Double-precision floating point

Comments

An overflow error will occur if numeric-expression exceeds 709.782712893. The value of the constant *e* is approximately 2.718282.

Exp (function) 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
```

FileAttr (function) — Return a file attribute

```
rc% = FileAttr(filenumber%, attribute%)
```

Returns information about an open file. Depending on the attribute chosen, this information is either the file mode or the operating system handle.

Parameter	**Description**
*rc***%**	The return value.
*filenumber***%**	An integer or integer expression corresponding to a file number assigned in a previous **Open** statement.
*attribute***%**	An integer or integer expression that evaluates to either one (1) or two (2). Set *attribute***%** to 1 to return one of the following file modes:

Return	**Mode Description**
1	Input
2	Output
8	Append

Comments

If *attribute***%** is set to 2, **FileAttr** returns the operating system handle for the file.

FileAttr (function) Example

The example uses FileAttr to report the file handle and mode of an open file.

```
Dim FileNum, Handle, Mode$, Msg$       ' Declare variables.
FileNum = FreeFile                     ' Get available file number.
Open "TESTFILE" For Append As FileNum  ' Create sample data file.
Handle = FileAttr(FileNum,2)           ' Get file handle.
Select Case FileAttr(FileNum,1)        ' Determine mode.
   Case 1 : Mode = "Input"
   Case 2 : Mode = "Output"
   Case 8 : Mode = "Append"
End Select
Close FileNum                          ' Close file.

Msg$ = "The file using DOS file handle " + STR$(Handle)
Msg$  = Msg$ + " was opened as an " + Mode + " file."
MsgBox Msg$                            ' Display message.
Kill "TESTFILE"                        ' Delete zero-length file.
```

FileCopy (statement) — Copies a source file to a new or existing file

FileCopy *sourcefile***$**, *destfile***$**

Copies the contents of a file to a new or existing file. This command will overwrite existing files.

Parameter	**Description**
*sourcefile***$**	A string or string expression pointing to the file to be copied. *Sourcefile***$** cannot contain wildcard characters. Drive and path information may be included.
*destfile***$**	A string or string expression pointing to the file to which *sourcefile* is to be copied. *Destfile***$** cannot contain wildcard characters. Drive and path information may be included.

Comments

Sourcefile cannot be copied if it is opened by EXTRA! Basic for anything other than Read-only access.

FileCopy (statement) 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
```

FileDateTime (function) — Return date/time a file was modified last

rc = **FileDateTime(***filename***$)**

Returns a string indicating the date and time the specified file was created or last saved.

Parameter	**Description**
rc	The return value.
filename**$**	A string or string expression pointing to the file you want to check. Filename**$** cannot contain wildcard characters. Drive and path information may be included.

FileDateTime (function) Example

This example writes data to a file if it hasn't been saved within the last 2 minutes.

```
Sub main
   Dim tempfile
   Dim filetime, curtime
   Dim msgtext
   Dim acctno(100) as Single
   Dim x, I
   tempfile="C:\TEMP001"
   Open tempfile For Output As #1
   filetime=FileDateTime(tempfile)
   x=1
   I=1
   acctno(x)=0
   Do
      curtime=Time
      acctno(x)=InputBox("Enter an account number (99 to end):")
      If acctno(x)=99 then
```

```
            For I=1 to x-1
               Write #1, acctno(I)
            Next I
            Exit Do
         ElseIf (Minute(filetime)+2)<=Minute(curtime) then
            For I=I to x
               Write #1, acctno(I)
            Next I
         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
```

FileLen (function) — Return the length of a file (in bytes)

`rc = FileLen(filename$)`

Returns a Long integer indicating the length (in bytes) of the specified file.

Parameter	**Description**
rc	The return value.
*filename***$**	A string or string expression pointing to the file you want to check. *Filename***$** cannot contain wildcard characters. Drive and path information may be included.

Comments

If *filename***$** is open when this function is called, **FileLen** returns the length of the file before it was opened.

FileLen (function) 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
```

Fix (function) — Return integer portion of a number

```
rc% = Fix (numeric-expression)
```

Returns the integer portion of a numeric expression.

Parameter	**Description**
*rc***%**	The return value. This return value type will be the same type as *numeric-expression*. If *numeric-expression* is a variant, however, of vartype 8 (String) that can be converted to a number, or vartype 7 (Date), *rc* will be a variant of vartype 5 (Double).
numeric-expression	Any numeric data type.

Comments

For both positive and negative *numeric-expression* parameters, **Fix** removes the fractional portion of the expression and returns the integer portion only. For example,

```
Fix (6.2)
```

returns 6; and

```
Fix (-6.2)
```

returns -6.

Important: For negative numeric expressions, **Int** returns the first negative integer less than or equal to *numeric-expression*, and **Fix** returns the first negative integer *greater* than or equal to *numeric-expression*. For example, **Int** converts -5.6 to -6 and **Fix** converts -5.6 to -5.

If the numeric expression evaluates to a Null, **Fix** returns a Null.

Fix (function) Example

The example illustrates the difference between Int and Fix.

```
Dim Msg$, NL$, TB$                ' Declare variables.
NL$ = Chr$(13) + Chr$(10):
```

```
TB$ = Chr$(9)                         ' Define newline, tab.
Msg$ = "Int(-99.8) returns" + TB$ + STR$(Int(-99.8))
Msg$ = Msg$ + TB$ + "Fix(-99.8) returns" + TB$ + STR$(Fix(-99.8))
Msg$ = Msg$ + NL$ + "Int(-99.2) returns" + TB$ + STR$(Int(-99.2))
Msg$ = Msg$ + TB$ + "Fix(-99.2) returns" + TB$ + STR$(Fix(-99.2))
MsgBox Msg                            ' Display message.
```

For ... Next (statement) — Repeat a group of instructions *x* number of times

```
For counter = start To end [Step increment]
   [ statementblock ]
   [ Exit For ]
   [ statementblock ]
Next [ counter ]
```

Repeats the statement block the specified number of times.

Parameter	**Description**
counter	A numeric variable used as the loop counter. *counter* cannot be an element of an array or record.
start	The initial value of *counter*.
end	The final value of *counter*.
increment	The amount by which the counter is changed each time through the loop. (The default is 1.)
statement block	The group of EXTRA! 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. Provided the signs of *start*, *end*, and *increment* are the same, and *end* does not equal *start*, the **For...Next** loop is entered. If not, the loop is skipped entirely.

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 *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	**Produces this Loop Execution**
Positive	If *counter* is less than or equal to *end*, the **Step** value is added to *counter*. Control returns to the statement following the **For** statement and the process is repeated. If *counter* is greater than *end*, the loop is exited and execution resumes with the statement following the **Next** statement.
Negative	The loop repeats until *counter* is less than *end*.
Zero	The loop repeats indefinitely.

For...Next loops can be nested within one another. Each nested loop, however, should be assigned a unique *counter* variable name. The **Exit For** statement can be used as an alternative exit from a **For...Next** loop.

If the *counter* variable is omitted, the **Next** statement will match the most recent **For** statement. If a **Next** statement occurs prior to its corresponding **For** statement, EXTRA! Basic will return an error message.

Multiple, consecutive **Next** statements can be merged together. If so, the innermost counter must appear first and the outermost counter last.

For ... Next (statement) 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. In this example, the For...Next statement is used to determine the integers between x and 1, multiply those numbers by x, and produce x!.

```
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
```

Format (function) — Convert the value of *expression* to a *fmt*-defined string

```
rc = Format[$](expression[, fmt])
```

Converts the value of *expression* to a string based on the *fmt* specified.

Parameter	Description
rc	The return value.
expression	The value to be formatted. *Expression* can be a number, variant, or string.
fmt	Any string expression. Refer to the charts below for details.

Formatting Numbers

A numeric value can be formatted as either a number, a date, or a time. If the *fmt* argument is omitted or Null, the number will be converted to a string without any special formatting.

An *expression* of type	Is displayed as
General	The number without a thousands separator.
Fixed	The number with at least one digit to the left and at least two digits to the right of the decimal separator.
Standard	The number with a thousands separator and two digits to the right of the decimal separator.
Scientific	The number using standard scientific notation.
Currency	The number using a currency symbol as selected in the International section of the Control Panel. The thousands separator is displayed as are two degits to the right of the decimal separator. Negative values are enclosed in parentheses.

An *expression* of type	**Is displayed as**
Percent	The number multiplied by 100 and displayed with a percent sign appended to the right. Also displays two digits to the right of the decimal separator.
True / False	0 for false; any other number for true.
Yes / No	0 for no; any other number for true.
On / Off	0 for off; any other number for on.

User-defined numeric formatting rules:

A simple numeric *fmt* consists of digit placeholders and, an optional decimal separator. The valid digit placeholders are 0 and #. 0 forces the corresponding digit to appear in the output; while a # causes the digit to appear in the output if it is significant (in the middle of the number or non-zero). For example:

This *expression*	**Formatted with this *fmt***	**Produces**
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 between digit placeholders causes a comma to be placed between every three digits to the left of the decimal separator. For example:

This *expression*	**Formatted with this *fmt***	**Produces**
1234567.8901	#,#.##	1,234,567.89
1234567.8901	#,#.####	1,234,567.8901

Note that while period (.) and comma (,) are always used as the decimal and thousands separators in *fmt*, the output string will contain the decimal and thousands placeholders selected in the international settings for your PC.

Numbers may be scaled (divided) either by inserting one or more commas before the decimal separator or by including a percent sign in the *fmt* specification. Each comma preceding the decimal separator (or after all digits, if no decimal separator is supplied) will scale the number by 1000. The commas will not appear in the output string. A 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 *fmt*. For example:

This *expression*	**Formatted with this *fmt***	**Produces**
1234567.8901	#,.##	1234.57
1234567.8901	#,,.####	1.2346
1234567.8901	#,#,.##	1,234.57
0.1234	#0.00%	12.34%

Characters can be inserted into the output string by including them in the *fmt* specification. The following placeholders will be automatically inserted in the output string in a location matching their position in the *fmt* specification:

- + **$** ( ) space : /

Any set of placeholder characters may be inserted by enclosing them in quotation marks. Any single placeholder character may be inserted by preceding it with a backslash (\).

This *expression*	**Formatted with this *fmt***	**Produces**
1234567.89	**$#**,0.00	**$**1,234,567.89
1234567.89	"TOTAL:" **$#,#**.00	TOTAL: **$**1,234,567.89
1234	\=\>**#,#**\<\=	=>1,234<=

You may want to use the EXTRA! Basic **$CSTRINGS** metacommand or the **Chr** function to embed quotation marks in a format specification. The character code for a quotation mark is 34.

You can format numbers in scientific notation by including one of the following exponent strings in the *fmt* specification:

E-   E+   e-   e+

The exponent string should be preceded with one or more digit placeholders. The number of digit placeholders following the exponent string determines the number of exponent digits in the result. *Fmt* 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 *fmt* results in a sign preceding the exponent in the output.

This *expression*	**Formatted with this *fmt***	**Produces**
1234567.89	**###**.**##**E-00	123.46E04
1234567.89	**###**.**##**e+**#**	123.46e+4
0.12345	0.00E-00	1.23E-01

A numeric *fmt* 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.

This *expression*	**Formatted with this *fmt***	**Produces**
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 Date Times

Both numeric values and variants may be formatted as dates. When formatting numeric values as dates, the value is interpreted according the standard EXTRA! 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.

As with numeric formats, there is a number of predefined formats for formatting dates and times:

A date *expression* of type	**Is displayed as**
General	A date and time, if the number has both integer and real parts (e.g., 11/8/93 1:23:45 PM); as a date, if the number has only an integer part; as time, if the number has only a fractional part.
Long Date	A Long Date. Long Date is defined in the International section of the Control Panel.
Medium Date	A Short Date, but with the month abbreviated (e.g., 08-Nov-93). Short Date is defined in the International section of the Control Panel.

A date *expression* of type	**Is displayed as**
Short Date	A Short Date. Short Date is defined in the International section of the Control Panel.
Long Time	A Long Time. Long Time is defined in the International section of the Control Panel and includes hours, minutes, and seconds.
Medium Time	Hours in 12-hour format, with the AM/PM designator; no seconds.
Short Time	Hours in 24-hour format, with the AM/PM designator; no seconds.

Refer to the following chart for a description of the placeholders you can use to create user-defined date and time formats:

This date or time placeholder character	**Produces**
c	The date time as if the *fmt* was: ‘‘ddddd ttttt’‘. See the definitions below.
ddddd	The date including the day, month, and year based on the PC’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 based on the PC’s current Long Date setting. The default Long Date setting for the United States is mmmm dd, yyyy.
ttttt	The time including the hour, minute, and second based on the PC’s current time settings. The default time format is h:mm:ss AM/PM.

Finer control over the output is available by including *fmt* placeholders that deal with the individual components of the date and time.

This date or time placeholder character	**Produces**
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).
w	The day of the week as a number (where Sunday = 1 and Saturday = 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 number of minutes will be output if the preceding placeholder character is 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 number of minutes will be output if the preceding placeholder character is an hour; otherwise, the month will be output.
mmm	The month of the year as a three-letter abbreviation (e.g., Jan through Dec).
mmmm	The month of the year without abbreviation (e.g., January through December).
q	The quarter of the year as a number (1 through 4).

This date or time placeholder character	**Produces**
y	The day of the year as a number (1 through 366).
yy	The year as a two-digit number (00 through 99).
yyyy	The year as a four-digit number (100 through 9999).
h	The hour as a one- or two-digit number (0 through 23).
hh	The hour as a two-digit number (00 through 23).
n	The minute as a one- or two-digit number (0 through 59).
nn	The minute as a two-digit number (00 through 59).
s	The second as a one- or two-digit number (0 through 59).
ss	The second as a two-digit number (00 through 59).

By default, times will be displayed using a military (24-hour) clock. Several placeholder characters are provided in date / time *fmt* specifications to change this default. They all cause a 12-hour clock to be used. Refer to the following chart for details:

This time indicator	Produces
AM/PM	An uppercase AM for any hour before noon; an uppercase PM for any hour between noon and 11:59 PM.
am/pm	A lowercase am for any hour before noon; a lowercase pm for any hour between noon and 11:59 PM.
A/P	An uppercase A for any hour before noon; an uppercase P for any hour between noon and 11:59 PM.
a/p	A lowercase a for any hour before noon; a lowercase p for any hour between noon and 11:59 PM.
AMPM -or- 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 and AMPM are equivalent.

Any set of placeholders may be inserted into the output by enclosing them in quotation marks. Any single placeholder can be inserted by preceding it with a backslash, (\ ). See Number Formatting, above, for more details.

Formatting Strings

Strings are formatted by examining the *fmt* specification and transferring one placeholder character at a time from the input *expression* to the output string.

Refer to the following chart:

This *fmt* placeholder character	**Does this**
@	Displays a character or a space. If there is a character in the string you want to format in the position where the **@** appears in the *fmt* string, display it; otherwise, display a space in that position.
&	Displays a character or nothing. If there is a character in the string you want to format in the position where the **&** appears in the *fmt* string, display it; otherwise, display nothing.
<	Forces output characters to be displayed in lowercase.
>	Forces output characters to be displayed in uppercase.
!	Causes characters to be transferred from right to left instead of the default, left to right.

Note: A *fmt* 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.

Format (function) 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
```

FreeFile (function) — Return an available file number

rc% = **FreeFile**

Returns the next valid, unused file number.

Parameter	Description
*rc***%**	The return value.

Comments

Use the **FreeFile** function to return the next available file number. The value returned can be used in a subsequent **Open** statement.

FreeFile (function) 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
```

Function ... End Function (statement) — Declare a function

```
[Static] [Private] Function name [(parameter _
   [As type] ... )]
   [As type]
   [statementblock]
   name = expression
End Function
```

Declares the name, arguments, and code that form a function procedure. The purpose of a function is to produce and return a single value of a specified type.

Parameter	**Description**
Static	Specifies that all variables declared within the function will retain their values as long as the program is running, regardless of the way the variables are declared.
Private	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.
name	The name of the function. *name* must be unique and may contain a type-declaration character. *name*'s type determines the type of the value returned. The return type can also be declared by the **As** *type* statement or through a **Def*type* (*Str*)** statement.

Parameter	Description
parameter	Names a variable that will be passed (by reference) to the function when it is called. Multiple parameters must be separated by commas. The data type of a parameter can be indicated using a type character or the **As** clause. Record parameters are declared using an **As** clause and a type that has previously been defined using the **Type** statement. Array parameters are indicated by empty parentheses after parameter. The array dimensions are not specified in the **Function** statement. Array dimensions are defined with an array declaration in the calling procedure. All references to an arrayed parameter within the body of the function must specify the same number of dimensions.
type	Declares the data type of the value returned by the function procedure. Legal data types are: Integer, Long, Single, Double, or String.
statement block	A group of EXTRA! Basic statements that form the body of the function procedure.
expression	The code or expression that will result in the function's desired return value. The function name is assigned to this return value. If you do not assign *expression* to *name*, *name* is set to zero (0) if it is a numeric return value, or a null string if it is a string return value.

Comments

Recursion is supported (that is, a function can call itself). Be careful to avoid stack overflow if you design a recursive function.

When calling the function, you do not need to specify the type declaration character.

The function returns to the caller when the **End Function** statement is reached or when an **Exit Function** statement is executed.

EXTRA! 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 the **Sub ... End Sub** statement to define a procedure with no return value.

Function ... End Function (statement) Example

This example declares a function that is later called by the main subprogram. The function sets its return value to 1.

```
Declare Function EB-exfunction()

Sub main

   Dim y as Integer

   Call EB-exfunction

   y=EB-exfunction

   MsgBox "The value returned by the function is: " & y

End Sub


Function EB-exfunction()

   EB-exfunction=1

End Function
```

FV (function) — Return future value of an annuity

`rc = FV(rate, nper, pmt, pv, due)`

Returns the future value of a constant periodic stream of cash flows (as in an annuity or a loan). The given interest rate is assumed to be constant for the life of the annuity or loan.

Parameter	**Description**
rc	The return value.
rate	The interest rate per period. If payments are on a monthly schedule, *rate* would be .0075 for an annual percentage rate (APR) of 9% (0.9 / 12 = 0.0075).
nper	The total number of payment periods. *nper* for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pmt	The constant payment per period.
pv	The present value or the inital lump sum amount paid (as in the case of an annuity) or received (as in the case of a car loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.

FV (function) 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
```

Get (statement) — Read data from a disk file and copy it into a variable

```
Get[#] filenumber% [recordnumber&], variable
```

Reads from a record from a file opened in Random or Binary mode and writes that data to a variable.

Parameter	**Description**
*filenumber***%**	An integer or integer expression identifying the open file from which you want to read. This is the same *filenumber***%** that is used in the **Open** statement to open the file.
*record number***&**	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. Recordnumber is in the range 1 to 2,147,483,647. If *recordnumber* is omitted, the next record or byte is read. Note that the comma delimiters in the option list are required, even if no *recordnumber* is specified.
variable	The name of the variable into which **Get** will read the file data. *Variable* can be any variable except object, application data type, or array variables (although Single array elements can be used).

Comments

The following should be taken into consideration when opening a file in random mode:

- Blocks of data are read from the file in segments equal to the size specified in the *Len* clause of the **Open** statement. If the size of *variable* is smaller than the record length, the additional data is discarded. If the size of *variable* is larger than the record length, an error occurs.

