Table of Contents

Visual CommBasic Overview

Introduction to Visual CommBasic

Visual CommBasic allows you to create powerful macro programs to automate OutsideView processes using VCBasic, an event-driven scripting language. With Visual CommBasic, you can write macros that OutsideView users can execute, either selectively by opening macro files or automatically via toolbar buttons.

Visual CommBasic has the following advantages:

- VCBasic is very similar to Visual Basic. You don't have to learn a proprietary programming language to write macros. The operation, structure, syntax and commands are similar to those you're accustomed to in Visual Basic. We have created a comparison of Visual CommBasic and Visual Basic for you.
- Commands and functions are provided for accessing session data, transferring files, and changing the I/O properties of the session. These emulation manipulation commands give you complete control of OutsideView.
- VCBasic includes all the tools and functionality you need to write and debug your macro application. It's not just a language, it's a development platform.
- We’ve even included some complete, useful sample macros so that you can see how they work.

What is a macro?

A macro is the term used for a program developed specifically to work with OutsideView. A macro is normally used to perform tasks automatically for the user. Macros can perform complete (or closed) tasks, such as connecting to a host system, logging on, downloading a file, uploading a separate file, and logging off.

Macros provide you with tremendous flexibility. You can access all the communications power of OutsideView and also perform tasks that can't be done within OutsideView such as reading and writing data files. Macros can also be used to perform tasks without user intervention. For example, an unattended PC could dial into a remote system and download a file late at night, when connect charges are lower.

Writing your macro program

To write your macro program in Visual CommBasic, use the following procedure:

1. Create a new form (using File:New).
2. Design your form by creating controls on it and setting their properties.
3. Write scripts for your form using the Script Editor.
Test and debug the form and its scripts with the integrated debugger.

This procedure is normally iterative: repeat as necessary when adding forms, and their associated controls and scripts, to your project.

Visual CommBasic Fundamentals

Visual CommBasic is a comprehensive development environment that supports many of the constructs found in modern programming languages. Presented here are some of the core concepts of forms-based, event-driven programming. For a more thorough description of these techniques, please refer to the Microsoft Visual Basic Programmer's Guide.

Objects, Properties, Events and Methods

Visual CommBasic incorporates many of the concepts of modern programming environments including objects, properties, events and methods.

Objects

An object is simply a combination of code and data that may be treated as a unit. Examples of objects in Visual CommBasic include forms and controls.

Visual CommBasic will create unique default names for objects as they are created. For example, as button controls are added to a form, the first button will be named "button1", the second "button2", etc. While there is no error using these default names to access the objects, your code will be much more easily understood and supportable if descriptive object names are used. These descriptive names should be assigned to all objects for which event procedures are to be written before you start writing code. The object naming conventions of Visual Basic are used throughout this introduction.

Properties

A property is the named attribute of an object. Just as a bicycle would have color, type (mountain or road) and speed (10, 16, 21, etc.) properties, a Visual CommBasic form object has properties that define its size, colors and caption. An object's properties may be set at design time using the Property Sheet. Properties may also be accessed at run time using the syntax *objectname.property*. For example, the following commands will read the state (of the property called "value") of an option button named "optChoice", and then clear its state:

```
intRet = optChoice.value

optChoice.value = 0
```

Events

An event is any action recognized by an object. When a user clicks on a push button, for example, the button's click event is triggered. When an object in a Visual CommBasic macro detects that one of its events has occurred, it automatically invokes the procedure corresponding to that event. The naming syntax for event procedures is *objectname_eventname*. For example, when the button named "cmdExit" is clicked, the code in the procedure named "cmdExit_click" is executed.

There is no need to manually create the SUB…END SUB statements for event procedures. The Visual CommBasic script editor automatically generates the procedure template, including the correct event procedure names, by selecting objects and events in the script editor. As with other SUB procedures, an event procedure may also be called from any other procedure in your macro.

Methods

A method performs an action on a particular object. The syntax for executing a method, *objectname.methodname*, is similar to the syntax for accessing a property. The AddItem method, for example, may used to add an item to a listbox control. For a listbox named lstFiles, the command

```
lstFiles.AddItem "NewFile"
```

will add the string "NewFile" to the listbox contents.

Event-Driven Programming

An event-driven application executes code in response to user, system or program-generated actions. A macro's form and its controls each have a fixed set of events to which it can respond. When one of these actions occur, the associated event procedure is invoked. It's up to you, the developer, to determine how each object responds to its events.