- For variable length String variables, **Get** reads the two bytes of data that indicate the length of the string, then reads the data into the variable.
- For variant variables, **Get** reads the two bytes of data that indicate the type of the variant, then it reads 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 information, followed by the body of the string.
- User defined types are read as if each member were read separately, except that no padding occurs between elements.

Files opened in binary mode behave similarly to those opened in random mode, except for the following:

- **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 into a string variable is equal to the length of the existing string variable.

Refer to the **Open** statement description for details.

Get (statement) 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
```

GetAttr (function) — Return the attributes of a file, directory, or volume

```
rc = GetAttr(filename$)
```

Returns the attributes of the specified file, directory, or volume label.

Parameter	**Description**
rc	The return value. Refer to the chart below for valid return values and their meanings:
*filename***$**	A string or string expression that indicates the name of the file whose attributes are to be returned. *Filename* cannot contain wildcards.

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.

Comments

You can determine which attributes have been set by performing a bit-wise **And** comparison of the **GetAttr** return value and the value of the attribute you want to check. (Refer to the chart above.) For example:

rc = **GetAttr**(stats.xls) **And** somesymbolicconstantnameTBS

If rc *is not* equal to zero, the attribute is set. If rc *is* equal to zero, the attribute is not set.

Note that file attribute symbolic constants are supplied in the EXTRA! Basic file. Include this file using the $Include metacommand to reference them in your macro.

GetAttr (function) 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
```

GetObject (function) — Return an OLE object

GetObject("*FileName*",[*class*])

Retrieves an OLE Automation object from a file.

Parameter	**Description**
filename	The name of the file that contains the object you want to retrieve. For *EXTRA! Personal Client*, specify a session file. A display session includes an .EDP extension; a printer session includes an EPP extension. If you specify an empty string, then *class* is required.
class	An optional string indicating the name of the application that provides the object, and the type of object. To specify the class, use the following syntax: "*AppName.ObjectType*" Note that you must insert a period (.) between *AppName* and *ObjectType.* *AppName* — For *EXTRA! Personal Client*, the *AppName* is EXTRA. *ObjectType* — For *EXTRA! Personal Client*, the ObjectType is Session.

Comments

GetObject starts the specified session, even if *EXTRA! Personal Client* has not previously been started.

To access the object returned by GetObject, use the Set statement to assign an object reference to a variable. In the example below, an object variable called Ses1 is declared. Then, with the Set statement, a Session object, returned from a file with GetObject, is assigned to the variable Ses1.

```
Dim Ses1 As Object
Set Ses1 = GetObject("c:\program
files\e!pc\sessions\Session1.EDP")
```

You can also use GetObject to retrieve a currently active session. As shown below, you must specify an empty string (" ") for the *FileName* parameter, as well the application name and object type. Note that this statement will result in an error if a Session object has not previously been retrieved.

```
Set ActiveSession = GetObject("","Extra.Session")
```

In addition to starting *EXTRA! Personal Client*, GetObject retrieves objects from any OLE Automation application that stores object in files. Such applications include Microsoft Excel 5.0 or higher and Microsoft Project 4.0 or higher. For information on retrieving an application’s OLE Automation objects, see the documentation for that application.

GetObject (function) Example

This example retrieves a Session object from the file Normal.EDP.

```
Sub Main()
  Dim Sess As Object
  Set Sess = GetObject("C:\Program _
      Files\E!pc\Sessions\Normal.EDP")
  MsgBox Sess.Name
End Sub
```

GetField (function) — Return a substring

*rc***$** = **GetField$(** *string***$**, *fieldnumber***%**, _
*separatorchars***$** **)**

Returns a substring from a source string.

Parameter	Description
*rc***$**	The return value.
*string***$**	The source string from which a field needs to be extracted.
*field number***%**	The designation for which field to extract.
*separator chars***$**	The delimiter character in the string that separates the fields.

Comments

The source *string***$** is considered to be divided into fields by separator characters. Multiple separator characters may be specified. The fields are numbered starting with one.

If *fieldnumber***%** is greater than the number of fields in the string, an empty string is returned.

GetField (function) Example

This example finds the third value in a string, delimited by plus signs (+).

```
Sub main

  Dim teststring,retvalue

  Dim msgtext

  teststring="9+8+7+6+5"

  retvalue=GetField(teststring,3,"+")

  MsgBox "The third field in: " & teststring & " is: " & retvalue

End Sub
```

Global (statement) — Declare global variables

```
Global variableName [As type] [,variableName _
[As type]] ...
```

Declare global variables.

Parameter	Description
variable Name	Indicates the name of the global variable you want to declare.
type	Declares the data type of the global variable.

Comments

Global data is shared across all loaded modules. If an attempt is made to load a module that has a doubly-defined global variable with different data types, the module load will fail.

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 the *variableName*. The two different type-specification methods can be intermixed in a single **Global** statement (although not for the same variable).

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

Global (statement) 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
```

GoTo (statement) — Transfer program control

GoTo *label*

Sends control to a line label.

Parameter	Description
label	A label has the same format as any other EXTRA! Basic name. That is, it must begin with an alphabetic character, end with a colon, and be 40 characters, or fewer, in length. *label* must also be unique to the current module. Line labels are not case-sensitive. Reserved words are not valid labels.

Comments

GoTo cannot be used to transfer control out of the current function or sub function.

Line numbers are not supported in the **GoTo** statement.

GoTo (statement) Example

This example shows a subroutine that illustrates the GoTo statement syntax.

```
Sub Main


' Subroutines begin here

GetUserInput:                     ' Start of first subroutine.

   UserInput$ = InputBox$("Type something ")

   StrLen% = Len(UserInput$)

   If StrLen% Then

      Msg$ = "You entered """ + UserInput$ + """."

   Else

      Msg$ = "You entered nothing or chose Cancel."

   End If


GoTo AllDone:                     ' End of Sub label.
```

```
ShowInput:                        ' Start of second subroutine.
   Msg$ = "You entered """ + UserInput$ + """."  ' Display message.

AllDone:                          ' End of sub label.
MsgBox "You branched around the subroutine ShowInput."
'Display message.

End Sub
```

GroupBox (statement) — Create a group box

```
GroupBox x, y, dx, dy, text$ [, .id]
```

Creates an area within a dialog box that groups a set of similar items.

Parameter	**Description**
x, y	Specifies the position of the group box relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the group box is centered horizontally within the client area. If *y* is omitted, the group box is centered vertically within the client area.
dx, dy	Specifies the width and height of the group box. *dx* is measured in one-quarter system-font character-width units. *dy* is measured in one-eighth system-font character-width units. A *dy* value of 14 typically accommodates system font text.
*text***$**	Supplies the title for the group box. If the width of this string is greater than *dx*, trailing characters are truncated. If *text***$** is an empty string (“ “), the top border of the group box will be a solid line.
.id	An optional identifier used by the dialog statements that act on this control.

Comments

The **GroupBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

GroupBox (statement) Example

This example creates a dialog box with two group boxes.

```
Sub main
   Begin Dialog UserDialog 242, 146, "Print Dialog Box"
      '$CStrings Save
      GroupBox  115, 14, 85, 57, "Page Range"
      OptionGroup .OptionGroup2
         OptionButton  123, 30, 46, 12, "All Pages", .OptionButton1
         OptionButton  123, 50, 67, 8, "Current Page", .OptionButton2
      GroupBox  14, 12, 85, 76, "Include"
      CheckBox  26, 17, 54, 25, "Pictures", .CheckBox1
      CheckBox  26, 36, 54, 25, "Links", .CheckBox2
      CheckBox  26, 58, 63, 25, "Header/Footer", .CheckBox3
      PushButton  34, 115, 54, 14, "Print"
      PushButton  136, 115, 54, 14, "Cancel"
      '$CStrings Restore
   End Dialog
   Dim mydialog as UserDialog
   Dialog mydialog
End Sub
```

Hex (function) — Return a number in hexadecimal

```
rc$ = Hex[$](numeric-expression)
```

Returns the hexadecimal representation of a numeric-expression.

Parameter	**Description**
*rc***$**	The return value.
numeric-expression	Any numeric data type.

Comments

If the numeric expression is an integer, the return string will contain up to four hexadecimal digits; otherwise, the expression will be converted to a Long integer, and the return string will contain up to eight hexadecimal digits.

The dollar sign (**$**) in the function name is optional. If the dollar sign is included, the return type is String. If the dollar sign is omitted, Hex will return a variant of vartype 8 (String).

Hex (function) Example

The example uses Hex to return the hexadecimal representation of a decimal number.

```
Msg$ = "Enter a number. "
Num% = Val(InputBox$(Msg$))       ' Get user input.

Msg$ = LTrim$(Str$(Num%)) + " decimal is &H"
Msg$ = Msg$ + Hex$(Num%) + " in hexadecimal notation."
MsgBox Msg$                       ' Display results.
```

Hour (function) — Return the hour of day portion of a date/time value

rc = **Hour(***expression***)**

Returns an integer that represents the hour of the day derived from the supplied string or numeric expression.

Parameter	**Description**
rc	The return value (an integer between 0 and 23).
expression	Any type of expression (numeric or string) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Numbers to the left of the decimal point are interpreted as a date; numbers to the right are interpreted as time values. Negative numbers are interpreted as dates prior to December 30, 1899.

Comments

If expression is Null, Hour returns a Null.

Hour (function) 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
```

If...Then...Else (statement) — Allow conditional execution of an instruction block

```
If condition Then thenstatement [Else elsestatement]
```

-or-

```
If condition Then
   statementblock
[ElseIf expression Then
   statementblock]...
[Else
   statementblock ]
End If
```

Allows you to organize an alternative action into a separate block of code. The resulting action depends on the logical value of one or more conditions supplied in the structure.

Parameter	**Description**
condition	A numeric or string expression that evaluates to a Boolean value.
then statement	The statement that is executed when *condition* is true. *thenstatement* can be one (for single-line syntax, shown above) or more (multiple-line syntax) EXTRA! Basic statements or a **GoTo** statement pointing to a valid program label.
else statement	The statement that is executed when condition is false. *elsestatement* can be one (single-line syntax, shown above) or more (multiple-line syntax) EXTRA! Basic statements or a **GoTo** statement pointing to a valid program label.
statement block	Consists of one or more EXTRA! Basic statements, separated with colons. These statements are executed if the enclosing **If**, **Else**, or **ElseIf** statement is true.

Comments

First EXTRA! Basic tests the *condition* expression. If the result of the expression is true, the statements following **Then** are executed. If the result is false, EXTRA! Basic evaluates each **ElseIf** condition until it finds one that evaluates to true. Then, the statements immediately following the associated **Then** are executed. If none of the **ElseIf** conditions are true, the statements following the **Else** are executed. After a **Then** or **Else** statement has been executed, control returns to the statement following **End If**.

In the single-line version of the **If** statement, the *thenstatement* and *else_statement* can be any valid single statement. Multiple statements separated by colons (:) are not allowed. When multiple statements are required in either the **Then** or **Else** clauses, use the multiple-line version of the **If** statement syntax.

In the multiple-line version, the *statementblock(s)* can be separated by colons (:) or located on subsequent, consecutive lines.

If...Then...Else (statement) Example

The example illustrates the various forms of the If...Then...Else syntax.

```
Do
      Msg$ = "Enter a number greater than 0 and less than 100,000:"

   X& = Val(InputBox$(Msg$))
   If X& > 0 And X& < 100000 Then    ' Check range of input.
      Exit Do                        ' Exit loop if in range.
   Else
      Msg$ = " ERROR: Out of Range! "
      MsgBox Msg$                    ' Display error message.
   End If
Loop
' Determine number of digits in number input by the user.
If X& < 10 Then
   Y% = 1                            ' 1 digit
ElseIf X& < 100 Then
   Y% = 2                            ' 2 digits
ElseIf X& < 1000 Then
   Y% = 3                            ' 3 digits
ElseIf X& < 10000 Then
```

```
   Y% = 4                              ' 4 digits
Else
   Y% = 5                              ' 5 digits
End If

' Single-line form of the If...Then structure for singular
' or plural digits
If Y% > 1 Then Unit$ = " digits." Else unit$ = " digit."
Msg$ = "The number you entered has " + LTrim$(Str$(Y%)) + Unit$
MsgBox Msg$                          ' Display message.
```

$Include (metacommand) — Include statements from another file, by reference

```
'$Include: "filename"
```

Instructs the compiler to copy statements and/or instructions from another file to the current location in the current file. The statements are copied in by reference.

Parameter	**Description**
filename	The valid path and filename of the remote file containing the statements and/or instructions you want to include in the current file.

Comments

Comments that include metacommands are only recognized when located at the beginning of a line. For compatibility with other versions of EXTRA! Basic, you can use apostrophes (') to enclose *filename*.

To comment an Include statement, use the **Rem** statement.

A file extension of .EBH is suggested as a convention for EXTRA! include files. Any other valid file extension may be used.

Include files must be located either in your program directory (for example, C:\EXTRAWIN) or in your USER directory (for example, C:\EXTRAWIN\USER). The EXTRA! Basic macro compiler will not search the DOS path for Include files.

Create a different Include search path by creating a section named [MACRO] in your EXTRA4W.INI file (located in your USER directory) and add the following:

```
IncludePath = path [;path;...]
```

where *path* is the drive and directory you want EXTRA! to search.

Any files used must be saved as text files, including macros.

Include (metacommand) 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 EXTRA! Basic 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
```

Input (function) — Return characters read from a file

*rc***$** = **Input$ (***numchars***%**, [**#**]*filenumber***%)**

Returns a string containing characters read from the specified sequential file.

Parameter	**Description**
*rc***$**	The return value.
*numchars***%**	The number of characters (or bytes) to be read from the file. *numchars***%** must be less than 65,535.
*filenumber***%**	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number.

Comments

The file pointer is advanced the number of characters read. Unlike the **Input #** statement, **Input$** returns all of the characters it reads, including carriage returns, line feeds, commas, quotation marks, and leading spaces.

Input (function) 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)
   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
```

Input (statement) — Assign a line of data to a variable

```
Input [#] filenumber%, variable[, variable]...
```

-or-

```
Input [prompt$,] variable[, variable]...
```

Reads a line of data from a sequential file and assigns each line read to a variable.

Parameter	**Description**
*filenumber***%**	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number.
variable	The name of a variable.Any variable type is allowed except user-defined variables and array variables.
*prompt***$**	An optional string that can be used to prompt for keyboard input.

Comments

The **Input #** statement will read as many lines as there are variables to fill. However, an attempt to read past an End of File will result in a an error.

The *filenumber***%** argument is an integer expression identifying the open file to read from. This is the number used in the **Open** statement to open the file. The *variable* arguments list the variables that are assigned the values read from the file. The list of *variables* is separated by commas.

If *filenumber***%** is not specified, the user is prompted for keyboard input with an empty message box unless *prompt***$** is specified. Only one variable can be specified if *prompt***$** is used.

The **Input #** statement advances the file pointer to the beginning of the next line of the file.

End of Input Characters

When a variable is

- **numeric:** the number is assumed to begin at the first nonspace character. When a space, a comma, or the end of a line is encountered, it is assumed to be the end of the number. Blank lines are read as zero. If the data is not a valid number, the variable is assigned a value of zero.
- **string:** the character string is assumed to begin at the first nonspace character. If the first character is a quotation mark (‘‘), it is ignored, and all characters following it (including spaces and commas) up to the next quotation mark are read into the variable. When the string is not delimited by quotation marks, the end of a string is assumed when a comma, a space, or the end of a line is encountered. Blank lines are read as zero-length strings.

Input (statement) 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. 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.

```
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
```

InputBox (function) — Display prompt and field for user input

```
rc$ = InputBox[$](prompt$ [,title$ [,default$ _
   [,xpos%,ypos%]]])
```

Displays a dialog box containing a prompt and a field for user input.

Parameter	**Description**
*rc***$**	The return value.
*prompt***$**	A string expression containing the prompt that will be displayed next to the input field. The length of *prompt***$** is restricted to approximately 255 characters depending on the width of the characters used. Note that a carriage return and a line-feed character must be included in *prompt***$** if it spans multiple lines.
*title***$**	The text that will appear in the dialog box's title bar. If this parameter is omitted, the title bar remains blank.
*default***$**	The string expression that will be shown in the input field as the default response. If this parameter is omitted, the field is initialized with a Null string.
*xpos***%**, *ypos***%**	Numeric expressions, specified in dialog box units, that determine the position of the input box. *xpos***%** determines the horizontal distance between the left edge of the screen and the left border of the input box. *ypos***%** determines the horizontal distance from the top of the screen to the input box's upper edge.

Comments

If *xpos***%** and *ypos***%** are not supplied, the input box is centered roughly one third of the way down the screen. A horizontal input box unit is one-quarter the average character width in the system font; a vertical dialog box unit is one-eighth the height of a character in the system font.

When the user presses the ENTER key, 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.

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

InputBox (function) 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
```

InStr (function) — Return a substring

```
rc% = InStr([position%,] string$, substring$)
```

-or-

```
rc% = InStr(position%, string$, substring$[,
comparetype%])
```

Returns an integer representing the position of the first occurrence of a substring within a string.

Parameter	**Description**
*rc***%**	The return value.
*position***%**	Indicates the offset where the search for *string***$** should begin. If *position***%** is not supplied, the search starts at the beginning of *string***$** (that is, *position***%** equals 1).
*string***$**	The string in which you want to search for *substring***$**. *string***$** can be a string variable, string expression, or string literal.
*substring***$**	The string you want to locate within *string***$**. *substring***$** can be a string variable, string expression, or string literal.
*compare type***%**	Determines the comparison method used. If *comparetype***%** is 0, a case-sensitive comparison is performed based on the ANSI character set sequence. If *comparetype***%** is 1, a case-insensitive comparison is performed based on the relative order of characters as determined by the country code setting for your system. If omitted, the module-level default (specified with the **Option Compare** statement) is used.

Comments

InStr will return a zero if any of the following conditions are met:

- *position***%** is greater than the length of the substring
- *string***$** is a Null string
- *substring***$** cannot be located

If *substring***$** is a Null string, *position***%** will be returned. If *string***$** or *substring***$** is a Null variant, **Instr** returns a Null variant.

InStr (function) 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
```

Int (function) — Return integer portion of a number

rc = **Int(***numeric-expression***)**

Returns the integer portion of a numeric expression.

Parameter	**Description**
rc	The return value. The return value type matches the type of the numeric expression. This includes variant expressions which will return a result of the same vartype as the input, with the following exceptions: vartype 8 (String) will be returned as vartype 5 (Double) and vartype 0 (empty) will be returned as vartype 3 (Long).
numeric-expression	*numeric-expression* can be of any numeric data type.

Comments

For both positive and negative *numeric-expressions*, **Int** removes the fractional portion of the expression and returns only the integer portion.

Important: For negative numeric expressions, **Int** returns the first negative integer *less* than or equal to *numeric-expression*, and **Fix** returns the first negative integer *greater* than or equal to *numeric-expression*. For example, **Int** converts -5.6 to -6 and **Fix** converts -5.6 to -5.

If the numeric expression evaluates to a null, **Int** returns a null.

Int (function) 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
```

IPmt (function) — Return the interest portion of an annuity payment

```
rc = IPmt(rate, per, nper, pv, fv, due)
```

Returns the interest portion of the payment for a given period of an annuity. The interest rate is considered to be constant throughout the life of the loan.

Parameter	Description
rc	The return value.
rate	The interest rate per period. If payments are on a monthly schedule, rate would be .0075 for an annual percentage rate (APR) of 9% (0.9 / 12 = 0.0075).
per	A particular payment period in the range 1 through *nper*.
nper	The total number of payment periods. nper for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pv	Present value (in or the initial lump sum paid (as in an annuity) or received (as in a loan).
fv	The future value of the final lump sum amount required (as in a savings plan) or paid (0, in the case of a loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.

IPmt (function) 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
```

IRR (function) — Return the internal rate of return

```
rc = IRR(valuearray(), guess)
```

Returns the internal rate of return (interest rate) for a stream of periodic cash flows.

Parameter	Description
rc	The return value.
valuearray	An array containing cash flow values. The array must contain at least one positive value (representing a receipt) and one negative value (representing a payment).
guess	A rough estimate of the value returned by **IRR**. In general, between 0.1 (10%) and 0.15 (15%) is a reasonable *guess*.

Comments

All payments and receipts must be present in *valuearray* in the correct sequence. The value returned by **IRR** will vary with a change in the sequence of cash flows.

IRR is an iterative function. EXTRA! Basic cycles through an IRR calculation until the result is accurate to within 0.00001 percent. If IRR does not produce a result after 20 iterations, IRR fails.

IRR (function) 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 intnl 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
   intnl=IRR(cashflows(),guess)
   msgtext="The IRR for your cash flow amounts is: "
   msgtext=msgtext & Format(intnl, "Percent")
   MsgBox msgtext
End Sub
```

Is (operator) — Compares two object expressions

```
rc = objectExpression1 Is objectExpression2
```

Compares two object expressions and returns a value that indicates whether the expressions refer to the same object.

Parameter	**Description**
rc	The return value. -1 (true) if the two object expressions refer to the same object. 0 (false), if they do not refer to the same object.
object Expression	An expression that points to a particular object.

Comments

Also use the **Is** operator to test whether an object variable has been **Set** to **Nothing**.

Is (operator) 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
```

IsDate (function) — Determines whether a value can be interpreted as a date

```
rc = IsDate(variant)
```

Evaluates whether a variant argument can be converted to a date.

Parameter	**Description**
rc	The return value.
variant	Any variant or variant expression of vartype 7 (date) or vartype 8 (String).

Comments

Valid dates range from January 1, 100 A.D. through December 31, 9999 A.D.

IsDate (function) 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
```

IsEmpty (function) — Indicates whether a variant has been initialized

rc = **IsEmpty(***variant***)**

Returns a value that indicates whether or not the specified variant argument has been initialized.

Parameter	**Description**
rc	The return value.
variant	A variant argument (usually a single variable name as opposed to a variant expression).

Comments

IsEmpty returns True (-1) if the variant of vartype 0 (Empty); otherwise **IsEmpty** returns False (0). Any newly-defined variant is Empty, by default, indicating that it is not initialized. Note that Empty differs from Null in that Null indicates that the variant does not contain any data.

An Empty variant converts to zero when used in a numeric expression. An Empty variant converts to an Empty String when used in a String expression.

IsEmpty (function) Example

This example prompts for a series of test scores and uses IsEmpty to determine whether the maximum allowable limit has been reached. (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
```

IsNull (function) — Indicates whether an expression results in a Null

```
rc = IsNull(variant)
```

Returns a value that indicates whether or not the specified variant or variant expression contains or has resulted in a Null value.

Parameter	**Description**
rc	The return value.
variant	A variant or variant expression.

Comments

IsNull returns True (-1) if the variant contains a Null value. IsNull returns False (0) if the variant does not contain a Null value.

Note that Null differs from Empty in that Empty indicates that the variant has not been initialized.

Important: Because of Null propagation, although the following expressions might appear to potentially return True, they will always return False simply because they contain a Null:

```
If variant = Null ...
If variant <> Null ...
```

Use IsNull to guarantee correct interpretation of a variant expression containing a Null.

IsNull (function) 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
```

IsNumeric (function) — Indicates whether a variant can be converted to a number

```
rc = IsNumeric(variant)
```

Evaluate whether a variant or the result of a variant expression can be converted to a numeric data type.

Parameter	**Description**
rc	The return value.
variant	A variant or variant expression.

Comments

IsNumeric returns True (-1) if the variant or the result of a variant expression is of vartype 0 (Empty) and 2 through 6 (numeric) or is a vartype 8 (String) that can be interpreted as a number. If not, **IsNumeric** returns False (0).

IsNumeric (function) 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
```

Kill (statement) — Delete file(s)

Kill *filespec***$**

Deletes files from a disk.

Parameter	Description
*filespec***$**	A string expression that specifies a valid DOS file path and filename. *filespec***$** can contain wildcards.

Comments

An error is produced if *filespec***$** specifies an open file.

Kill deletes files only; it does not delete directories. Use the **RmDir** statement to delete directories.

Kill (statement) Example

This example uses Kill to remove a file the user specifies.

```
On Error GoTo Errhandler              ' Set up error handler.
Msg$ = "Enter the name of the file you want to delete."
DelFile$ = UCase$(InputBox$(Msg$))    ' Get filename.
If Len(DelFile$) Then                 ' Check for entry.
   Msg$ = "Are you absolutely sure you want to delete " + DelFile$
   Msg$ = Msg$ + " from your disk?"
   Ansr% = MsgBox(Msg$, 4)            ' Make sure.
   If Ansr% = 6 Then                  ' User chose "Yes."
      Msg$ = "Deleting " + DelFile$ + " from your disk."
      Kill DelFile$                   ' Delete file from disk.
   Else
      Msg$ = DelFile$ + " was not deleted."
   End If
Else
   Msg$ = "You didn't enter a file name."
End If
MsgBox Msg$                           ' Display message.
```

```
Exit Sub

Errhandler:

If Err = 53 Then                      ' Error 53 is "File not Found".
   Msg$ = "Sorry, the file you named could not be found."
End If
Resume Next
```

LBound (function) — Return lowest subscript of a dimensioned array

```
rc% = LBound (arrayVariable [,dimension])
```

Returns the smallest available subscript for the specified dimension of the specified array.

Parameter	**Description**
*rc***%**	The return value.
arrayVariable	The name of an array.
dimension	An integer indicating which dimension's lower bound is returned. 1 equals the first dimension, 2 equals the second, and so forth. If dimension is not supplied, it is assumed to be 1.

Comments

For the array shown below, **LBound** returns the following:

```
Dim Sales(1 to 3, 4 to 6)
```

Statement	**Return Value**
LBound(sales, 1)	1
LBound(sales, 2)	4

Depending on how **Option Base** has been set, the default lower bound for any dimension is either 0 or 1. Any integer value is valid as a lower bound for dimensions set using the **To** clause in a **Dim**, **Global**, or **ReDim** statement.

Use **LBound** in conjunction with **UBound** to determine the length of an array.

LBound (function) Example

The example uses the LBound function to determine the lower bounds for a three-dimensional array.

```
NL$ = Chr$(13) + Chr$(10): TB$ = Chr$(9)     ' Define newline, tab.
```

```
'Generate some random dimensions between 2 and 10 for array size.
Randomize
A% = -Int(9 * Rnd + 2)                        ' First dimension.
B% = Int(9 * Rnd + 2)                         ' Second dimension.
C% = Int(9 * Rnd + 2)                         ' Third dimension.

ReDim Array%(A% to 20, B% to 20 , C% to 20)  ' Set dimensions.

Msg$ = "The test array has the following lower bounds: " + NL$
Msg$ = Msg$ + TB$ + "Dimension 1 = " + Str$(LBound(Array%, 1)) + NL$
Msg$ = Msg$ + TB$ + "Dimension 2 = " + Str$(LBound(Array%, 2)) + NL$
Msg$ = Msg$ + TB$ + "Dimension 3 = " + Str$(LBound(Array%, 3))
MsgBox Msg$                                  ' Display message.
```

LCase (function) — Return a string in lowercase

*rc***$** = **LCase**[**$**] **(***string***$)**

Returns a copy of a source string, with all uppercase letters converted to lowercase.

Parameter	Description
*rc***$**	The return value.
*string***$**	Any string expression.

Comments

Conversion is based on the country specified in the Windows.

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified, the return type is String. If the dollar sign is omitted, **Lcase** will typically return a variant of vartype 8 (String). If the value of *string* is Null, a variant of vartype 1 (Null) is returned.

LCase (function) Example

The example uses LCase to return an all-lowercase version of the argument string.

```
Uppercase$ = "ONCE UPON A TIME"    ' String to convert.


Lowercase$ = LCase$(UpperCase$)    ' Convert to lowercase.

Msg$ = "LCase$ converts """ + Uppercase$ + """ to """

Msg$ = Msg$ + Lowercase$ + ". """

MsgBox Msg$                        ' Display message.
```

Left (function) — Return a substring

`rc$ = Left[$] (string$, length%)`

Returns the leftmost *length*% characters of a source string.

Parameter	**Description**
rc$	The return value.
string$	The source string from which the substring will be returned.
length%	A Long integer expression that indicates how many characters Left$ should return.

Comments

The dollar sign ($) in the function name is optional. If the dollar sign is specified, the return type is String. If the dollar sign is omitted, **Left** will typically return a variant of vartype 8 (String). If the value of *string*$ is Null, a variant of vartype 1 (Null) is returned.

If the length of *string*$ is less than *length*%, **Left** returns the entire string.

Left (function) Example

The example uses the Left function to return the first of two words input by the user.

```
Msg$ = "Enter two words separated by a space."
UsrInp$ = InputBox$(Msg$)                     ' Get user input.


   SpcPos% = InStr(1, UsrInp$, " ")           ' Find space.
   If SpcPos% Then
      LeftWord$ = Left$(UsrInp$, SpcPos% - 1) ' Get left word.
      RightWord$ = Right$(UsrInp$, Len(UsrInp$) - SpcPos%)
' Get right.
      Msg$ = "The first word you entered is """ + LeftWord$
      Msg$ = Msg$ + "." + """"  + " The second word is """
      Msg$ = Msg$ + RightWord$ + "." + """"
   Else
      Msg$ = "You didn't enter two words."
   End If
   MsgBox Msg$                                ' Display message.
```

Len (function) — Return length of string or storage bytes required

```
rc% = Len (string$)
```

-or-

```
rc% = Len (variable)
```

Returns the length of the specified string parameter or the number of bytes required to store a variable.

Parameter	**Description**
rc%	The return value.
string$	Any string or string expression.
non-string	The name of a variable. This variable can be of any user-defined or EXTRA! Basic data types. The data type must be indicated.

Comments

If **Len** is passed a string, the length of that string is returned. If **Len** is passed a variant variable name, the number of bytes required to store that variable is returned; otherwise, the length of the built-in or user-defined data type is returned. If *variable* is a variant containing a Null value, **Len** returns a Null variant.

Len (function) Example

The example shows how Len can be used to return the number of bytes needed to store each fundamental EXTRA! Basic data type.

```
Dim A As Integer, B As Long
Dim C As Single, D As Double

NL$ = Chr$(13) + Chr$(10)        ' Define newline.
A% = 1                           ' Integer variable.
B& = A%                          ' Long integer.
C! = B&                          ' Single-precision, floating-point.
D# = C!                          ' Double-precision, floating-point.
```

```
Msg$ = "Integer values are stored in" + Str$(Len(A%)) + " bytes."
Msg$ = Msg$ + NL$ + "Long integer values are stored in"
Msg$ = Msg$ + Str$(Len(B&)) + " bytes." + NL$
Msg$ = Msg$ + "Single-precision values are stored in"
Msg$ = Msg$ + Str$(Len(C!)) + " bytes." + NL$ + "Double-precision "
Msg$ = Msg$ + "values are stored in" + Str$(Len(D#)) + " bytes."
Msg$ = Msg$ + NL$ + "This message contains "
Msg$ = Msg$ + "more than" + Str$(Len(Msg$)) + " characters."
MsgBox Msg$                      ' Display message.
```

Let (statement) — Assign a value to a variable

```
[Let] variable = expression
```

Stores a value in an EXTRA! Basic variable.

Parameter	Description
variable	The name of the variable that will hold the value of *expression*.
expression	The value that will be assigned to *variable*.

Comments

The keyword **Let** is optional.

In addition to standard string or numeric assignments, you can also use the **Let** statement to assign to a record field or to an element of an array.

Standard conversion rules apply when assigning a dissimilar value to a numeric or string variable.

Let (statement) Example

The example uses statements with and without the Let reserved word to assign the value of an expression to a variable.

```
Let Pi# = 4 * Atn(1)      ' Assign with Let keyword.


'Assign Msg$ without Let.
Msg$ = "The area of circle whose radius is 3 inches is: "
Msg$ = Msg$ + Str$(Pi# * (3^2)) + " inches."
MsgBox Msg$               ' Display message.
```

Like (operator) — Indicates whether two strings match

```
rc = string Like pattern
```

Used to compare a string or string expression against a supplied pattern (including wildcard characters).

Parameter	Description
rc	The return value.
string	A string or string expression.
pattern	A string or string expression that may contain one or more of the following wildcard characters:

This character	Matches any
?	single character
*****	set of zero or more characters
#	single-digit character
[*chars***]**	single character in *chars*
[!*chars***]**	single character not in *chars*
[*schar*-*echar***]**	single character in the range *schar* to *echar,* ascending
[!*schar*-*echar***]**	single character not in the range *schar* to *echar*, ascending
[*schar*-*echar* *schar*-*echar*...**]**	single character in the ranges *schar* to *echar,* ascending or *schar* to *echar,* ascending
[!*schar*-*echar***]** *schar*-*echar*...**]**	single character not in the range *schar* to *echar*, ascending or *schar* to *echar,* ascending

Comments

The **Like** operator returns True (-1) if the string or string expression matches *pattern*, and False (0) if it does not. If either *string* or *pattern* is a Null, the result of the **Like** operation is also Null.

Case sensitivity and sorting order are determined by the **Option Compare** statement. (**Option Compare Binary** results in a **Like** operation that is case sensitive. **Option Compare Text** results in a **Like** operation that is not case sensitive.) If no Option Compare statement is found, EXTRA! Basic defaults to a case sensitive comparison (equivalent to Option Compare Binary).

chars pattern rules:

- The special wildcard characters bracket ([ ), question mark (?), number sign (#), and asterisk (*) can be used as a pattern char, if enclosed in brackets. For example:

  ```
  rc = "rev199?.xls" Like rev199[?].xls
  ```
- To match a hyphen in chars, place it either at the beginning of the chars (after the exclamation point, if used) or at the end of chars. For example:

  ```
  rc = socsec-no Like [-0000-1000]
  ```
- Œ, œ, Æ, and æ are special single characters that, in certain languages, represent two characters. The **Like** operator considers these special characters equivalent to O and E, o and e, A and E, and a and e, respectively. Therefore, an occurrence of Æ in a string and AE in an expression will return True (provided the language specified in the WIN.INI file uses this special character).

Like (operator) 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:")
   retvalue=userstr LIKE pattern
```

```
   If retvalue=-1 then
      msgtext="The letter " & userstr & " is lowercase."
   Else
      msgtext="Not a lowercase letter."
   End If
   Msgbox msgtext
End Sub
```

Line Input (statement) — Read a line into a variable

`Line Input` `[#]` *`filenumber`*`%,` *`variable`***`$`**

-or-

`Line Input` `[`*`prompt`***`$`**`,]` *`variable`***`$`**

Reads a line from a sequential file into a string variable, or takes input from a message box.

Parameter	**Description**
filenumber%	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number. The **#** preceding the *filenumber* parameter is required.
variable$	The name of the variable that will receive the line of text from the file.
prompt$	The *prompt*$ parameter is a string expression containing the prompt that will be displayed next to the input field. The length of *prompt*$ is restricted to approximately 255 characters depending on the width of the characters used. Carriage return and line-feed characters must be included in *prompt*$ if it spans multiple lines.

Comments

If neither *filenumber*% nor *prompt*$ is supplied, *prompt*$ is assumed and an empty message box requests user input.

Line Input (statement) Example

The example uses the Line Input # statement to read and display the first line in the CONFIG.SYS file.

```
NL$ = Chr$(13) + Chr$(10)                ' Define newline.

Msg$ = "Which drive does your operating system boot from ?"

Drive$ = UCase$(Left$(InputBox$(Msg$), 1)) ' Get user input.
```

```
If Drive$ = "C" or Drive$ = "A" Then
   FileName$ = Drive$ + ":\CONFIG.SYS"
   If Dir$(FileName$) <> "" Then           ' Check if file exists.
      Open FileName$ For Input As #1       ' If it does, open it.
      Line Input #1, TextLine$             ' Get complete line.
      Close #1                             ' Close file.
      Msg$ = "The first line of your CONFIG.SYS file is:"
      Msg$ = Msg$ + NL$ + TextLine$ + "."
   Else                                    ' No CONFIG.SYS.
      Msg$ = "Could not find a CONFIG.SYS file on drive "
      Msg$ = Msg$ + UCase$(Drive$) + ":"
   End If
Else
   Msg$ = "You did not provide a valid boot drive. Systems "
   Msg$ = Msg$ + "normally boot from either A: or C: drive."
End If
MsgBox Msg$                                ' Display message.
```

ListBox (statement) — Create a list of user choices

```
ListBox x, y, dx, dy, text$, .field
```

–or–

```
ListBox x, y, dx, dy, stringarray$(), .field
```

Creates a list of choices from which users can select.

Parameter	Description
x, y	*x* and *y* specify the position of the list box relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the list box is centered horizontally within the client area. If *y* is omitted, the list box is centered vertically within the client area.
dx, dy	The combined width of the list box and the *text*$ field. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text*$ parameter (including blanks and punctuation) by 4 and add 12. The *dy* argument is the height of the *text*$ parameter. 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 list box and the accompanying text will move downward within the dialog box.
text$	A string containing the selections for the list box. This string must be defined using a **Dim** statement, before the **Begin Dialog** statement is executed.

Parameter	**Description**
string array$( )	An array of dynamic strings that will be used to populate the list box.
.field	The *.field* parameter is the name of the dialog-record field that will hold the list box selection. When the user selects OK (or selects the customized button created using the Button statement), a number representing the selection's position in the *text*$ string is recorded in the *.field* field. *listchoice* numbers begin at 0. If no item is selected, *.field* is set to -1.

Comments

The **ListBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

ListBox (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, "E! Basic 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
```

Loc (function) — Return current record number or offset into specified open file

rc = **Loc(***filenumber***%)**

Returns the current offset within the specified open file.

Parameter	**Description**
rc	The return value. For files opened in Random mode, *rc* is the number of the last record read or written to. For files opened in Append, Input, or Output mode, *rc* is the current byte offset divided by 128. For files opened in Binary mode, *rc* is the offset of the last byte read or written to.
*filenumber***%**	An integer or integer expression identifying the open file to query. This argument should reference the same parameter specified in the **Open** statement.

Loc (function) 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
```

Lock, Unlock (statements) — Controls access to some or all of an open file

```
Lock [#]filenumber%[, {record | [start&] [To end&]}]
...
Unlock [#]filenumber%[, {record | [start&] [To
end&]}]
```

In a networked environment, controls access by other processes to some or all of the records or bytes in an open file.

Parameter	**Description**
filenumber	An integer or integer expression identifying the open file. This argument should reference the same parameter specified in the **Open** statement.
record	The number of the block or record you want to lock or unlock (1 to 2,147,483,647, inclusive).
start	A Long integer indicating the first byte or record you want to lock or unlock.
end	A Long integer indicating the last byte or record you want to lock or unlock.

Comments

Important: The arguments passed to **Lock** and **Unlock** must match exactly. Also, locked open files must be unlocked before closing or unpredictable results may occur.

For files opened in Random mode, *start* and *end* are record numbers. For files opened in Binary mode, *start* and *end* are byte offsets. For Input, Output, and Append modes, *start* and *end* are ignored and the whole file is locked or unlocked.

If an *end* argument is supplied without a *start* argument, all records or bytes from *record* or offset 1 to *end* are locked or unlocked. If a *start* argument is supplied without an *end* argument, only the record or byte at the location indicated by *start* is locked or unlocked.

The number sign (#) in the Lock and Unlock statements is optional.

Lock, Unlock (statements) 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
```

Lof (function) — Return the length of an open file

```
rc% = Lof (filenumber%)
```

Returns the length of the file specified by *filenumber***%**.