A typical Visual CommBasic macro operates in the following manner:

1. When the macro is started, the form is painted and the form's load event procedure is executed.

2. An event occurs which may be generated by the user, the system (e.g., a timer event), or by the macro itself.

3. If an event procedure has been defined for that event, it is executed.

4. The macro goes into an idle state waiting for the next event.

This model of operation is fundamentally different than that of CommBasic (the macro language used by OutsideView 4.x), which follows a "procedural" model. In procedural applications, program execution flow is completely determined by the code. Execution begins at the first line of executable code (the first line of SUB MAIN in CommBasic) and follows the path determined by the programmer. Event-driven applications such as Visual CommBasic macros, however, are developed from the perspective of responding to the actions of the user.

VCBasic Files

All macro components created by the Visual CommBasic development environment are saved in a single file with a ".vcb" extension. This file, any text files included in the macro via the $Include command, and any pictures used by the Picture Box control need to be distributed in order to provide the macro to other OutsideView users. If it is necessary to restrict modification of the macro, the .vcb file may have its read-only attribute set or be placed in a folder with read-only access.

The File Open dialog displayed when the user selects Macro/Run Macro from OutsideView or File/Open from the development environment displays the contents of the default macro folder. This default folder is also used if a macro without a fully qualified path is included on the command line (e.g., "c:\Program Files\Crystal Point\OutsideView\Outside.exe" /Mlogin.vcb). Review the Command Line Options section of the online System Administrator's Guide for more details on available options. The default folder locations for OutsideView may be altered through the Application Settings dialog (Edit:Application Settings).

Forms

Each macro has a single form. Additional dialogs may be created using the Begin Dialog…End Dialog statements. Simple prompts or messages may be presented to the user via the InputBox or MessageBox functions. The visible property of controls may be used to present portions of the user interface in macros with complex forms. The procedures of a macro are private to that macro and cannot be accessed by another macro.

Many of the tasks that may be automated by a Visual CommBasic macro require no user interface. In this case, set the form's visible property to FALSE. When the macro starts, the form's Load event will still fire and any code in the Load event procedure executes.

Variables

A variable that is declared (using Dim) external to any procedure body in the form Common area has scope to all event procedures for that macro's objects.

Variables declared as Global have scope to all running macros. To prevent unexpected behavior, variables should be declared as Global only when the intent is to share data with other macros.

Session Binding

A macro is bound to the session which is active (has focus) when the first command which interacts with the session (e.g., CrtGet or Emit) is executed. A macro may only communicate with the session to which it is bound. There is no means to switch the binding of a macro to another session.

Visual CommBasic Tutorial

Visual CommBasic is a powerful tool that allows you to easily build macro programs for OutsideView. This tutorial describes some of the basic terminology and components of VCBasic, then leads you through creating a simple macro. The time required for completing the tutorial is about one hour.

This is a hands-on example, so keep the VCBasic editor running throughout the tutorial. You will be given the instructions needed to complete each step or task. You can end the tutorial at any time. If you need to leave the tutorial but wish to continue, save your work to a macro file. You can then open the macro file later and continue where you left off before.

While working through the tutorial, you can easily switch between this help window and the VCBasic Editor by either clicking on the desired window or using the Alt+Tab keys.

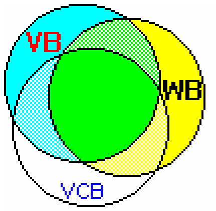
You may want to look at some basic VCBasic Terminology if you're not familiar with it.

Let's move ahead and start creating your first macro!

Visual CommBasic and other Basics

How VCBasic Compares to Visual Basic and Word Basic

There are several versions of Basic with which you might be familiar, the most common being Visual Basic and Word Basic. VCBasic shares a substantial common core of functions and statements with these versions; however, each one has unique capabilities.



Differences Between VCBasic and Visual Basic

VCBasic is very similar to Microsoft's Visual Basic; however, there are some differences.