Parameter	Description
*rc***%**	The return value.
*filenumber***%**	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number.

Comments

The value returned by **Lof** is in bytes.

Lof (function) Example

The example uses LOF to determine the size of an open disk file.

```
NL$ = Chr$(13) + Chr$(10)           ' Define newline.


'Make sample data file.
Open "LOFDATAX" For Output As #1    ' Open file for output.
For I% = 0 To 200                   ' Generate random values.
   Print #1, Int((500 - 100 + 1) * Rnd + 100)
Next I%
Close #1                            ' Close file.


Open "LOFDATAX" For Input As #1     ' Open file just created.
FileLength% = LOF(1)                ' Get length of file.
Close #1                            ' Close file.
```

```
Msg$ = "The length of the LOFDATAX file just created is "
Msg$ = Msg$ + LTrim$(Str$(FileLength%)) + " bytes." + NL$ + NL$
Msg$ = Msg$ + "Choose OK to remove the sample data file."
MsgBox Msg$                        ' Display message.

Kill "LOFDATAX"                    ' Delete file from disk.
```

Log (function) — Return the logarithm of a number

```
rc% = Log(numeric-expression)
```

Returns the natural logarithm of a numeric expression.

Parameter	**Description**
*rc***%**	The return value.
numeric-expression	*numeric-expression* can be of any numeric data type. *numeric-expression* must result in a value greater than 0.

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

Comments

An Overflow error will occur if *numeric-expression* exceeds 709.782712893. The value of the constant *e* is approximately 2.718282.

Log (function) 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
```

LSet (statement) — Left align string, assign one user-defined type variable to another

```
LSet string$ = string-expression
```

–or–

```
LSet variable1 = variable2
```

The first form of **LSet** left-aligns a string within a string variable. The second form copies a variable of user-defined type to another variable of a different user-defined type.

Parameter	Description
string	The name of a string variable.
string-expression	The string that is to be left-aligned within string.
variable1	The destination user-defined variable.
variable2	The source user-defined variable.

Comments

Form 1 of LSet:

If *string* is shorter than *string-expression*, **LSet** copies the leftmost characters of *string-expression* into *string*. The number of characters copied is equal to the length of *string;* extraneous characters are truncated. If *string* is longer than *string-expression*, all characters in *string-expression* are copied into *string* from left to right and extra *string* characters are replaced with spaces.

Form 2 of LSet:

The number of characters copied is equal to the length of the shorter of *variable1* and *variable2*.

General:

LSet cannot be used to assign variables of different user-defined types if either contains a variant or variable-length string.

LSet (statement) Example

This example puts a user’s last name into the variable LASTNAME. If the name is longer than the size of LASTNAME, the user’s name is truncated.

```
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
```

LTrim (function) — Remove leading spaces

rc$ = **LTrim**[$]**(***string*$**)**

Returns a copy of a source string, with all leading space characters removed.

Parameter	**Description**
rc$	The return value.
string$	Any string expression.

Comments

The dollar sign (**$**) in the function name is optional. If specified, the return type is String. If the dollar sign is 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.

LTrim (function) Example

The example uses LTrim and RTrim, to strip leading and trailing spaces from a string variable. Stripping of both leading and trailing spaces can be done more efficiently using the Trim function.

```
Dim Msg$, NL$, TestStr$, TestStr1$   ' Declare variables.


NL = Chr(10)                         ' Define newline.

TestStr = "  Test String  "

TestStr1 = LTrim(RTrim(TestStr))     ' Strip spaces left and right.

Msg = "The original TestStr """ + TestStr + """ was "

Msg = Msg + STR$(Len(TestStr)) + " characters long. There were two "

   Msg = Msg + "leading spaces and two trailing spaces." + NL + NL

   Msg = Msg + "The TestStr returned after stripping the spaces "

Msg = Msg + "is """ + TestStr1 + """ and it contains only "

Msg = Msg + STR$(Len(TestStr1)) + " characters."

MsgBox Msg                           ' Display message.
```

Mid (function) — Return a portion of a string

`rc$ = Mid$ (string$, position%[, length%])`

Returns a substring of a specified length from a source string, starting with the character at the specified position.

Parameter	**Description**
rc$	The return value.
string$	The string variable containing the substring you want to extract.
position%	The offset into the *string*$ where the substring begins.
length%	The number of characters in the substring.

Comments

If the *length*% argument is omitted, or if there are fewer characters in a string than specified in *length*%, **Mid$** will return all the characters from *position*% to the end of the string. If *position*% is larger than the number of characters in the indicated string, **Mid$** returns a Null string.

A value of 1 for *position*% indicates the first character in a string.

To modify a portion of a string value, refer to the **Mid** statement.

The dollar sign (**$**) in the function name is optional. If the dollar sign is specified, the return type is String. If the dollar sign is omitted, **Mid** will return a variant of vartype 8 (String). If the value of *string*$ is Null, a variant of vartype 1 (Null) is returned.

Mid (function) Example

The example uses the Mid function to return the middle word from a variable containing three words.

```
Title$ = "Mid$ Function Demo"


'Return the word 'Function' from Title$.
SpcPos1% = InStr(1, Title$, Chr$(32))           ' Find 1st space.
```

```
SpcPos2% = InStr(SpcPos1% + 1, Title$, Chr$(32)) ' Find 2nd space.
WordLen% = (SpcPos2% - SpcPos1%) - 1             ' Calculate word
                                                 ' length.
MidWord$ = Mid$(Title$, SpcPos1% + 1, WordLen%)

Msg$ = "The word in the middle of Title$ is """ + MidWord$ + """"
MsgBox Msg$, 0, Title$                          ' Display message.
```

Mid (statement) — Replace a portion of a string

Mid (*string***$**, *position*%[, *length*%] **)** = *subst-string***$**

Replaces the specified substring in a string with a substitute string.

Parameter	**Description**
*string***$**	The string variable you want to modify.
*position***%**	The offset into *string***$** where you want the replacement text to begin.
*length***%**	The number of characters to replace.
*subst-string***$**	The string or string expression that replaces the portion of *string***$** beginning at *position***%**.

Comments

If the *length***%** argument is omitted, or if there are fewer characters in *string***$** than specified in *length***%**, **Mid** will replace all the characters from *position***%** to the end of the string. If *position***%** is larger than the number of characters in *string***$**, **Mid** appends *subst-string***%** to *string***$**.

A value of 1 for *position***%** indicates the first character in a string.

Mid (statement) Example

This example uses the Mid statement to replace the last name in a user-entered string with 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
```

Minute (function) — Return the minute portion of a date/time value

rc = **Minute(***expression***)**

Returns an integer that represents the minute of the day derived from the supplied string or numeric expression.

Parameter	Description
rc	The return value (an integer between 0 and 59, inclusive).
expression	Any type of expression (numeric or string) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Numbers to the left of the decimal point are interpreted as a date; numbers to the right are interpreted as time values. Negative numbers are interpreted as dates prior to December 30, 1899.

Comments

If expression is Null, **Minute** returns a Null.

Minute (function) 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
```

MkDir (statement) — Create a new directory

`MkDir` *`pathname`***`$`**

Creates a new directory.

Parameter	**Description**

*pathname***$**	A string expression identifying the new directory name. The syntax for *pathname***$** is:
	
	`[`*`drive`*`:] [\]`
	`directory``[\`*`directory`*
	`[\`*`directory...`*`]]`
	
	The *drive* argument is optional. If omitted, **MkDir** makes a new directory on the current drive. The *directory* argument is a directory name.
	
	*pathname***$** is limited to 128 characters.

Comments

MkDir is equivalent to the mkdir operating system command, however, **MkDir** cannot be abbreviated.

MkDir (statement) Example

This example uses the MkDir statement to create a \TMP subdirectory off the root directory of the currently logged drive.

```
On Error Resume Next                   ' Set up error handling.


CurDrv$ = Left$(CurDir$, 2)            ' Get current drive letter.

TmpPath$ = UCase$(CurDrv$ + "\tmp")    ' Make path specification.

TmpPath$ = UCase$(CurDrv$ + "\__GG__") ' Make path specification.


MkDir TmpPath$                         ' Make new directory.

If Err = 75 Then                       ' Check if directory existed.

   Msg$ = TmpPath$ + " directory already existed."

Else
```

```
   Msg$ = TmpPath$ + " directory created."
End If

Msg$ = Msg$ + " Do you want it removed ?"
Answer% = MsgBox(Msg$, 4)              ' Display message and get
If Answer% <> 7 Then RmDir TmpPath$    ' user response.

RmDir TmpPath$
```

Month (function) — Return month from a date/time value

rc = **Month(***expression***)**

Returns an integer that represents the month of the year derived from the supplied string or numeric expression.

Parameter	**Description**
rc	The return value (an integer between 1 and 12, inclusive).
expression	Any type of expression (numeric or string) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Numbers to the left of the decimal point are interpreted as a date; numbers to the right are interpreted as time values. Negative numbers are interpreted as dates prior to December 30, 1899.

Comments

The return value is a variant of vartype 2 (Integer).

If expression is Null, **Month** returns a Null.

Month (function) 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
```

MsgBox (function) — Display a message that solicits user input

```
rc% = MsgBox( message$[,type%[, caption$]])
```

Displays a message in a dialog box and waits for the user to choose a button. This function returns an integer value indicating which button the user selected.

This function can also be used as a Statement.

Parameter	**Description**
*rc***%**	The return value.
*message***$**	The message that is displayed in the dialog box. *message***$** must be no more than 1024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.
*type***%**	Determines which icons, buttons, and button defaults will be displayed in the dialog box. This parameter is the sum of the desired buttons, icons, and defaults as follows: (Choose at least one value from each group.)

Buttons	**Value**
OK only	0
OK, Cancel	1
Abort, Retry, Ignore	2
Yes, No, Cancel	3
Yes, No	4
Retry, Cancel	5

Icons	**Value**
Critical Message (Stop sign)	16
Warning Query (Question mark)	32
Warning Message (Exclamation)	48
Information Message (lower-case i)	64

Parameter	Description	
*type***%** (continued)	**Button Defaults**	**Value**
	First button	0
	Second button	256
	Third button	512
	If *type***%** is omitted, a single OK button will be displayed	
*caption***$**	The string expression that will appear in the dialog box's title bar.	

Comments

Once the user has selected a button, the **MsgBox** function 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

MsgBox (function) 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 = " & btngrp & Chr(10)
      msgtext=msgtext + "   Default group = " & defgrp & Chr(10)
      msgtext=msgtext + Chr(10) + "   Continue?"
      answer=MsgBox(msgtext, btngrp+icongrp+defgrp)
      Select Case answer
        Case 2,3,7
          Exit Do
      End Select
      If icongrp<>64 then
         icongrp=icongrp+16
      End If
      btngrp=btngrp+1
   Loop
End Sub
```

MsgBox (statement) — Display a message that solicits user input

```
MsgBox message$[,[type%][, caption$]])
```

Displays a message in a dialog box and waits for the user to choose a button. This function does not return a value indicating which button the user selected. Use the MsgBox function if you need to capture the user response.

Parameter	**Description**
message$	The message that is displayed in the dialog box. *message*$ must be no more than 1024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.
type%	Determines which icons, buttons, and button defaults will be displayed in the dialog box. This parameter is the sum of the desired buttons, icons, and defaults as follows: (Choose at least one value from each group.)

Buttons	**Value**
OK only	0
OK, Cancel	1
Abort, Retry, Ignore	2
Yes, No, Cancel	3
Yes, No	4
Retry, Cancel	5

Icons	**Value**
Critical Message (Stop sign)	16
Warning Query (Question mark)	32
Warning Message (Exclamation)	48
Information Message (lower-case i)	64

Parameter	Description	
	Button Defaults	**Value**
	First button	0
	Second button	256
	Third button	512
*caption***$**	The string expression that will appear in the dialog box's title bar.	

MsgBox (statement) 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
```

Name (statement) — Rename a file

Name *oldfilename***$ As** *newfilename***$**

Renames a file. Can also be used to move a file from one directory to another.

Parameter	**Description**
*oldfilename***$**	A string expression that designates the name of the file that you want to rename. *oldfilename***$** may contain a path.
*newfile name***$**	A string expression that designates the new filename. *newfilename***$** may contain a path.

Comments

If *oldfilename***$** is currently open, EXTRA! Basic generates an error message. If the file *newfilename***$** already exists, EXTRA! Basic will generate an error message.

The **Name** statement does not produce a copy of the original file--after the **Name** statement has been executed only one file will exist and it will be named *newfilename***$**.

Name (statement) Example

The example moves a file from one directory to another and renames it at the same time.

```
NL$ = Chr$(13) + Chr$(10)                 ' Define newline.


FName1$ = "NMSTMTX.DAT"                   ' Define filenames.

FName2$ = "NMSTMTY.DAT"

TestDir$ = "\TEST.DIR"                    ' Test directory name.

Open FName1$ For Output As #1             ' Create a test file.

Print #1, "test data"                     ' Put something in file.

Close #1


MkDir TestDir$                            ' Make test directory.
```

```
Name FName1$ As TestDir$ + "\" + FName2$  ' Move and rename.

Msg$ = "A new file, " + FName1$ + " has been created "
Msg$ = Msg$ + "in " + CurDir$ + ". Once created, it was "
Msg$ = Msg$ + "moved to " + TestDir$ + " and renamed "
Msg$ = Msg$ + FName2$ + "." + NL$ + NL$
Msg$ = Msg$ + "Choose OK to remove the test data file and "
Msg$ = Msg$ + "directory."
MsgBox Msg$                               ' Display message.

Kill TestDir$ + "\" + FName2$             ' Remove file from disk.
RmDir TestDir$                            ' Remove test directory.
```

New (reserved word) — Create a new instance of an object

```
Set objectVar = New className
```

or

```
Dim objectVar As New className
```

Creates and initializes a new object of the specified class.

Parameter	**Description**
objectVar	The name of the variable that references the object you want to instantiate.

Comments

Using the first syntax shown above (the **Set** statement), **New** creates and initializes a new object of the named class.

Using the second syntax shown above (the **Dim** statement), **New** marks the object variable so that a new object will be created and initialized when the object variable referenced for the first time. If the object variable is not referenced, no new object will be created.

Note: An object variable that was declared with **New** will allocate a second object if the variable is **Set** to **Nothing** and referenced subsequently.

NoCStrings (metacommand) — Treat backslash as a normal character

```
'$NOCSTRINGS [SAVE]
```

Instructs the compiler to treat a backslash inside a string as a normal character. This is the default.

Parameter	**Description**
SAVE	This option saves the current CStrings setting before CStrings are disabled. The CSTRINGS RESTORE command will restore a setting saved previously. SAVE and RESTORE function as a stack and allow you to change the CStrings setting for a range of the macro without impacting the rest of the macro.

Comments

Use the **$CStrings** metacommand to instruct the compiler to treat a backslash as an escape character.

NoCStrings (metacommand) 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
```

Nothing (reserved word) — Set the value of an object variable to nothing

```
Set variableName = Nothing
```

Assigns the value **Nothing** to an object variable freeing memory and system resources when it is no longer being used.

Parameter	**Description**
variable Name	An object variable name.

Comments

Using the **Set** statement to assign the value **Nothing** to an object variable is equivalent to the following:

- The value of an object the first time it is declared (using the **Dim** statement)
- The value of a new instantiation of an object (using the **Dim** statement and the **New** keyword)

Note that local object variables are automatically set to **Nothing** at the end of the procedure; global or static object variables are not.

Nothing (reserved word) 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
```

Now (function) — Return the current date and time

```
rc = Now()
```

Returns the current date and time derived from the system clock settings.

Parameter	**Description**
rc	The return value.

Comments

The **Now** function returns a variant of vartype 7 (Date) (stored as a double-precision number). Valid dates range from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, is returned if the system date is set to ‘‘January 1, 1900’‘. Numbers to the left of the decimal point represent time; numbers to the right represent the date.

Now (function) 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
```

NPV (function) — Return net present value

rc = **NPV(***rate*, *valuearray***())**

Returns the net present value of an investment based on a variable stream of periodic future cash flows and a constant interest rate (also referred to as the discount rate).

Parameter	Description
rc	The return value.
rate	Discount rate per period in decimal notation. If the discount rate is 12% per period, for example, the *rate* parameter is 0.12.
valuearray( )	An array containing the cash flow values. A positive value is interpreted as a receipt and a negative value is interpreted as a payment. The array must contain at least on positive value and one negative value.

Comments

Note: All payments and receipts must be included in *valuearray* in the correct sequence. The **NPV** return value will vary changes in the sequence of cash flows.

Because **NPV** calculation is based on future cash flows, if the first cash flow occurs at the beginning of the first period, its value must not be included in *valuearray*. Instead it should be added to the result returned by **NPV**.

Payments (negative values) supplied in *valuearray* must be due at the end of the period.

NPV (function) 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
```

Null (function) — Return a variant set to Null

`variableName = Null`

Explicitly sets a variant value to null.

Parameter	**Description**
variableName	The name of the variable you want to set to the null value.

Comments

Note that variants are initialized to the Empty value, by default.

Null (function) 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
```

Object (reserved word) — Declare an OLE2 object

Dim *variableName* **As Object**

Object is a class that provides access to OLE2 automation objects.

Parameter	Description
variable Name	The name of the variable you want to declare as belonging to the class Object.

Comments

To create a new Object, first dimension a variable, and then **Set** the variable to the return value of **CreateObject** or **GetObject** as shown below:

```
Dim ole2 As Object
Set ole2 = CreateObject("spoly.cpoly")
ole2.reset
```

Object (reserved word) 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
```

Oct (function) — Return the octal value of a number

rc$ = **Oct$(***numeric-expression***)**

Returns an octal representation of a numeric-expression as a string.

Parameter	Description
rc$	The return value.
numeric-expression	Any numeric data type. The result of *numeric-expression* is rounded to the nearest whole number before being evaluated.

Comments

If the numeric expression is an integer, the string will contain up to six octal digits; otherwise, the expression will be converted to a Long integer, and the string will contain up to 11 octal digits.

Oct (function) Example

This example prints the octal values for the numbers from 1 to 15.

```
Sub main
   Dim x,y
   Dim msgtext
   Dim nofspaces
   msgtext="Octal numbers from 1 to 15:" & Chr(10)
   For x=1 to 15
     nofspaces=10
     y=Oct(x)
     If Len(x)=2 then
        nofspaces=nofspaces-2
     End If
     msgtext=msgtext & Chr(10) & x & Space(nofspaces) & y
   Next x
   MsgBox msgtext
End Sub
```

OkButton (statement) — Specify position and size of an OK button

```
OkButton x, y, dx, dy [.id]
```

Determines the position and size of an OK button.

Parameter	**Description**
x, y	*x* and *y* specify the position of the OK button relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the OK button is centered horizontally within the client area. If *y* is omitted, the list box is centered vertically within the client area.
dx, dy	*dx* and *dy* set the width and height of the button. A *dy* value of 14 typically accommodates text in the system font.
.id	An optional identifier used by the dialog statements that act on this control.

Comments

The **OkButton** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

OkButton (statement) Example

This example defines a dialog box with a dropcombo 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, "E! Basic 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
```

On Error (statement) — Activate an error-handling routine

```
On [Local] Error {GoTo label [ Resume Next ] GoTo 0}
```

Activates the specified error-handling routine. Control is passed to this routine in the event of a run-time error.

Parameter	Description
label	The line label of the error-handling routine.

Comments

On Error can also be used to disable an error-handling routine.

Unless an **On Error** statement is used to control error handling, any run-time error will be fatal (that is, EXTRA! Basic will terminate the execution of the program).

The **On Error** statement uses the following reserved words:

Reserved Word	Definition
Local	Used to ensure compatibility with other variants of EXTRA! Basic. It is allowed in error-handling routines only at the procedure level.
GoTo	Enables the error-handling routine that begins at the specified label. If the designated label cannot be found in the same procedure as the **On Error** statement, EXTRA! Basic will generate an error message.
Resume Next	Passes control to the statement immediately following the statement in which the error occurred.
GoTo 0	Disables any error handler that has been enabled.

The error handler remains active from the time the run-time error has been trapped until a **Resume** statement is executed within the error handler. After an error has been handled, program execution resumes in the current procedure at the statement indicated in the **Resume** statement.

If another error occurs while the error handler is active, EXTRA! Basic searches for an error handler within the calling procedure (and then in the calling procedure’s calling procedure, if none can be found). When an error handler is found, the current procedure terminates, and the error handler is activated. If EXTRA! Basic cannot find an inactive error handler, a fatal error occurs. This fatal error can be traced back to the original error location.

When control is passed back to a calling procedure, that procedure becomes the current procedure.

An error occurs if an **End Sub** or **End Function** statement is executed 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.

Use the **Err** and **Error$** function to retrieve the run-time error code and error message.

On Error (statement) Example

This example uses the On Error statement to set up error handling in a procedure.

```
On Error GoTo ErrorHandler                    ' Set up error handler.
Msg$ = "This demo attempts to open a file that does not exist on a "
Msg$ = Msg$ + "drive that may not exist. When the operation fails, "
Msg$ = Msg$ + "the ErrorHandler routine will display a message box "
Msg$ = Msg$ + "that indicates what error occurred."
MsgBox Msg$                                  ' Display opening message.

'Generate random drive letter.
Try-again:
Randomize
Drive$ = Chr$(Int((26) * Rnd + 1) + 64)

If Drive$ = "A" or Drive$ = "B"  then goto Try-again

Open Drive$ + ":\TEST\X.DAT" For Input As #1 ' Try to open file.
```

```
Close #1                                     ' Close the file.
Exit Sub                                     ' Exit before entering
                                             ' error handler.
ErrorHandler:                             ' Error handler line label.
Select Case Err
    Case 53: Msg$ = "ERROR 53: That file doesn't exist."
    Case 68: Msg$ = "ERROR 68: Drive " + Drive$ + ": not available."
    Case 76: Msg$ = "ERROR 76: That path doesn't exist."
    Case Else: Msg$ = "ERROR " + Str$(Err) + " occurred."
End Select

MsgBox Msg$                                  ' Display error message.
Print Msg$

Resume Next                                  ' Resume procedure.
```

On Goto (statement) — Branch to a macro line label

On *numeric-expression* **GoTo** *branchlist*

Branch to the line with the specified label.

Parameter	**Description**
numeric-expression	A numeric expression that evaluates to a number (rounded to the nearest integer) between 0 and 255 (inclusive). If *numeric-expression* evaluates to 1, control branches to the first label in *branchlist*. If *numeric-expression* evaluates to 2, control branches to the second label in *branchlist,* and so forth.
branchlist	A comma separated list of line labels or line numbers.

Comments

If *numeric-expression* evaluates to 0 or to a number greater than the number of labels or line numbers in *branchlist,* macro control jumps to the next statement. If *numeric-expression* evaluates to a number less than 0 or greater than 255, an ‘‘Illegal function call’‘ error is issued.

On Goto (statement) 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
```

Open (statement) — Opens a file or a device

```
Open filename$ [For mode] [Access access] [lock] _
As [#] filenumber% [Len = reclen]
```

Enables I/O to a file or a device.

Parameter	**Description**
filename$	A string or string expression that names the file or device you want to open. *filename*$ can include a path, if necessary.
mode	Mode must be one of the following reserved words:
	Reserved Word / **Description**
	Append — Indicates sequential output mode. Append sets the file pointer to the end of the file so that subsequent **Print #** or **Write #** statements, in effect, are appended to *filename*$.
	Input — Indicates sequential input mode.
	Output — Indicates sequential output mode.
filenumber%	An integer expression that evaluates to between 1 and 255, inclusive. When an **Open** statement is executed, *filenumber*% is assigned to *filename*$ and remains assigned until *filename*$ is closed. Use the **FreeFile** function to return the next available filenumber.

Comments

A file must be opened before any input/output operation can be performed on it.

Open (statement) Example

This example uses the Open statement to open a file for output.

```
TestString$ = "The quick brown fox"     ' Create test string.
For I% = 1 To 5
   FNum% = FreeFile                     ' Determine next file number.
   FName$ = "FILEIOX" + LTrim$(Str$(FNum%)) + ".DAT"
   Open FName$ For Output As FNum%      ' Open file.
   Print #I%, TestString$               ' Write string to file.
Next I%
Close                                   ' Close all files.
Msg$ = "Several test files have been created on your disk. "
Msg$ = Msg$ + "Choose OK to remove the test files."
MsgBox Msg$
Kill "FILEIOX?.DAT"                     ' Remove test file from disk.
```

Option Base (statement) — Specify default lower bound for array Subscripts

Option Base *lowerBound%*

Specifies the default lower bound for array subscripts. This statement is not required in order to declare the dimensions of an array.

Parameter	**Description**
lower Bound%	*lowerBound*% must be either 0 or 1.

Comments

If the **Option Base** statement is not specified, the default lower bound for array subscripts will be 0.

The **Option Base** statement must appear in the Declarations section of a module, and must precede any use of any declared arrays. Only one **Option Base** statement is permitted per module.

Option Base (statement) Example

This example specifies the lower bound to be used for array A is set to 1. 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.

```
Option Base 1                          ' Stays in Declarations section.
Sub Main
   NL$ = Chr$(13) + Chr$(10)           ' Define newline.
   ReDim A(20)                         ' Create an array.
   Msg$ = "The lower bound of the A array is" + Str$(LBound(A)) + "."
   Msg$ = Msg$ + NL$ + "The upper bound is" + Str$(UBound(A)) + "."
   MsgBox Msg$                         ' Display message.
End Sub
```

OptionButton (statement) — Define option buttons

```
OptionButton x, y, dx, dy, text$ [, .id]
```

Defines the position and text associated with an option button. There must be at least two OptionButton statements and they must be used in conjunction with the OptionGroup statement.

Parameter	**Description**
x, y	*x* and *y* specify the position of the option button relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the option button is centered horizontally within the client area. If *y* is omitted, the option button is centered vertically within the client area.
dx, dy	The width of the option button. The *dy* argument is the height of the *text***$** parameter.
*text***$**	Contains the text that will be displayed to the right of the option button. To indicate an accelerator key in *text***$**, precede the accelerator key character with an ampersand (**&**).

Comments

The **OptionButton** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

OptionButton (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, "E! Basic 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
```

Option Compare (statement) — Specify the default string comparison method

`Option Compare {Binary | Text}`

Specifies the default method of string comparison.

Parameter	**Description**
Binary	Perform string comparisons based on the ANSI character set.
Text	Perform string comparisons based on the relative order of characters determined by the country code specified in the International section of the Windows WIN.INI file.

Comments

Binary comparisons are case sensitive; text comparisons are case-insensitive.

If your macro does not specify an **Option Compare** statement, **Binary** string comparisons will be performed, by default.

Option Compare (statement) Example

This example compares two strings: ‘‘MARIA SMITH‘‘ and ‘‘maria 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 retvalue as Integer

   strg1="MARIA SMITH"

   strg2="maria smith"

i:

   retvalue=StrComp(strg1,strg2)
   If retvalue=0 then
      MsgBox "The strings are identical"
   Else
      MsgBox "The strings are not identical"
      Exit Sub
   End If
End Sub
```

Option Explicit (statement) — Force all variables to be declared explicitly

`Option Explicit`

Specifies that all variables must be declared explicitly. Without an **OptionExplicit** statement, any variables that do not appear in a **Dim**, **Global**, **ReDim**, or **Static** statement will be implicitly declared as a variant data type (unless a data type symbol or a **Def*Type*** statement is provided).

Comments

If an **OptionExplicit** statement is included in your macro and you attempt to use an undeclared variable name, EXTRA! Basic returns a ‘‘Variable not declared’‘ error message.

Option Explicit (statement) 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
```

OptionGroup (statement) — Name user selection variable

```
OptionGroup .field
```

Used in conjunction with **OptionButton** statements to set up a series of related options. The OptionGroup statement also establishes the dialog-record field that will contain the user's selection.

Parameter	**Description**
.field	The name of the variable that will hold the result of the user's selection. *field* is set to 0 when the choice associated with the first **OptionButton** statement is selected, or a value of 1 when the choice associated with the second **OptionButton** statement is chosen, and so forth.

Comments

The **OptionGroup** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

PasswordBox (function) — Return a password entered by the user

```
rc = PasswordBox[$](prompt$[, [title$] [,[default$] _
    [,xpos%. ypos%]]])
```

Returns a password entered by the user.

Parameter	**Description**
rc	The return value.
prompt$	A string expression (up to approximately 255 characters long depending on the width of the font) containing the text to be displayed in the dialog box. Multiple-line prompts must contain carriage return and line feed characters.
title$	A string expression containing the caption that will appear in the dialog box's title bar. If *title* is not specified, the title bar will be blank.
default$	A string expression containing the text to which the edit box will be initialized. If *default* is not specified, the edit box will be blank.
xpos%, *ypos*%	Numeric expressions, specified in dialog box units, that determine the position of the password box. *xpos*% determines the horizontal distance between the left edge of the screen and the left border of the input box. *ypos*% determines the horizontal distance from the top of the screen to the input box's upper edge. If one or both of these arguments are omitted, the password box is centered roughly one third of the way down the screen.

Comments

When the user presses Enter or selects the OK button, **PasswordBox** returns the text that the user entered in the text box. If the user selects the Cancel button, **PasswordBox** returns a Null string.

PasswordBox (function) 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
```

Picture (statement) — Define a picture control

```
Picture x, y, dx, dy, filename$, type [, .id]
```

Defines a picture control in a custom dialog box.

Parameter	**Description**
x, y, dx, dy	*x* and *y* determine the position of the picture relative to the upper left corner of the dialog box. *dx* and *dy* set the width and height of the picture. The picture will be scaled equally in both directions and centered if the dimensions of the picture (*x* and *y*) are not proportional to the height and width (*dx* and *dy*).
filename$	Specifies the name of the bitmap file (.BMP) containing the picture. Note that if *type* is 3 (indicating that the picture should be copied from the Clipboard), *filename* is ignored.
type	Specifies the bitmap type. Valid *type* values are 0, 3, 16, and 19. A *type* of 0 indicates that the bitmap should be taken from *filename*. A *type* of 3 indicates that the bitmap should be copied from the Clipboard.
.id	An optional identifier used by the dialog statements that act on this picture control.

Comments

The **Picture** statement can only be used within a **Begin Dialog...End Dialog** statement.

If the picture is not available (if *filename* does not exist or does not contain a bitmap, or if there is no bitmap in the Clipboard), the picture control will display the picture frame and the text ‘‘Missing Picture.‘‘ To trigger a run-time error instead, add 16 to the value of *type*.

Picture (statement) Example

This example defines a dialog box with a picture, and the OK and Cancel buttons.

```
Sub main
   Begin Dialog UserDialog 148, 73, "E! Basic Dialog Box"
      Picture  8, 7, 46, 46, "C:\WINDOWS\ARCADE.BMP", 0
      OKButton  80, 10, 54, 14
      CancelButton  80, 30, 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
```

Pmt (function) — Return a constant periodic payment

```
rc = Pmt (rate, nper, pv, fv, due)
```

Returns a constant periodic payment amount for a series of constant cash payments made over a period of time (an annuity; a loan or investment).

Parameter	**Description**
rc	The return value.
rate	The interest rate per period. If payments are on a monthly schedule, *rate* would be .0075 for an annual percentage rate (APR) of 9% (0.9 / 12 = 0.0075). *rate* is assumed to be constant over the life of the annuity.
nper	The total number of payment periods. *nper* for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pv	The present value of the initial lump sum amount paid (as in an annuity) or received (as in a loan).
fv	The future value of the final lump sum amount required (as in a savings plan) or paid (0, in the case of a loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.

Comments

rate and *nper* must be calculated using the same payment period unit of measurement (e.g., months or days).

For all arguments, negative values indicate cash paid out (as in a savings account deposit) and positive values indicate cash received (as in a dividend check).

Pmt (function) Example

This example finds the monthly payment on a given loan.

```
Sub main
   Dim aprate, totalpay
   Dim loanpv, loanfv
   Dim due, monthlypay
   Dim yearlypay, 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
```

PPmt (function) — Return the principal portion of a payment value

```
rc = PPmt (rate, per, nper, pv, fv, due)
```

Returns the principal portion of the payment for a given period of an annuity (a loan or investment).

Parameter	**Description**
rc	The return value.
rate	The interest rate per period. If payments are on a monthly schedule, *rate* would be .0075 for an annual percentage rate (APR) of 9% (0.9 / 12 = 0.0075). *rate* is assumed to be constant over the life of the annuity.
per	The number of the particular payment period for which you want to determine the principal. *per* must be in the range 1 through *nper*.
nper	The total number of payment periods. *nper* for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pv	The present value of the initial lump sum amount paid (as in an annuity) or received (as in a loan).
fv	The future value of the final lump sum amount required (as in a savings plan) or paid (0, in the case of a loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.

Comments

rate and *nper* must be calculated using the same payment period unit of measurement (e.g., months or days).

For all arguments, negative values indicate cash paid out (as in a savings account deposit) and positive values indicate cash received (as in a dividend check).

PPmt (function) 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
```

Print (statement) — Write data to a file

```
Print [# filenumber%,] expressionlist [{;|,}]
```

Outputs data to the sequential file indicated by *filenumber***%**.

Parameter	**Description**
# *filenumber***%**	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number. *filenumber***%** is optional. If it is omitted, the **Print #** statement outputs data to the screen in a message box.
expression list	Contains the values that are printed. *expressionlist* can contain numeric or string expressions. Expressions 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 an intervening space. A comma indicates that the next value should be positioned at the next print zone. Print zones begin every 14 spaces. If neither separator is specified, a Carriage Return and Line Feed will be generated and the next **Print #** statement will print on the next line, or display in the next message box.

Comments

If *expressionlist* is omitted, a blank line is written to the file. At the minimum, a comma must be included.

Note that monospaced text screen characters printed to a graphical environment using proportionally spaced characters may lose some degree of readability.

The Print statement supports only elementary EXTRA! Basic data types. See the **Input** statement description for more information on parsing this statement.

Print (statement) Example

This example uses the Print # statement to write data to a test file.

```
Dim FileData$, Msg$, NL$                ' Declare variables.
NL = Chr(10)                            ' Define newline.

Open "TESTFILE" For Output As #1        ' Open to write file.
Print #1, "This is a test of the Print # statement."
Print #1,                               ' Print blank line to file.
Print #1, "Zone 1", "Zone 2"            ' Print in two print zones.
Print #1, "With no space between" ; "."  ' Print two strings
                                        ' together.
Close #1                                ' Close file.

Open "TESTFILE" for Input As #1         ' Open to read file.
Do While Not EOF(1)
   Line Input #1, FileData              ' Read a line of data.
   Msg = Msg + FileData + NL            ' Construct message.
Loop
Close #1                                ' Close file.

MsgBox Msg                              ' Display message.
Kill "TESTFILE"                         ' Remove file from disk.
```

Put (statement) — Write from a variable to an open file

```
Put [#] filenumber%, [recordnumber&], variable
```

Writes a variable to a file opened in Binary or Random mode.

Parameter	**Description**
filenumber%	An integer or integer expression that evaluates to the number of an open file.
record number&	A Long expression containing the number of the record (for Random mode) or the offset of the byte (for Binary mode) at which to begin writing.
variable	The name of the variable from which the corresponding **Get** statement copied file data. Note that an array variable cannot be specified; however, an element within an array can.

Comments

For files opened in **Random** mode, the following rules apply:

- Data is written to the file in blocks equal to the size specified in the **Len** clause of the **Open** statement. If the size of *variable* 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 String Variants 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 is written separately, however, no padding occurs between elements.

For files opened in **Binary** mode, the following rules apply:

- **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.

Put (statement) 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
```

PV (function) — Return present value

```
rc = PV (rate, nper, pmt, fv, due)
```

Returns the present value of a constant periodic stream of cash flows (annuity).

Parameter	**Description**
rc	The return value.
rate	The interest rate per period. If payments are on a monthly schedule, *rate* would be .0075 for an annual percentage rate (APR) of 9% (0.9 / 12 = 0.0075). *rate* is assumed to be constant over the life of the annuity.
nper	The total number of payment periods. *nper* for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pmt	The constant periodic payment per period.
fv	The future value of the final lump sum amount required (as in a savings plan) or paid (0, in the case of a loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.

Comments

rate and *nper* must be calculated using the same payment period unit of measurement (e.g., months or days).

For all arguments, negative values indicate cash paid out (as in a savings account deposit) and positive values indicate cash received (as in a dividend check).

PV (function) 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
```

Randomize (statement) — Initialize the random number generator

```
Randomize [numeric-expression%]
```

Seeds the random number generator.

Parameter	**Description**
numeric-expression%	An integer value between -32768 and 32767 and can be of any numeric data type.

Comments

If *numeric-expression*% is omitted, EXTRA! Basic uses the Timer function to initialize the random number generator.

If the **Randomize** statement is not used prior to the **Rnd** function, or if **Randomize** is executed multiple times with the same *numeric-expression*%, the Rnd funtion will return the same sequence of random numbers each time the program is run. To generate different random numbers each time you run a particular program, place a **Randomize** statement at the beginning of the program and omit the *numeric-expression*% parameter.

Randomize (statement) 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
```

Rate (function) — Return interest rate per period

```
rc = Rate (nper, pmt, pv, fv, due, guess)
```

Returns the interest rate per period for an annuity.

Parameter	**Description**
rc	The return value.
nper	The total number of payment periods. *nper* for a 3-year car loan, for example, would be 36 (3 * 12 = 36).
pmt	The constant periodic payment per period.
pv	The present value of the initial lump sum amount paid (as in an annuity) or received (as in a loan).
fv	The future value of the final lump sum amount required (as in a savings plan) or paid (0, in the case of a loan).
due	0 if payments are due at the end of each payment period; 1 if they are due at the beginning of the period.
guess	A ballpark estimate of the value that will be returned by Rate. In general, between 0.1 (10%) and 0.15 (15%) is a reasonable *guess*.

Comments

For all arguments, negative values indicate cash paid out (as in a savings account deposit) and positive values indicate cash received (as in a dividend check).