Functions and Statements Unique to VCBasic

VCBasic offers a few statements and functions not found in Visual Basic:

$CStrings

Assert

CrtAttr

$Include

GetField$

CrtCls

WaitTime

$NoCStrings

SetField$

CrtCopy

CrtRow

CrtSetCursor

Emit

FtSet$

IoInput$

IoTrigger$

Shutdown

WaitDCD

WaitStr

CrtEmit

CrtPosition

CrtSearch

CrtTrigger$

EmitBrk

FtTrigger$

IoQuery$

IoTypeSet$

WaitCrtCursor

WaitKeystrokes

CrtCol

CrtFieldSearch

CrtQuery$

CrtTypeSet$

FtQuery$

FtTypeSet$

IoSet$

RunMacro

WaitCrtUnlock

WaitSilent

Control-Based Objects

VCBasic does not predefine or include any Visual Basic object, such as a Button Control. As a result, a VB property such as "BorderStyle" is not an intrinsic part of VCBasic. This does not mean that as an integrator, you cannot define a VCBasic object that has BorderStyle as a property. You will probably define many objects that are intrinsic 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. In contrast, VCBasic has a set of functions and statements to enable the use of dialog boxes (they are 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.

Forms

Each macro has a single form. Additional dialogs may be created using the Begin Dialog…End Dialog statements. Simple prompts or messages may be presented to the user via the InputBox or MsgBox functions. The visible property of controls may be used to present portions of the user interface in macros with complex forms. The Event and Common procedures for the form are private to that macro and cannot be accessed by another macro.

Variables

With Visual CommBasic, a variable that is declared (using Dim) in the form Common area has scope to all event procedures for that macro's objects. Variables **cannot** be shared between macros.

Differences Between VCBasic and Word Basic

Word Basic is a precursor to 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 VCBasic and Word are very similar. Word does offer some statements and functions that VCBasic does not, such as DlgFilePreview. As well, VCBasic offers some features that Word does not.

In response to the need for certain types of dialog box support, VCBasic offered some dialog box options before Word Basic did. Later, Word Basic came out with their own syntax for these options. As a result, there are minor differences in the way the two languages handle dialog boxes.

Button vs. PushButton

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

PushButton is preferred, and is used throughout the Examples.

Dialog Box Units

The measurement units used in the two dialog box syntaxes are different. VCBasic supports both, and you can choose to use either. Since many of our clients have built scripts based on the original VCBasic units, those are the ones used in the Examples. As a result, if you use Word units, some of the dialog boxes created in the Examples might look odd.

User Input Mechanisms

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

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

How VCBasic Compares to Other Versions of Basic

See also: How VCBasic Compares to Visual Basic and Word Basic

Differences Between Visual CommBASIC and Earlier Versions of Basic

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

The topics below describe some of the differences you will notice between the older Basics and VCBasic.

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.

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, which is placed at the beginning of a line of code. These labels are used by the **Goto** statement.

Subroutines and Modularity of VC Basic

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

Variable Scope

The placement of variable declarations determines their scope:

Scope **Definition**

Local Dimensioned inside a subroutine or function. The variable is accessible only to the subroutine or function that dimensioned it.

Module Dimensioned outside any subroutine or function. The variable is accessible to any subroutine or function in the same file.

Global Dimensioned outside any subroutine or function using the **Global** statement. The variable is accessible to any subroutine or function in any macro.

Data Types

Modern Basic is now a typed language. In addition to the standard data types -- numeric, string, array, and record -- VCBasic 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 OLE2 objects.

Dialog Box Handling

VCBasic contains extensive dialog box support to give you great flexibility in creating and running your own custom 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).

VCBasic stores information about the selections the user makes in the dialog box. When the dialog box is closed, your program can access this information.

VCBasic 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

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

Date and Time Functions

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

Object Handling

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

Environment Control

VCBasic 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$**).

Visual CommBasic Reference

Reference Topics

Conventions
Describes program and typographic conventions

Using the Examples
How to use the example provided with each function and statement

Comparing VCBasic to Other Versions of Basic
Describes differences between VCBasic and earlier versions of Basic, Visual Basic, and Word Basic

Using VCBasic

Data Types	Defines data types and their use
Dialog Boxes	How to create and run a custom dialog box
Dynamic Data Exchange	How to use DDE to talk with other applications
Error Handling	How to trap errors
Expressions	How to use operators to form string and numeric expressions
Objects	How to create and use objects

VC Basic Functions and Statements

Click on the appropriate functional group to jump to a complete list of its functions and statements

Array Handling
Compiler Directives
Control Flow and Assignment
Conversion Functions
Date Time Functions
DDE Functions
Declarations
Dialog Boxes
Disk and Directory Control
Emulation Manipulation Functions
Environment Control
Error Handling
File Control
File Input/Output
Financial Functions
Numeric Functions
Object Handling
Screen Input/Output
SQL Functions
String Functions
Trigonometric Functions
Variant Handling

You can also click here to go to an alphabetical list of all statements, or click here to see a list of statements grouped by function. Refer to the Emulation Manipulation Functions for functions specific to communicating with a host and the operation of OutsideView.