Rate is an iterative function. EXTRA! Basic cycles through a **Rate** calculation until the result is accurate to within 0.00001 percent. If **Rate** does not produce a result after 20 iterations, it fails.

Rate (function) 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
```

ReDim (statement) — Redefine the subscript range of an array

```
ReDim variableName (subscriptRange,...)[As [New]
type],...
```

Declares a dynamic-array variable at the procedure level. Also used to size or resize a dynamic array that has already been formally declared using a **Global** or **Dim** statement.

Parameter	Description
variable Name	The name of the variable you want to declare.
subscript Range	The dimensions of the array. *subscriptRange* uses the following format: `[ startSubscript To ] endSubscript` where *startSubscript* is the lower bound and *endSubscript* is the upper bound of the array variable's subscripts. If *startSubscript* is not specified, it is set to zero (0). Use the **Option Base** statement to change this default.
type	The data type of the variable. (*type* must be one of the following reserved words: **Any**, **Integer**, **Long**, **Single**, **Double**, or **String**.)

Comments

Memory for the dynamic array will be reallocated to support the specified dimensions, and the array elements will be reinitialized. **ReDim** cannot be used at the module level; it must be used inside a procedure.

A dynamic array is normally created by using **Dim** to declare an array without a specified *subscriptRange*. The maximum number of array dimensions you can specify with **ReDim** for a dynamic array created in this fashion is eight. If you need more than eight dimensions, use the **ReDim** statement to declare a local array that has not previously been declared using **Dim** or **Global**. In this case, the maximum number of dimensions is 60. (The amount of memory used by a numeric array is calculated by multiplying the number of elements in the array by the number of bytes required by the data type of the array.) If you specify a subscript that is greater than the maximum or smaller than the minimum, or if the size of the array exceeds the allowable limit, EXTRA! generates an error message.

If the **As** clause is not used, the type of the variable can be specified using a type character suffix. The two different type-specification methods can be intermixed in a single **ReDim** statement (although not for the same variable).

Note that you can use the **ReDim** statement repeatedly to change the *number of elements* in an array. However, it cannot be used to change the *number of dimensions* in an array once the array has been given dimensions. **ReDim** can be used to change the upper and lower bounds of the dimensions of the array. (Use **LBound** and **UBound** to determine the current bounds of an array variable's dimensions.) Similarly, you cannot use **ReDim** to change an array's data type.

Note: Try to avoid using **ReDim** on an array in a procedure that has already received a reference to an element in that array; the result is unpredictable.

ReDim (statement) Example

This example uses the ReDim statement to dynamically resize an array while a procedure is running.

```
Dim TestArray() As Integer       ' Declare dynamic array.
Dim Size As Integer              ' Declare integer variable.

Randomize
Size = Int(100 * Rnd + 1)        ' Generate random number.
ReDim TestArray(Size)            ' Make array have Size% elements.
For I% = 1 to Size               ' Index for number of elements.
   TestArray(I%) = Rnd           ' Put number in each element.
Next I%

ReDim TestArray(Size * 10)       ' Make array 10 times larger.
For I% = 1 to Size * 10          ' Index for number of elements.
```

```
   TestArray(I%) = Rnd           ' Put number in each element.
Next I%

ReDim TestArray(Size)            ' Shrink array to original size.
For I% = 1 to Size               ' Index for number of elements.
   TestArray(I%) = Rnd           ' Put number in each element.
Next I%

Msg$ = "This demo created a dynamic array of" + Str$(Size)
Msg$ = Msg$ + " elements and filled each with a random number. "

Msg$ = Msg$ + "It then resized the array to" + Str$(Size * 10)
Msg$ = Msg$ + " elements, again filling each with a random number. "
Msg$ = Msg$ + "Finally, it restored the array back to"
Msg$ = Msg$ + Str$(Size) + " elements."
MsgBox Msg$                      ' Display message.
```

Rem (statement) — Add a comment

```
Rem comments
```

Inserts a comment in an EXTRA! Basic program.

Parameter	**Description**
comments	Any explanatory text that documents how the code that precedes or follows the **Rem** statement works. Spaces and other punctuation are permitted

Comments

Everything from **Rem** to the end of the line is ignored.

If **Rem** does not start in column one, it must be separated from any preceding executable statements on the same line by a colon.

An apostrophe (') can also be used to initiate a comment. Metacommands (**$CStrings**, for example) must be preceded by the apostrophe comment form. If the apostrophe form is used and the comment is preceded by executable statements on the same line, a colon separator is not required.

Using line numbers or line labels, you can direct program control from a **GoTo** or **GoSub** statement to a line containing a **Rem** statement. Execution will continue with the first executable statement following the **Rem** statement.

Rem (statement) Example

This example illustrates the various forms of the Rem statement syntax.

```
Rem This is the first form of the syntax.

' This is the second form of the syntax.

Msg$ = "Hello" : Rem Comment after a statement; separated with colon.

MsgBox Msg$      ' Comment after a statement with no colon.
```

Reset (statement) — Close all open disk files

`Reset`

Closes all open disk files, and writes any data still remaining in the operating system buffers to disk.

Reset (statement) 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
```

Resume (statement) — Resume program execution

```
Resume Next
```

-or-

```
Resume label
```

-or-

```
Resume [ 0 ]
```

Resumes program execution at the conclusion of an error-handling routine.

Parameter	**Description**
label	A program line label. A label has the same format as any other EXTRA! Basic name. That is, it must begin with an alphabetic character, end with a colon, and be 40 characters, or fewer, in length. *label* must also be unique to the current module. Line labels are not case-sensitive. Reserved words are not valid labels.

Comments

When the **Resume Next** statement is used, control is passed to the statement immediately following the statement in which the error occurred.

When the **Resume** *label* statement is used, control is passed to the statement immediately following the specified label.

When the **Resume** [0] statement is used, control is passed to the statement in which the error occurred.

Note that the location of the error handling routine determines where execution will resume NOT the location of the error itself. 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, program control reverts to the statement that last called out the procedure containing the error handler.

An error will occur if you use a **Resume** statement outside an error-handling routine. An error will also occur if the end of the procedure is encountered before a **Resume** statement in an active error-handling routine—the EXTRA! Basic compiler interprets this as a logical error.

However, if an **Exit Sub** or **Exit** function statement is encountered while an error handler is active, no error occurs because it is considered a deliberate redirection of program flow.

Resume (statement) Example

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.

```
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
```

Right (function) — Return a substring

`rc$ = Right[$] (string$, length%)`

Returns a string of a specified length copied from the end of a source *string*. **Right$** counts backwards *length***$** characters from the last character in *string***$**.

Parameter	**Description**
*rc***$**	The return value.
*string***$**	The source string from which the substring will be returned.
*length***%**	A Long integer or Long integer expression that indicates how many characters **Right$** should return.

Comments

If the length of *string***$** is less than *length***%**, **Right$** returns the entire string.

Right (function) Example

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.

```
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
```

RmDir (statement) — Remove a directory

RmDir *pathname***$**

Removes a directory.

Parameter	**Description**
*pathname***$**	A string expression identifying the directory you want to remove. The syntax for *pathname***$** is: `[drive:] [\] directory[\directory [\directory...]]` The *drive* argument is optional. If omitted, **RmDir** searches the current drive for the specified directory. The *directory* argument is a directory name. *pathname***$** must contain fewer than 128 characters.

Comments

The directory you want to remove must not contain any user-created subdirectories or files.

RmDir (statement) Example

The example uses the RmDir statement to remove a previously created directory at the user's request.

```
On Error Resume Next                ' Set up error handling.


CurDrv$ = Left$(CurDir$, 2)         ' Get current drive letter.
TmpPath$ = CurDrv$ + "\__G_G__"     ' Make path specification.
MkDir TmpPath$                      ' Make new directory.


If Err \ 75 Then                    ' Check if directory existed.
   Msg$ = TmpPath$ + " directory already existed."
Else
```

```
   Msg$ = " Do you want it removed? "
Answer% = MsgBox(Msg$, 4)           ' Display message and get
If Answer$ <> 7 Then RmDir TmpPath$ ' user response.
```

```
RmDir TmpPath$
```

Rnd (function) — Return a random number

```
Rnd [(number!)]
```

Returns a single precision random number between 0 and 1.

Parameter	**Description**
number!	Any valid number or numeric expression. The value of *number*! determines the **Rnd** return value as follows:

If number!'s value is	**The return value is**
less than zero (0)	the same each time the **Rnd** function runs using that same *number*!
greater than zero (0)	the next random number in the sequence
equal to zero (0)	the previously generated random number
omitted	the next random number in the sequence. In the Absence of a previous random number, the random number seed is determined by the **Randomize** statement and / or the **Timer** function.

Comments

The same sequence of random numbers is generated whenever the program is run using the same *number*!, unless *number*! is reinitialized using the **Randomize** statement.

Use the following formula to generate random numbers that fall within a given range:

```
Int((upperbound - lowerbound + 1) * Rnd +
lowerbound)
```

where *upperbound* is the highest number in the range and *lowerbound* is the lowest number in the range. The possible random numbers produced include *upperbound* and *lowerbound.*

Rnd (function) Example

This example uses the Rnd function to simulate rolling a pair of dice by generating random values from 1 to 6. Each time this program is run, Randomize uses the Timer function to generate a new random-number sequence. The timer function is automatically used with the Randomize statement to generate a seed for the Rnd function.

```
Randomize                      ' Seed random number generator.
Dice1% = Int(6 * Rnd + 1)      ' Generate first die value.
Dice2% = Int(6 * Rnd + 1)      ' Generate second die value.
Msg$ = "You rolled a " + LTrim$(Str$(Dice1%))
Msg$ = Msg$ + " and a " + LTrim$(Str$(Dice2%))
Msg$ = Msg$ + " for a total of "
Msg$ = Msg$ + LTrim$(Str$(Dice1% + Dice2%)) + "."
MsgBox Msg$                    ' Display message.
```

Rset (statement) — Right-align a string expression

```
Rset string$ = string-expression
```

Right-aligns a string expression within a string.

Parameter	**Description**
string$	The name of a string variable.
string-expression	The string expression within *string* that is to be right-aligned.

Comments

If *string* is longer than *string-expression*, only the leftmost characters of *string-expression* are copied and the remaining characters are truncated.

Rset cannot be used to assign a user-defined variable type to a different user-defined variable type.

Rset (statement) 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
```

RTrim (function) — Remove trailing spaces

```
rc$ = RTrim[$](string$)
```

Returns a copy of a source string, with all trailing space characters removed.

Parameter	Description
*rc***$**	The return value.
*string***$**	Any string expression.

Comments

The dollar sign (**$**) in the function name is optional. If specified, the return type is String. If the dollar sign is 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.

RTrim (function) Example

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.

```
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
```

Second (function) — Return the seconds portion of a date/time value

`rc = Second(expression)`

Returns an integer (between 0 and 59, inclusive) that represents the second component of a date-time value.

Parameter	**Description**
rc	The return value (an integer between 0 and 59, inclusive).
expression	Any type of expression (numeric or string) that represents a date-time value. Numbers to the left of the decimal point are interpreted as a date; numbers to the right are interpreted as time values.

Comments

If expression is Null, **Second** returns a Null.

Second (function) 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
```

Seek (function) — Return current file position

`rc = Seek(filenumber%)`

Returns the current file position for the file specified by *filenumber***%**.

Parameter	**Description**
rc	The return value.
*filenumber***%**	The file number used in a previous **Open** statement or any integer expression that evaluates to that file number.

Comments

Seek returns a value between 1 and 2,147,483,647 inclusive that indicates the byte at which the next operation will take place. Byte positions are numbered sequentially beginning with the first byte in the file.

Seek (function) Example

The example uses the Seek statement to find the byte position in a sequential file.

```
'$Cstrings
sub main
Const STRLEN% = 20

Dim NameField As String * STRLEN%   ' Declare form variable.
Dim NameFields As String
Dim I%, Max%, Msg$                  ' Create sample data file.

Open "TSTFILE" For Output As #1
Length% = Len(NameField)            ' Get user input to fill records.
For I = 1 To 3
   NameField$ = InputBox$("Enter student name: ")
   Write #1,  NameField             ' Put record on disk.
Next I
Close #1                            ' Close data file.
```

```
Open "TSTFILE" For Input As #1      ' Open data file.
Max = LOF(1)                        ' Returns number of bytes in
                                    ' the file.
NumRecords% = LOF(1) / (STRLEN% + 4)
Counter% = 0

For I = 1 To Max Step STRLEN% + 4   ' Read file, step STRLEN + 4
                                    ' because of 2 quote marks
                                    ' and CR/LF.
   Counter% = Counter% + 1
   Seek 1, I + 1                    ' Seeks byte position for
                                    ' sequential files.
   Input #1, NameFields             ' Get record.
   Msg = "Record #" + " " + STR$(Counter%)
   Msg = Msg + " \" " + NameFields
   MsgBox Msg                       ' Display string.
Next I
Close #1                            ' Close file.

Msg = "Choose OK to remove the test file."
MsgBox Msg                          ' Display message.
Kill "TSTFILE"                      ' Remove file from disk.
```

Seek (statement) — Set position for file input/output

`Seek[#] filenumber%, position&`

Sets the position within a file for the next read or write operation.

Parameter	**Description**
*filenumber***%**	*filenumber***%** is the file number used in a previous **Open** statement or any integer expression that evaluates to that file number.
*position***&**	*position***&** is a number or numeric expression that evaluates to the location at which the next read or write should occur. The value of *position***&** must be between 1 and 2,147,483,647, inclusive.

Comments

If *position***&** points past the end of the file, subsequent writes extend the length of the file. EXTRA! Basic returns an error message if a **Seek** specifies a negative or zero *position***&**.

Seek (statement) 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
```

Select Case (statement) — Execute one of several statement blocks

```
Select Case testexpression
[Case expressionlist
   [statement-block] ]
[Case expressionlist
   [statement-block] ]
.
.
.
[Case Else
   [statement-block] ]
End Select
```

Executes one of a series of statement blocks, depending on the value of an expression.

Parameter	**Description**
test expression	Any numeric or string expression.

expression list	A comma-delimited list of expressions. These expressions can take one of the following forms:
	`expression [, expression [,...]]`
	-or-
	`expression [, expression [,...]] To expression [, expression [,...]]`
	-or-
	`Is comparison-operator expression [, expression [,...]]`
	where *comparison-operator* is: <, >, =, <=, >=, or <>.
	Note that the type of each expression must be compatible with the type of *testexpression*.

Parameter	**Description**
statement-block	Any number of EXTRA! Basic statements on one or more lines.

Comments

If *testexpression* matches the *expressionlist* associated with a **Case** clause, the *statementblock* following that **Case** clause is executed up to the next **Case** clause (or **End Select** statement, for the final **Case** clause). Control then passes to the statement following **End Select**. It is important to note that if *testexpression* matches more than one **Case** clause, only the statements following the first match are executed.

You can specify ranges and multiple expressions for character strings. In the following example, **Case** matches strings that are exactly equal to the string ‘‘Accounts Receivable’‘, strings that fall between A and L in alphabetical order, and the value of **CurrentRec$**:

```
Case "Accounts Receivable", "A" To "L", CurrentRec$
```

Select Case statements can also be nested. Remember to supply a matching **End Select** statement for each nested **Select Case**.

Select Case (statement) Example

The example uses Select Case to decide what action to take based on user input.

```
Msg$ = "Please enter a letter or number from 0 through 9."
UserInput$ = InputBox$(Msg$)     ' Get user input.

If Val(UserInput$) = 0 Then      ' Is it letter or number?
   If UserUnput$ = "" Then       ' Check for cancel button.
      Exit Sub
   End If
   Select Case Asc(UserInput$)   ' If it's a letter,
      Case 65 To 90              ' must be uppercase.
         Msg$ = "You entered the uppercase letter '"
         Msg$ = Msg$ + Chr$(Asc(UserInput$)) + "'."
      Case 97 To 122             ' must be lowercase.
         Msg$ = "You entered the lower-case letter '"
         Msg$ = Msg$ + Chr$(Asc(UserInput$)) + "'."
      Case Else                  ' must be something else.
         Msg$ = "You did not enter a letter or number."
   End Select
Else
   Select Case Val(UserInput$)   ' If it's a number,
      Case 1, 3, 5, 7, 9         ' it's odd.
         Msg$ = UserInput$ + " is an odd number."
      Case 0, 2, 4, 6, 8         ' it's even.
         Msg$ = UserInput$ + " is an even number."
      Case Else                  ' it's out of range.
         Msg$ = "You entered a number outside "
         Msg$ = Msg$ + "the requested range."
   End Select
End If
MsgBox Msg$                      ' Display message.
```

SendKeys (statement) — Send keystrokes to the active application

```
SendKeys string-expression[, wait]
```

Sends keystrokes to the active application.

Parameter	**Description**
string-expression	The keystrokes to be sent. For details, refer to the **Comments** section below.
wait	If True, **SendKeys** does not return until all keys are processed. If False, **SendKeys** does not wait for an application to process the keys. The default *wait* value is False.

Comments

In *string-expression*, to specify that SHIFT, ALT, or CONTROL keys should be pressed simultaneously with a character, use appropriate prefix from the table below:

+	to specify SHIFT
%	to specify ALT
^	to specify CONTROL

Parentheses may be used to specify that SHIFT, ALT, or CONTROL key should be pressed with a group of characters. For example, ‘‘%(abc)’‘ is equivalent to ‘‘%a%b%c’‘.

Since '+', '%', '^' ,'(' and ')' have special meaning to **SendKeys**, they must be enclosed in braces. For example *string-expression*’‘{%}’‘ specifies a percent character '%'.

The other characters that need to be enclosed in braces are '~' which stands for a newline or ‘‘Enter’‘ if used alone, and braces themselves (use {{} to send '{' and {}} to send '}'). Brackets '[' and ']' do not have special meaning in EXTRA! Basic 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, followed by a space, followed by the number of times the character is to be sent. For example, use {X 20} to send 20 characters 'X'.

To send a non-printable key, use a special keyword enclosed with 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, 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 as you would send several normal keys: {UP 25} sends 25 Up arrows.

SendKeys can send keystrokes only to the currently active application. Use the **AppActivate** statement to activate an application.

SendKeys cannot be used to send keys to an application which was not designed to run under Windows.

SendKeys (statement) 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
```

Set (statement) — Assign an object reference to a variable

```
Set ObjectVar = {ObjectExp | Nothing}
```

Assigns an object reference to a variable. Objects provide programmatic access to applications that support OLE Automation.

Parameter	**Description**
ObjectVar	Name of a variable to which a reference to an object is assigned.
ObjectExp	Expression consisting of the name of an object, another declared variable of the same object type, or a function, property, or method that returns an object.
Nothing	Reserved word that discontinues association of *ObjectVar* with any specific object. If no other variables refer to the object, **Nothing** releases all the resources associated with the object.

Comments

Before using the Set statement, you must first declare a variable as an object with one of the following statements: Dim, Global, or Static.

The Set statement assigns an object reference to a variable, not a copy of the object. Therefore, more than one object variable can refer to the same object. Any change to an object affects all variables that reference that object.

In *EXTRA! Personal Client*, the Set statement is typically used to assign a reference to an object returned by a function, method, and property, as shown below.