General Help Topics

Expressions	How to form expressions.
Type Conversion	Converting between different data types.
Application Data Types	Extended data types defined through API functions.
Dialog Functions	List of functions and statements to be used when an active dialog box is on the screen.

Conventions
See Also Help Typographic Conventions

VCBasic uses the following programming conventions:

Arguments

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

If an argument is passed by value, it means that the variable used for that argument retains its value when the subroutine or function returns to the caller. If an argument is passed by reference, it means that the variable's value might 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 will always be 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.

When an argument is passed by value to a procedure, the called procedure receives a copy of the argument. If the called procedure modifies its corresponding formal parameter, it will have no affect on the caller. Procedures written in other languages such as C may receive their arguments by value.

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

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

Procedures written in VCBasic are defined to receive their arguments by reference. If you call such a procedure and pass it a variable, and if the procedure modifies its corresponding formal parameter, it will modify the variable.

Passing an expression by reference is valid in VCBasic; if the called procedure modifies its corresponding parameter, a temporary value will be modified with no apparent affect on the caller.

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

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

Arguments passed by reference to a procedure may be modified by the procedure.

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.

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 the statement or function. 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, you get:

myfunction id:=1, action:="get", value:=0

The advantage of named arguments, though, is that you do not 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 forget about 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 VCBasic functions and statements accept named arguments. The argument names are listed in their syntax for the statement and function.

Arrays

Array dimensions are enclosed in parentheses after the array name:

arrayname(a,b,c)

Comments

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

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

Line Continuation

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

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

Records

Elements in a record are identified using the following syntax:

```
record.element
```

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

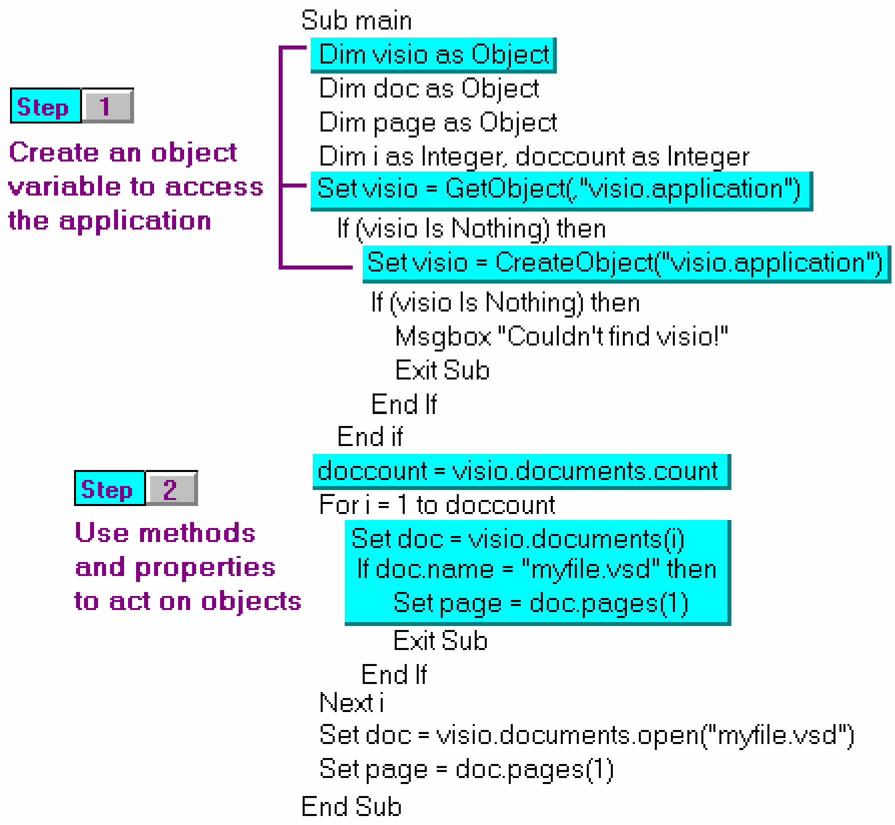
Object Handling
See Also

Objects are the end products of a software application, such as a spreadsheet, graph, or document. Each software application has its own set of properties and methods that change the characteristics of an object. Objects provide access to software functionality outside of VCBasic. Object variables are always **Dim**ed as a particular class. One such class, named **Object**, provides access to OLE2 automation.

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

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

In VCBasic, you have the ability to access an object and use the originating software application to change properties and methods of that object. Before you can use an object in a procedure, however, you must access the software application associated with the object by assigning it to an object variable. Then you attach an object name (with or without properties and methods) to the variable to manipulate the object. The syntax for doing this is shown in the following example code. Click on the blue highlights for more details.



Step 1

Create an object variable to access the application

Step 2

Use methods and properties to act on objects

```
Sub main
  Dim visio as Object
  Dim doc as Object
  Dim page as Object
  Dim i as Integer, doccount as Integer
  Set visio = GetObject(,"visio.application")
    If (visio Is Nothing) then
      Set visio = CreateObject("visio.application")
      If (visio Is Nothing) then
        Msgbox "Couldn't find visio!"
        Exit Sub
      End If
    End if
  doccount = visio.documents.count
  For i = 1 to doccount
    Set doc = visio.documents(i)
     If doc.name = "myfile.vsd" then
        Set page = doc.pages(1)
        Exit Sub
     End If
  Next i
  Set doc = visio.documents.open("myfile.vsd")
  Set page = doc.pages(1)
End Sub
```

Note: The examples shown here are specific to the VISIO software application. Object, property and method names vary from one application to another. You will need to refer to the software documentation for the application you want to access for the applicable names to use.

Dynamic Data Exchange (DDE)
See Also

Dynamic data exchange (DDE) is a process by which two applications communicate and exchange data. One application can be your Basic program. To "talk" to another application and send it data, you need to open a connection, called a DDE channel, using the statement **DDEInitiate**. The application must already be running before you can open a DDE channel. To start an application, use the **Shell** command.

DDEInitiate requires two arguments: the DDE application name and a topic name. The DDE application name is usually the name of the .EXE file used to start the application, without the .EXE extension. For example, the DDE name for Microsoft Word is "WINWORD". The topic name is usually a filename to get or send data to, although there are some reserved DDE topic names, such as **System**. Refer to the application's documentation to get a list of the available topic names.

After you have opened a channel to an application, you can get text and numbers (**DDERequest**), send text and numbers (**DDEPoke**), or send commands (**DDEExecute**). When you have finished communicating with the application, you should close the DDE channel using **DDETerminate**. Because you have a limited number of channels available at once (depending on the operating system in use and the amount of memory you have available), it is a good idea to close a channel as soon as you finish using it.

The other DDE command available in VCBasic is **DDEAppReturnCode**, which you use for error checking purposes. After getting text, sending text, or executing a command, you might want to use

DDEAppReturnCode to make sure the application performed the task as expected. If an error did occur, your program can notify the user of the error.

Alphabetical List

A	
Abs	Returns the absolute value of a number
AppActivate	Activate another application
AppClassActivate	Activates an application window (dynamic title bar).
Asc	Return an integer corresponding to a character code
Assert	Trigger an error if a condition is false
Atn	Return the arc tangent of a number
B	
Beep	Produce a short beeping tone through the speaker
Begin Dialog	Begin a dialog box definition
Button	Define a button dialog box control
ButtonGroup	Begin definition of a group of button dialog box controls
C	
Call	Transfer control to a subprogram
CancelButton	Define a cancel-button dialog box control
Caption	Define the title of a dialog box
CCur	Convert a value to currency
CDbl	Convert a value to double-precision floating point
ChDir	Change the default directory for a drive
ChDrive	Change the default drive
CheckBox	Define a checkbox dialog box control
Chr	Convert a character code to a string
CInt	Convert a value to an integer by rounding
Class List	List of available classes
Clipboard	Access the Windows Clipboard
CLng	Convert a value to a long by rounding
Close	Close a file
ComboBox	Define a combobox dialog box control
Command	Return the command line specified when the MAIN sub was run
Const	Declare a symbolic constant