Function

In this example, the GetObject function returns a Session object, and a reference to the object is assigned to the Ses1 variable.

```
Dim Ses1 As Object
Set Ses1 = GetObject("c:\program
files\e!pc\sessions\Session1.EDP")
```

Method

In this example, the Screen object's Area method returns an Area object. A reference to the Area object is then assigned to the City variable.

```
Dim City As Object
Set City = Ses1.Screen.Area(1,1,12,80)
City = "Seattle"
```

Property

In this example, the System object's ActiveSession property returns the currently active Session object. A reference to the Session object is then assigned to the Ses variable.

```
Dim Ses As Object
Set Ses = SystemObject.ActiveSession
```

Set (statement) Example

This example returns two Session objects and uses the Set statement to assign object references to two variables, ses1 and ses2.

```
Sub Main()
  Dim ses1 As Object, ses2 As Object
  Set ses1 = GetObject("c:\program _
      files\e!pc\sessions\Session1.EDP")
  Set ses2 = GetObject("c:\program _
      files\e!pc\sessions\Session2.EDP")
End Sub
```

SetAttr (statement) — Set the attributes for a file

SetAttr *filename*$, *attributes*%

Sets the attributes for a file.

Parameter	Description
filename$	A String expression containing the name of the file whose attributes are to be modified. Wildcards are not permitted.
attributes	An Integer containing the new attributes for the file:

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

Comments

Using the **SetAttr** statement, you can only modify the attributes of a read-only file.

SetAttr (statement) 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
```

SetField (function) — Return a string containing a replacement substring

```
rc$ = SetField$(string$,field-number%,_
field$,separator-chars$)
```

Returns a string created from a copy of the source *string* with a substring replaced.

Parameter	**Description**
*rc***$**	The return value.
*string***$**	The source string
*field-number***%**	The field number to be replaced
*field***$**	The string that replaces *field***$**
*separator-chars***$**	The character that separates the two fields

Comments

The source *string* is considered to be divided into fields by separator characters. Multiple separator characters may be specified. The fields are numbered starting with one.

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. If more than one separator character was specified, the first one will be used as the separator character.

It is legal for the new field value to be a different size than the old field value.

SetField (function) 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
```

Sgn (function) — Return the sign of a number

```
rc% = Sgn (numeric-expression)
```

Returns a value indicating the sign of a numeric expression.

Parameter	**Description**
rc%	The return value.
numeric-expression	Any numeric data type.

Comments

The value that the **Sgn** function returns depends on the sign of the numeric expression:

If numeric-expression is:	**Sgn(numeric-expression) returns:**
greater than zero	1
equal to zero	0
less than 0	-1

Sgn (function) 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
```

Shell (function) — Run an executable program

```
rc% = Shell(commandstring$,[windowstyle%])
```

Runs the executable program you specify. When execution begins, **Shell** returns a task ID: a unique number that identifies the running program.

Parameter	**Description**
*rc***%**	The return value.
*command string***$**	The name of the program you want to execute. It can be the name of any valid .COM, .EXE, .BAT, or .PIF file. Arguments or command line switches can also be included.
*window style***%**	The style of the window in which the program is to be run. Choose from the following:

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 omitted, it defaults to 1.

Comments

If *commandstring***$** is not a valid executable file name, or if **Shell** cannot start the program, an error message is generated.

Shell (function) Example

The example uses Shell to leave the current application and run the Calculator program included with Microsoft Windows. It uses the two Windows calls to find and minimize the Program Manager and EXTRA! Basic. It then returns EXTRA! Basic back to normal after prompting the user.

```
Declare Function ShowWindow Lib "User" (ByVal hWnd As Integer,
ByVal nCmdShow As Integer) As Integer
' The Declare statement above must appear on one line in your macro
'  source code.
Declare Function FindWindow Lib "USER.EXE" (ByVal szClass$, ByVal _
      lpsz As Long) as Integer
' The Declare statement above must appear on one line in your macro
'  source code.

Dim Hwnd as Integer

Sub Main

Hwnd% = FindWindow("progman",0)       ' Find the Program Manager
x% = ShowWindow(Hwnd%,6)              ' Minimize Program Manager
Hwnd% = FindWindow("EBFrameClass",0)  ' Find EXTRA! Basic
x% = ShowWindow(Hwnd%,6)              ' Minimize EXTRA! Basic

hWndCalc% = Shell("calc.exe", 1)      ' Shell the calculator.

Msg$ = "Close the calculator. Choose OK to return to EXTRA! Basic."
MsgBox Msg$

x% = ShowWindow(Hwnd%,1)              ' Bring EXTRA! Basic back to
                                      ' normal size

End Sub
```

Sin (function) — Return the sine of an angle

rc# = **Sin(***angle***)**

Function returns the sine of an angle.

Parameter	Description
rc#	The return value.
angle	*angle* is specified in radians, and can be either positive or negative.

Comments

Refer to the following table to determine how a returned parameter will be converted.

Sign	Parameter Type	Return Type
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

The return value is between -1 and 1.

To convert radians to degrees, multiply radians by 180/Pi (or 57.2957795130824) where Pi equals 3.141593.

Sin (function) Example

The example uses Sin to calculate the sine of an angle with a user-specified number of degrees.

```
Msg$ = "Enter an angle in degrees."
Pi# = 4 * Atn(1#)                    ' Calculate Pi.
Degrees# = Val(InputBox$(Msg$))      ' Get user input.

Radians# = Degrees# * (Pi# / 180)    ' Convert to radians.
```

```
Msg$ = "The sine of a " + Str$(Degrees#) + " degree angle is "
Msg$ = Msg$ + Str$(Sin(Radians#)) + "."
MsgBox msg$                           ' Display results.
```

Space (function) — Return a string containing spaces

*rc***$** = **Space**[**$**]**(***numeric-expression***)**

Returns a string containing the specified number of spaces.

Parameter	**Description**
*rc***$**	The return value.
numeric-expression	The number of spaces the returned string should contain. numeric-expression can be of any numeric data type but will be rounded to the nearest whole number before being evaluated. *numeric-expression* must be between 0 and 32,767.

Space (function) Example

The example uses Space to create a variable containing 10 spaces.

```
Msg$ = "Enter your first name."
UserInput$ = InputBox$(Msg$)          ' Get user input.
Pad$ = Space$(10)                     ' Create a 10-space pad string.
Msg$ = "Notice the 10-space pad between the first part of this "
Msg$ = Msg$ + "and your name.  " + Pad$ + UserInput$
MsgBox Msg$                           ' Display message.
```

Spc (function) — Specify number of spaces to print (within a Print statement)

rc = **Spc(***numeric-expression***)**

Specifies the number of spaces to print within a print statement.

Parameter	**Description**
rc	The return value.
numeric-expression	Specifies the number of spaces that should be output.

Comments

The **Spc** function uses the following rules for determining the number of spaces to output:

If the width of the output line is not set (using the **Width** statement), **SPC** outputs the *numeric-expression* spaces. Otherwise, it outputs *numeric-expression* **Mod** *width* spaces, unless the difference between the width of the line and the current print position is less than *numeric-expression* **Mod** *width.* In this case, the **Spc** function skips to the beginning of the next line and outputs (*numeric-expression* **Mod** *width*) **-** (*width* **-** *current-position*) spaces.

Spc (function) 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
```

Sqr (function) — Return the square root of a number

```
rc# = Sqr(numeric-expression)
```

Returns the square root of a numeric expression.

Parameter	**Description**
rc#	The return value.
numeric-expression	Any numeric data type and must evaluate to greater than 0. The result of *numeric-expression* is rounded to the nearest whole number before being evaluated.

Comments

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

Sqr (function) Example

The example uses Sqr to calculate the square root of a user-supplied number.

```
Msg$ = "Enter a non-negative number."
Number# = Val(InputBox$(Msg$))        ' Get user input.
If Number# < 0 Then
   Msg$ = "Cannot determine the square root of a negative number."
```

```
Else
   Msg$ = "The square root of " + LTrim$(Str$(Number#)) + " is "
   Msg$ = Msg$ + Str$(Sqr(Number#)) + "."
End If
MsgBox Msg$                           ' Display results.
```

Static (statement) — Declare a variable's value as unchanging

```
Static variableName [As type] [, variableName _
[As type]] ...
```

Used within a procedure to declare variables and allocate storage space.

Parameter	**Description**
variable Name	Indicates the name of the variable you want to declare.
type	The available data types are: number, string, and record.

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 the procedure definition.

Static (statement) ExampleThis 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<>0i:

    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
```

StaticComboBox (statement) — Create combo box in which listbox is always visible

```
        StaticComboBox x, y, dx, dy, text$, .field
```
–or–
```
        StaticComboBox x, y, dx, dy, stringarray$(), .field
```

Creates a combination of a list of choices and a textbox. The listbox remains visible.

Parameter	**Description**
x, *y*	Specifies the position of the combination box relative to the upper left corner of the client area. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the combination box is centered horizontally within the client area. If *y* is omitted, the box is centered vertically within the client area.
dx, *dy*	The sum of the width of the combination box and the *text***$** parameter. Because proportional spacing is used, the width will vary with the characters used. To approximate the width, multiply the number of characters in the *text***$** parameter (including blanks and punctuation) by 4 and add 12. *dy* is the height of the *text***$** parameter. 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 combination box and the accompanying text will move downward within the dialog box.

Parameter	Description
*text***$**	A string containing the selections for the combobox. This string must be defined, using a **Dim** statement, before the **Begin Dialog** statement is executed. The arguments in the *text***$** string are entered as follows: *dimname* = ‘‘*listchoice*’‘ + **Chr$**(9) = ‘‘*listchoice*’‘ + **Chr$**(9) + ‘‘*listchoice*’‘ + **Chr$**(9)...
stringarray	An array of dynamic strings.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box. The user's selection is recorded in the field designated by the *.field* argument when the OK button (or any button other than CANCEL) is selected. The *.field* argument is also used by the dialog statements that act on this control.

Comments

The **StaticComboBox** statement is equivalent to the **ComboBox** or **DropComboBox** statements, but the listbox of **StaticComboBox** always stays visible. All dialog functions and statements that apply to the **ComboBox** apply to the **StaticComboBox** as well.

StaticComboBox (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, "E! Basic 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
```

Stop (statement) — Halt program execution

```
Stop
```

Halts program execution.

Comments

Stop statements can be placed anywhere in a program to suspend its execution. While the **Stop** statement halts program execution, it does not close files or clear variables.

Stop (statement) Example

For each step through this For...Next loop, Stop suspends execution. To resume program execution after Stop, choose Continue from the Run menu.

```
On Error GoTo ErrorHandler            ' Set up error handler.


Restart:


Msg$ = "Enter a number."
UserInput$ = InputBox$(Msg$)          ' Get user input.
TestNum# = Val(UserInput$)            ' Convert to number.
If TestNum# = 0 Then
   Goto ErrorHandler
End If

BadInput:


SquareRoot# = Sqr(TestNum#)           ' Calculate square root.
Msg$ = "The square root of " + UserInput$ + " is: "
Msg$ = Msg$ + LTrim$(Str$(SquareRoot#))
MsgBox Msg$                           ' Display message.

Exit Sub
```

```
ErrorHandler:
Msg$ = "The number you entered was not zero or a "
Msg$ = Msg$ + "positive number."
MsgBox Msg$                           ' Display error message.
Print "User entered: " + UserInput$ + "."

Stop                                  ' Suspend program.
Resume Restart                        ' Try it again.
```

Str (function) — Return the string representation of a number

`rc$ = Str[$](numeric-expression)`

Returns a string representation of the value of a numeric expression.

Parameter	Description
rc$	The return value.
numeric-expression	Any numeric data type.

Comments

A leading space indicating the sign of the number is reserved in the returned value.

Refer to the following table to determine how the numeric-expression will be evaluated before being cast to a string.

Sign	Parameter Type	Evaluates To
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

Str (function) Example

The example uses Str to return a string representation of the values contained in two variables. Note that because the numbers are positive, a space for the implied plus sign precedes the first string character.

```
Randomize                       ' Seed random number generator.
Dice1% = Int(6 * Rnd + 1)       ' Generate first die value.
Dice2% = Int(6 * Rnd + 1)       ' Generate second die value.
Msg$ = "You rolled a " + LTrim$(Str$(Dice1%))
```

```
Msg$ = Msg$ + " and a " + LTrim$(Str$(Dice2%))
Msg$ = Msg$ + " for a total of "
Msg$ = Msg$ + LTrim$(Str$(Dice1% + Dice2%)) + "."
MsgBox Msg$                      ' Display message.
```

StrComp (function) — Compare two strings

```
rc = StrComp(string1$, string2$ [, comparetype%])
```

Compares two strings.

Parameter	Description
rc	The return value. -1 if the *string1* is less than *string2*, 0 if the two strings are identical, 1 if *string1* is greater than *string2*, and null if either string is Null.
string1$, *string2*$	The strings to compare.
compare type%	Determines method of comparison. If 0, a case sensitive comparison based on the ANSI character set sequence is performed. If 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** will be used.

Comments

The *string1* and *string2* arguments are both passed as variants. Therefore, any type of expression is supported. Numbers will be automatically converted to strings.

StrComp (function) 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
```

String (function) — Return a string of repeated characters

```
rc$ = String[$](numeric-expression, charcode%)
```

–or–

```
rc$ = String[$](numeric-expression, _
string-expression$)
```

Returns a string consisting of a repeated character.

Parameter	**Description**
*rc***$**	The return value.
numeric-expression	Any numeric data type and indicates the length of the string. The result of numeric-expression is rounded to the nearest whole number before being evaluated. *numeric-expression* must be between 0 and 32,767.
*charcode***%**	Any decimal ANSI code. This becomes the repeated character. *charcode***%** must be a numeric expression that evaluates to an integer between 0 and 255, inclusive.
*string-expression***$**	A string, the first character of which becomes the repeated character.

String (function) Example

The example uses the String function to return a Variant consisting of 10 asterisks. The ANSI code of the asterisk character is 42.

```
NL$ = Chr$(13) + Chr$(10)        ' Define newline.
Msg$ = "This demo generates two strings of 10 asterisks. Both "
Msg$ = Msg$ + "forms of String$ syntax are illustrated.  " + NL$
Msg$ = Msg$ + NL$ + "Here is the 1st: " + String$(10, "*") + NL$
Msg$ = Msg$ + "Here is the 2nd: " + String$(10, 42)
MsgBox Msg$                      ' Display message.
```

Sub...End Sub (statement) — Define a subprogram procedure

```
[Static][Private] Sub name[(parameter _
[As type] ,...)]
[statementblock]
[Exit Sub]
[statementblock]
End Sub
```

Defines the name, parameters, and code that form the body of a subprogram procedure. A call to a subprogram stands alone as a separate statement. (Refer to the description of the **Call** statement). Recursion is supported.

Parameter	**Description**
name	The name of the subprogram procedure. Subprogram names follow the same naming rules and conventions as other variables but cannot include a type declaration character. In addition, *name* cannot be the same as any other globally recognized procedure, global variable, or global constant name in the program scope.
parameter	A comma-separated list of parameter names. Specify the data type of a parameter with a type character or by using the **As** clause. Record parameters are declared using an **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the parameter. The array's dimensions are not specified in the Sub statement. All references to an array parameter within the body of the subprogram must have a consistent number of dimensions.
type	The data type of the corresponding parameter. *type* is not necessary if a data type character is supplied with *parameter*.

Parameter	Description
statement block	Any group of EXTRA! Basic statements that will be executed when control is passed to the subprogram.

Comments

Executable code must be contained in either a Sub or Function procedure. Like a function, a subprogram can accept parameters, execute a series of statements, and alter the value of its parameters. Sub procedures cannot be nested within other Sub or Function procedures.

The function returns to the caller when the **End Sub** statement is reached or when an **Exit Sub** statement is executed.

Notes: While subprograms can be recursive, recursion can lead to stack overflow. Use caution when programming in this manner.

You cannot use GoTo to enter or exit from a Sub procedure.

The MAIN subprogram is unique. In many implementations of Basic, MAIN will be the first procedure called when the module is executed. The MAIN subprogram does not take parameters.

Use the **Function** statement to define a procedure that returns a value.

Sub...End Sub (statement) Example

This example is a subroutine that uses the Sub...End Sub function.

```
Sub main

  MsgBox "Hello, World."

End Sub
```

Tab (function) — Simulate pressing the Tab key (within a Print statement)

rc = **Tab**(*numeric-expression*)

Moves the current print position to the column specified by *numeric-expression*. The leftmost print position is position number 1.

Parameter	Description
rc	The return value.
numeric-expression	Specifies the new print position.

Comments

The **Tab** function can be used only within a **Print** statement.

If the width of the output line is not set (using the **Width** statement), the new print position is equal to *numeric-expression*. Otherwise, the new print position is equal to *numeric-expression* **Mod** *width*, unless the current print position is greater than *numeric-expression* **Mod** *width*. In this case, **Tab** skips to the next line and sets print position to *numeric-expression* **Mod** *width.*

Tab (function) 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
```

Tan (function) — Return the tangent of an angle

```
rc# = Tan (angle)
```

Takes an angle as input and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

Parameter	**Description**
rc#	The return value.
angle	Any valid numeric expression measured in radians. *angle* can be positive or negative.

Comments

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

To convert radians to degrees, multiply radians by 180/Pi (or 57.2957795130824) where Pi equals 3.141593.

To convert degrees to radians, multiply degrees by Pi/180 (or .0174532925199433) where Pi equals 3.141593.

Tan (function) Example

The example uses Tan to calculate the tangent of an angle with a user-specified number of degrees.

```
Pi# = 4 * Atn(1#)                   ' Calculate Pi.

Msg$ = "Enter an angle in degrees."
```

```
Degrees# = Val(InputBox$(Msg$))      ' Get user input.
Radians# = Degrees# * (Pi# / 180)    ' Convert to radians.
Msg$ = "The tangent of a " + LTrim$(Str$(Degrees#)) + " degree"
Msg$ = Msg$ + " angle is " + LTrim$(Str$(Tan(Radians#))) + "."
MsgBox Msg$                          ' Display results.
```

Text (statement) — Display text in a dialog box

```
Text x, y, dx, dy, text$[, .id]
```

Sets up line(s) of text in a dialog box.

Parameter	**Description**
x, y	*x* and *y* specify the position of the text relative to the upper left corner of the dialog box. The *x* argument is measured in units one-quarter the average width of the system font. The *y* argument is measured in units one-eighth the height of the system font. If *x* is omitted, the text is centered horizontally within the client area. If *y* is omitted, the text is centered vertically within the client area.
dx, dy	*dx* and *dy* specify the width and height of the text. The *dx* argument is measured in one-quarter system-font character-width units. The *dy* argument is measured in one-eighth system-font character-width units. A *dy* value of 14 typically accommodates system font text.
*text***$**	*text***$** supplies the text that will appear to the right of the position designated by the *x* and *y* coordinates. If the width of this string is greater than *dx*, trailing characters are wrapped to the next line until the height of *dx* and *dy* is exceeded (at which point extra characters are truncated). To indicate an accelerator key in *text***$**, precede the accelerator key character with an ampersand (&).