Cos	Return the cosine of an angle
CreateObject	Create an OLE2 automation object
CrtAttr	Determine data field type at CRT position.
CrtCls	Clear the emulation screen.
CrtCol	Determine column position from CRT image cell value.
CrtCopy	Copy CRT data to file, printer or clipboard.
CrtEmit	Send data to emulation module.
CrtFieldSearch	Search for start or end of data field.
CrtGet$	Read characters from the emulation screen.
CrtPosition	Convert row,column position to cell value.
CrtQuery$	Query current emulation settings.
CrtRow	Determine row position from CRT image cell value.
CrtSearch	Search for text on emulation screen.
CrtSetCursor	Position the emulation cursor.
CrtTrigger$	Send function keys under program control.
CrtTypeSet$	Query or set the emulation type.
CSng	Convert a value to single-precision floating point
CStr	Convert a value to a string
$CStrings	Treat backslash in string as an escape character as in 'C'
CurDir	Return the current directory for a drive
CVar	Convert an number or string to a variant
CVDate	Convert a value to a variant date
D	
Date Function	Return the current date
Date Statement	Set the current date
DateSerial	Return the date value for year, month, and day specified
DateValue	Return the date value for string specified
Day	Return the day of month component of a date-time value
DDEAppReturnCode	Return a code from an application on a DDE channel
DDEExecute	Send one or more commands to an application on a DDE channel
DDEInitiate	Open a dynamic data exchange (DDE) channel
DDEPoke	Send data to an application on a DDE channel
DDERequest	Return data from an application on a DDE channel
DDETerminate	Close a DDE channel

Declare	Forward declare a procedure in the same module or in a dynamic link library
Deftype	Declare the default data type for variables
Derived Functions	List of computed trigonometric and logarithmic functions
Dialog Function	Display a dialog box and return the command button pressed
Dialog Statement	Display a dialog box
Dim	Declare variables
Dir	Return a filename that matches a pattern
DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Determine whether a dialog control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgEnd	Closes the active dialog box
DlgFocus Function	Return ID of the dialog control having input focus
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgSetPicture	Change the picture in the Picture control
DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with a dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Determine whether a control is visible or hidden
DlgVisible Statement	Show or hide a dialog control
Do...Loop	Control repetitive actions
DoEvents	Let operating system process messages
DropComboBox	Define a drop combobox dialog box control
DropListBox	Define a drop list box dialog box control
E	
Emit	Send "keyboard" data to the I/O module.
EmitBrk	Simulate break signal or break key.
Environ	Return a string from the operating system's environment
Eof	Check for end of file
Erase	Reinitialize contents of an array
Erl	Return the line number where a run-time error occurred

Err Function	Return a run-time error code
Err Statement	Set the run-time error code
Error Function	Return a string representing an error
Error Statement	Generate an error condition
Exit	Cause the current procedure or loop structure to return
Exp	Return the value of *e* raised to a power
F	
FileAttr	Return information about an open file
FileCopy	Copy a file
FileDateTime	Return modification date and time of a specified file
FileLen	Return the length of specified file in bytes
Fix	Return the integer part of a number
For...Next	Loop a fixed number of times
Format	Convert a value to a string using a picture format
FreeFile	Return the next unused file number
FtQuery$	Query current file transfer settings.
FtSet$	Specify new file transfer settings.
FtTrigger$	Invoke special actions for file transfer.
FtTypeSet$	Query or specify file transfer settings.
Function	Define a function
FV	Return the future value for a stream of periodic cash flows
G	
Get	Read bytes from a file
GetAttr	Return attributes of specified file, directory of volume label
GetField	Return a substring from a delimited source string
GetObject	Return the name of an OLE2 object
Global	Declare a global variable
Goto	Send control to a line label
GroupBox	Define a groupbox in a dialog box
H	
Hex	Return the hexadecimal representation of a number, as a string
Hour	Return the hour of day component of a date-time value
I	
If ... Then ... Else	Branch on a conditional value