Comments

The **Text** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

Text (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, "E! Basic 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
```

TextBox (statement) — Create an editable text box within a dialog box

`TextBox` `[`**`NoEcho`**`]` *`x, y, dx, dy, .field`*

Creates a box, used within a dialog box, in which the user can enter and edit text.

Parameter	**Description**
x, y	*x* and *y* specify the position of the text box relative to the upper left corner of the dialog box. *x* is measured in units one-quarter the average width of the system font. *y* is measured in units one-eighth the height of the system font. If *x* is omitted, the text box is centered horizontally within the client area. If *y* is omitted, the text box is centered vertically within the client area.
dx, dy	*dx* and *dy* specify the width and height of the text box. *dx* is measured in one-quarter system-font character-width units. *dy* is measured in one-eighth system-font character-width units. A *dy* value of 14 typically accommodates system font text.
.field	The name of the dialog-record field that will hold any text entered in the text box. When the user chooses the dialog box's OK button (or any button other than CANCEL), the text string entered in the text box is assigned to the record field.

Comments

The **TextBox** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

TextBox (statement) Example

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

```
Sub main
   Begin Dialog UserDialog 194, 76, "E! Basic 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
```

Time (function) — Return current time

```
rc$ = Time$
```

Returns a string containing the current system time.

Parameter	**Description**
*rc***$**	The return value.

Comments

The **Time$** function returns an eight-character string. The form of the string is *hh:mm:ss,* where *hh* is the number of hours past midnight, *mm* is the number of minutes, and *ss* is the number of seconds. The hour is specified using a 24-hour clock, and ranges from 0 to 23.

Time (statement) — Set the current system time

```
Time[$] = expression
```

Sets the current system time.

Parameter	**Description**
expression	A string of one of the forms listed below, a valid date string, a Date variant, or a String variant.

Comments

If the **Time$** form is used, *expression* must evaluate to a string of one of the following forms:

hh	Set the time to *hh* hours 0 minutes and 0 seconds.
hh:mm	Set the time to *hh* hours *mm* minutes and 0 seconds.
hh:mm:ss	Set the time to *hh* hours *mm* minutes and *ss* seconds.

Time uses a 24-hour clock (e.g., 6:00 pm is 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 a **variant** of **vartype** 8 (String).

If *expression* is not already a **variant** of vartype 7 (date), **Time** attempts to convert it to a valid time. It recognizes the time separator defined in the Windows Control Panel. Both 12- and 24-hour clocks are accepted.

Time (statement) Example

This example writes data to a file if it hasn't been saved within the last 2 minutes.

```
Sub main
   Dim tempfile
   Dim filetime, curtime
   Dim msgtext
   Dim acctno(100) as Single
   Dim x, I
   tempfile="C:\TEMP001"
   Open tempfile For Output As #1
```

```
      filetime=FileDateTime(tempfile)
      x=1
      I=1
      acctno(x)=0
      Do
         curtime=Time
         acctno(x)=InputBox("Enter an account number (99 to end):")
         If acctno(x)=99 then
            For I=1 to x-1
               Write #1, acctno(I)
            Next I
            Exit Do
         ElseIf (Minute(filetime)+2)<=Minute(curtime) then
            For I=I to x
               Write #1, acctno(I)
            Next I
         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
```

Timer (function) — Return seconds since midnight

rc!= **Timer**

Returns the number of seconds that have elapsed since midnight.

Parameter	**Description**
rc!	The return value.

Comments

The Timer function can be used in conjunction with the **Randomize** statement to seed the random number generator (**Rnd**).

Timer (function) Example

This example uses Timer Function to find a lottery number.

```
Sub main
   Dim msgtext
   Dim value(9)
   Dim nextvalue
   Dim x
   Dim y
   msgtext="Your Lottery 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
   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
```

TimeSerial (function) — Return a serial time value

rc = **TimeSerial(***hour***%**, *minute***%**, *second***%)**

Returns the serial time equivalent to the specified date / time value.

Parameter	Description
rc	The return value, a variant of vartype 7 (Date).
hour%	An integer or integer expression indicating the number of hours (0 through 23, inclusive).
minute%	An integer or integer expression indicating the number of minutes (0 through 59, inclusive).
second%	An integer or integer expression indicating the number of seconds (0 through 50, inlcusive).

Comments

The range of numbers for each **TimeSerial** argument should conform to the accepted range of values for that unit. You can also 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.

TimeSerial (function) Example

This example displays the current time using TimeSerial.

```
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
```

TimeValue (function) — Return the time value equivalent to the supplied string

`rc = TimeValue(string-expression$)`

Returns a time value for the string specified.

Parameter	**Description**
rc	The return value.
string-expression$	A String representing a time from 0:00:00 through 23:59:59 (12:00:00 am through 11:59:59 pm). Both 12- and 24-hour clock times are valid.

TimeValue (function) 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, I
   tempfile="C:\TEMP001"
   Open tempfile For Output As 1
   ftime=FileDateTime(tempfile)
   filetime=TimeValue(ftime)
   minutes= TimeValue("00:02:00")
   x=1
   I=1
```

```
      acctno(x)=0
      Do
         curtime= TimeValue(Time)
         acctno(x)=InputBox("Enter an account number (99 to end):")
         If acctno(x)=99 then
            For I=I to x-1
               Write #1, acctno(I)
            Next I
            Exit Do
         ElseIf filetime+minutes<=curtime then
            For I=I to x
               Write #1, acctno(I)
            Next I
         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
```

Trim (function) — Remove leading and trailing spaces

rc = **Trim**[**$**]**(***string***)**

Returns a copy of a source string, with all leading and trailing space characters removed.

Parameter	Description
rc	The return string.
*string***$**	Any string expression.

Comments

The dollar sign (**$**) in the function name is optional. If specified, the return type is String. If the dollar sign is 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.

Trim (function) Example

This example removes leading and trailing spaces from a string entered by the user.

```
Sub main
   Dim userstr as String
   userstr=InputBox("Enter a string with leading/trailing spaces")
   MsgBox "The string is: " & Trim(userstr) & " with nothing after."
End Sub
```

Type...End Type (statement) — Declare a user-defined type

```
Type userType
   field1 As type1
   field2 As type2
   ...
End Type
```

Declares a user-defined type that can then be used in the **Dim** statement to declare a record variable. User-defined types are also sometimes referred to as *record types* or *structure types*.

Parameter	**Description**
userType	The name of a user-defined data type. Follow standard variable naming conventions.
fieldx	Name of the field.
typex	The data type of *fieldx*. *typex* must be one of the following: String (either dynamic or fixed), number (Integer, Long, Single, or Double), or a previously-defined record type. A field cannot be defined as an array. However, arrays of records are permitted.

Comments

The **Type** statement is not valid within a procedure definition; it can be used only in the Declarations section of a module. After you have declared a user-defined type with the **Type** statement, you can declare a variable of that type anywhere in your application. Refer to the **Dim**, **Global**, **ReDim**, or **Static** descriptions for more information about declaring a variable of a user-defined type.

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
```

Line number and line label references are not permitted within a **Type** statement.

Type...End Type (statement) 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 &" OK? (Y/N)")
   If answer="Y" or answer="y" then
      MsgBox "Thank you."
   Else
      MsgBox "Try again."
      Goto i
   End If
End Sub
```

Typeof (statement) — Perform statements if an object is of a given class

```
If Typeof objectVariable Is className Then
thenStatement
```

Returns a value indicating whether an object is of a given class (-1=True, 0=False).

Parameter	**Description**
object Variable	The object to test.
className	The class to compare the object to.
then statement	The statement that is executed when *objectVariable* is of type *className*. thenStatement can be one or more EXTRA! Basic statements or a **GoTo** statement pointing to a valid program label.

Comments

Typeof can only be used in an If statement and cannot be combined with other Boolean operators. Typeof can only be used exactly as shown in the syntax above.

To test whether an object belongs to a class, use the following code structure:

```
If Typeof objectVariable Is className Then ...
Else
...
    Rem Perform some action.
End If
```

UBound (function) — Return highest subscript of a dimensioned array

rc% = **UBound(***arrayVariable* [,*dimension*]**)**

Returns the largest available subscript for the specified dimension of the specified array.

Parameter	**Description**
*rc***%**	The return value.
arrayVariable	*arrayVariable* is the name of an array.
dimension	An integer indicating which dimension's upper bound is returned. 1 equals the first dimension, 2 equals the second, and so forth. If dimension is not supplied, it is assumed to be 1.

Comments

The statement:

```
Dim Sales (1 to 3, 4 to 6)
```

declares a 2 dimensional array.

For the array shown above, UBound returns the following:

Statement	**Return Value**
UBound(sales, 1)	3
UBound(sales, 2)	6

Use **UBound** in conjunction with **LBound** to determine the length of an array.

UBound (function) Example

The example uses the UBound function to determine the upper bounds for a three-dimensional array.

```
NL$ = Chr$(13) + Chr$(10): TB$ = Chr$(9)   ' Define newline, tab.


'Generate some random dimensions between 2 and 10 for array size.
```

```
Randomize
A% = Int(9 * Rnd + 2)                      ' First dimension.
B% = Int(9 * Rnd + 2)                      ' Second dimension.
C% = Int(9 * Rnd + 2)                      ' Third dimension.

ReDim Array%(A%, B%, C%)                   ' Set dimensions.

Msg$ = "The test array has these upper bounds: " + NL$
Msg$ = Msg$ + TB$ + "Dimension 1 = " + Str$(UBound(Array%, 1)) + NL$
Msg$ = Msg$ + TB$ + "Dimension 2 = " + Str$(UBound(Array%, 2)) + NL$
Msg$ = Msg$ + TB$ + "Dimension 3 = " + Str$(UBound(Array%, 3))
MsgBox Msg$                                ' Display message.
```

UCase (function) — Return a string in uppercase

rc$ = **UCase$(***string*$**)**

Returns a copy of a source string, with all lowercase letters converted to uppercase.

Parameter	**Description**
rc$	The return value.
string$	Any string expression.

Comments

Conversion is based on the country specified in the Windows Control Panel.

UCase (function) Example

The example uses UCase to return an all-uppercase version of the argument string.

```
Lowercase$ = "once upon a time"         ' String to convert.
Uppercase$ = UCase$(Lowercase$)         ' Convert to uppercase.
Msg$ = "UCase$ converts """ + Lowercase$ + """ to """
Msg$ = Msg$ + Uppercase$ + ".  """
MsgBox Msg$                             ' Display message.
```

Unlock, Lock (statements) — Controls access to some or all of an open file

```
Lock [#]filenumber%[, {record | [start&] _
[To end&]}]
...
Unlock [#]filenumber%[, {record | [start&] _
[To end&]}]
```

In a networked environment, controls access by other processes to some or all of the records or bytes in an open file.

Parameter	**Description**
filenumber	An integer or integer expression identifying the open file. This argument should reference the same parameter specified in the **Open** statement.
record	The number of the block or record you want to lock or unlock (1 to 2,147,483,647, inclusive).
start	A Long integer indicating the first byte or record you want to lock or unlock.
end	A Long integer indicating the last byte or record you want to lock or unlock.

Comments

Important: The arguments passed to Lock and Unlock must match exactly. Also, locked open files must be unlocked before closing or unpredictable results may occur.

For files opened in Random mode, start and end are record numbers. For files opened in Binary mode, start and end are byte offsets. For Input, Output, and Append modes, *start* and *end* are ignored and the whole file is locked or unlocked.

If an *end* argument is supplied without a *start* argument, all records or bytes from *record* or offset 1 to *end* are locked or unlocked. If a *start* argument is supplied without an *end* argument, only the record or byte at the location indicated by *start* is locked or unlocked.

Val (function) — Return the numeric value of a string of characters

```
rc! = Val(string$)
```

Returns a numeric value corresponding to the first series of numbers found in the specified *string*.

Parameter	**Description**
rc!	The return value.
string$	The string in which you want to search for numbers. *string*$ can be a string variable, string expression, or string literal.

Refer to the following table to determine how a returned parameter will be converted.

Sign	**Parameter Type**	**Return Type**
%	Integer	Single-precision integer
!	Single	Single-precision floating point
&	Long	Double-precision integer
#	Double	Double-precision floating point

Comments

The **Val** function stops processing the string expression when it encounters the first non-numeric character. The **Val** function processes the radix prefixes **&**O (octal) and **&**H (hexadecimal). Refer to the following examples:

String Expression Passed to Val	**Value Returned**
‘‘SS#344-65-9361‘‘	0
‘‘458266-Med.‘‘	458266
‘‘**&**HFFFB’‘	-5

If no number is found, **Val** returns zero (0).

Val (function) Example

The example uses the Val function to determine if the first characters in a string are a number.

```
Dim Msg$                                            ' Declare variables.
Dim Number As Double
Number = Val(InputBox$("Enter a number"))          ' Get user input.
Msg$ = "The number you entered is:" + STR$(Number) ' Display number.
MsgBox Msg$                                         ' Display message.
```

VarType (function) — Indicate the type of data stored in a variant

```
rc = VarType(variant)
```

Returns an ordinal number indicating the type of data that is currently stored in the specified variant variable.

Parameter	Description
rc	The return value.
variant	The name of the variant variable to check.

Ordinal	Representation
0	(Empty)
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	String
9	Object

VarType (function) 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
```

Weekday (function) — Return the day of the week portion of a date/time value

rc = **Weekday(***expression***)**

Returns the day of the week derived from a supplied date/time value.

Parameter	Description
rc	The return value (an integer between 1 and 7, inclusive).
expression	Any type of expression (numeric or string) that represents a date from January 1, 100 (-657434) through December 31, 9999 (2958465). A value of 2, for example, represents January 1, 1900. Numbers to the left of the decimal point are interpreted as a date; numbers to the right are interpreted as time values. Negative numbers are interpreted as dates prior to December 30, 1899.

Comments

The return value is a variant of vartype 2 (Integer).

If expression is Null, **Weekday** returns a Null.

Weekday (function) Example

This example finds the day of the week on which New Year's Day will fall in 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
```

While ... Wend (statement) — Execute a series of statements while a condition is true

```
While condition
   statementblock
Wend
```

Controls a repetitive action. The *condition* is tested, and if true, the *statementblock* is executed. This process is repeated until *condition* becomes false.

Parameter	**Description**
condition	Any Boolean expression that EXTRA! Basic can determine to be true (nonzero) or false (zero).
statement block	EXTRA! will repeat the program lines contained in the *statementblock(s)* as long as *condition* is true.

Comments

The While statement is included in EXTRA! Basic for compatibility with older versions of EXTRA! Basic. The **Do...Loop** statement is a more general and powerful flow control statement.

Caution: Do not branch into the body of a **While...Wend** loop. Failing to execute the **While** statement may cause run-time errors or transient program bugs.

While ... Wend (statement) Example

The example uses While...Wend in a sorting routine.

```
Const FALSE = 0, TRUE = -1, MAX = 5
NL$ = Chr$(13) + Chr$(10)           ' Define newline.
Dim A$(MAX)                         ' Create an array.

A$(1) = "New York"                  ' Put data in array in no
A$(2) = "Paris"                     ' particular order so it
A$(3) = "Chicago"                   ' can be sorted.
```

```
A$(4) = "London"
A$(5) = "Berlin"
Exchange% = TRUE                        ' Force first pass through array.
While Exchange%                         ' Sort until no elements are
   Exchange% = FALSE                    ' exchanged.
   ' Compare array elements by pairs.  When two are exchanged,
   ' force another pass by setting exchange to TRUE.
   For I% = 2 To MAX

      If A$(I% - 1) > A$(I%) Then
         Exchange% = TRUE               ' Make swap.
         Temp$ = A$(I%): A$(I%) = A$(I% - 1): A$(I% - 1) = Temp$
      End If
   Next I%
Wend
Msg$ = "Sorted order: " + NL$ + NL$     ' Create message that shows
For I% = 1 To MAX                       ' sorted order of array
   Msg$ = Msg$ + A$(I%) + NL$           ' contents.
Next I%

MsgBox Msg$                             ' Display message.
```

Width (statement) — Set output line width for an open file

```
Width filenumber%, width
```

Determines the output line width for the specfied file. The specified file must be open when the **Width** statement is executed.

Parameter	**Description**
filenumber%	An integer expression identifying an open file to query for position.
width	An integer expression in the range 0 to 255 specifying the number of characters on a line before a newline is started. A value of zero (0) indicates there is no line length limit. The default *width* for a file is zero (0).

Comments

filenumber% is the number assigned to the file when it is opened. See the Open statement for more information.

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
```

With (statement) — Execute statements on a specified variable

```
With variable
   statement-block
End With
```

Executes a series of statements on an object or user-defined type.

Parameter	**Description**
variable	An object or user-defined type.
statement-block	A series of statements to be performed on *variable.*

Comments

With statements can be nested.

For the example .rtf:

```
With Alex
    .ss = 037672947  'Sets Alex.ss
    .salary = 60000  'Sets Alex.salary
    .dob = #10-08-65#          'Sets Alex.dob
    With .address
            .street = ‘‘15 Chester Str.‘          'Sets Alex.address.street
            .apt = ‘‘28’‘                         'Sets Alex.address.apt
            .city = ‘‘Cambridge’‘                 'Sets Alex.address.city
            .state = ‘‘MA‘              'Sets Alex.address.state
    End With
    Print ‘‘Done with ‘‘; .name                   'Prints ‘‘Done with ‘‘
Alex.name
End With
```

With (statement) 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 ‘‘Alex‘‘.

```
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 Alex as custrecord
   Dim msgtext
   Alex.name="Alex"
   With Alex
      .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) & Alex.name & Chr(10)
   msgtext=msgtext & "SS#: " & Space(6) & Alex.ss & chr(10)
   msgtext=msgtext & "D.O.B:" & Space(4) & Alex.dob
   Msgbox "Done with: " & Chr(10) & msgtext
End Sub
```

Write (statement) — Write to a file

```
Write #filenumber% [,expressionlist]
```

Writes data to a sequential file.

Parameter	**Description**
*#file number***%**	*#filenumber***%** is the file number used in a previous **Open** statement or any integer expression that evaluates to that file number.
expression list	*expressionlist* specifies one or more values to be written to the file. An expression must be a string or numeric expression. Multiple expressions must be separated by commas. If the *expressionlist* argument is omitted, the **Write #** statement writes a blank line to the file.

Comments

The file must have been opened in output or append mode.

As the elements of *expressionlist* are written to the file, **Write #** inserts commas between items and places quotation marks around strings. **Write #** appends a newline character to *filenumber***%** after it has written the last character in *expressionlist*.

Write (statement) Example

The example uses Write # to write two variables to a sequential file.

```
NL$ = Chr$(13) + Chr$(10)             ' Define newline.
                                      ' Make a sample data file.
Open "WRIDATX.DAT" For Output As #1   ' Open file for output.
Msg$ = "Enter your name."
UsrName$ = InputBox$(Msg$)            ' Get user name.

Msg$ = "Enter your age."
UsrAge$ = InputBox$(Msg$)             ' Get user age.
```

```
Write #1, UsrName$, UsrAge$             ' Write data to file.

Close #1                                ' Close file.

' Read the sample data file
Open "WRIDATX.DAT" For Input As #1      ' Open file for input.
Input #1, UsrName$, UsrAge$             ' Read data.

Close #1                                ' Close file.
Msg$ = "The name '" + UsrName$ + "'was read from the data file. "
Msg$ = Msg$ + "'" + UsrAge$ + "'" +  "  is the age you gave."
Msg$ = Msg$ + NL$ + NL$ + "Choose OK to remove test file."
MsgBox Msg$                             ' Display message.
Kill "WRIDATX.DAT"                      ' Remove file from disk.
End Sub
```

Year (function) — Return the year component of a date/time value

rc = **Year(***expression***)**

Returns the year derived from the supplied date/time value.

Parameter	Description
rc	The return value, an integer between 100 and 9999, inclusive.
expression	Any type of *expression* including a string expression that can be converted to a date value.

Comments

The return value is a variant of vartype 2 (integer). If the value of *expression* is null a variant of vartype 1 (null) is returned.

Year (function) Example

This example returns the current year.

```
Sub main
  Dim nowyear
  nowyear=Year(Now)
  MsgBox "The current year is: " &nowyear
End Sub
```

INDEX