$Include	Tell the compiler to include statements from another file
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
InputBox	Display a dialog box that prompts for input
InStr	Return the position of one string within another
Int	Return the integer part of a number
IoInput$	Receive data from the I/O module.
IoQuery$	Query current I/O settings.
IoSet$	Specify new I/O settings.
IoTrigger$	Change I/O settings or request I/O module action.
IoTypeSet$	Query I/O settings, or prepare to change settings.
IPmt	Return the interest portion of a loan or annuity payment
IRR	Return the internal rate of return
Is	Determine whether two object variables refer to the same object
IsDate	Determine whether a value is a legal date
IsEmpty	Determine whether a variant has been initialized
IsMissing	Determine whether an optional parameter was supplied to a procedure
IsNull	Determine whether a variant contains a NULL value
IsNumeric	Determine whether a value is a legal number
K	
Kill	Delete files from a disk
L	
LBound	Return the lower bound of an array's dimension
LCase	Convert a string to lower case
Left	Return the left portion of a string
Len	Return the length of a string or size of a variable
Let	Assign a value to a variable
Like Operator	Compare a string against a pattern
Line Input	Read a line from a sequential file
ListBox	Define a list box dialog box control
Loc	Return current position of an open file
Lock	Control access to some or all of an open file by other processes
Lof	Return the length of an open file
Log	Return the natural logarithm of a value

Lset	Left-align one string or user-defined variable within another
LTrim	Remove leading spaces from a string
M	
Me	Get the current object
Mid Function	Return a portion of a string
Mid Statement	Replace a portion of a string with another string
Minute	Return the minute component of a date-time value
MkDir	Make a directory on a disk
Month	Return the month component of a date-time value
MsgBox Function	Display a Windows message box
MsgBox Statement	Display a Windows message box
N	
Name	Rename a disk file
New	Allocate and initialize a new OLE2 object
$NoCStrings	Tell the compiler to treat a backslash as a normal character
Nothing	Set an object variable not to refer to an object
Now	Return the current date and time
NPV	Return the net present value of an investment
Null	Return a null variant
O	
Object	Declare an OLE2 automation object
Oct	Return the octal representation of a number, as a string
OKButton	Define an OK button dialog box control
On...Goto	Branch to one of several labels depending upon value
On Error	Control run-time error handling
Open	Open a disk file or device for I/O
OptionButton	Define an OptionButton dialog box control
OptionGroup	Begin definition of a group of OptionButton dialog box controls
Option Base	Declare the default lower bound for array dimensions
Option Compare	Declare the default case sensitivity for string comparisons
Option Explicit	Force all variables to be explicitly declared
P	
PasswordBox	Display a dialog box that prompts for input. Don't echo input.
Picture	Include a bitmap picture (.BMP file) in a dialog box

Pmt	Return the periodic payment for a loan or annuity
PPmt	Return the principal paid on a loan or annuity
Print	Print data to a file or to the screen
PushButton	Define a push button dialog box control
Put	Write data to an open file
PV	Return the present value for a stream of cash flows
R	
Randomize	Initialize the random-number generator
Rate	Return the interest rate for a loan or annuity
ReDim	Declare dynamic arrays and reallocate memory
Rem	Treat the remainder of the line as a comment
Reset	Close all open disk files
Resume	End an error-handling routine
Right	Return the right portion of a string
RmDir	Remove a directory from a disk
Rnd	Return a random number
Rset	Right-align one string within another
RTrim	Remove trailing spaces from a string
RunMacro	Run another macro program from within current macro.
S	
Second	Return the second component of a date-time value
Seek Function	Return the current position for a file
Seek Statement	Set the current position for a file
Select Case	Execute one of a series of statement blocks
SendKeys	Send keystrokes to another application
Set	Set an object variable to a value
SetAttr	Set attribute information for a file
SetField	Replace a substring within a delimited target string
Sgn	Return a value indicating the sign of a number
Shell	Run an executable program
Shutdown	Shutdown (terminate) OutsideView.
Sin	Return the sine of an angle
Space	Return a string of spaces
Spc	Output given number of spaces

SQLClose	Close a data source connection
SQLError	Return a detailed error message ODBC functions
SQLExecQuery	Execute an SQL statement
SQLGetSchema	Obtain information about data sources, databases, terminology, users, owners, tables, and columns
SQLOpen	Establish a connection to a data source for use by other functions
SQLRequest	Make a connection to a data source, execute an SQL statement, return the results
SQLRetrieve	Return the results of a select that was executed by SQLExecQuery into a user-provided array
SQLRetrieveToFile	Return the results of a select that was executed by SQLExecQuery into a user-specified file
Sqr	Return the square root of a number
Static	Define a static variable or subprogram
StaticComboBox	Define a combination of a list box and text box in a dialog box
Stop	Stop program execution
Str	Return the string representation of a number
StrComp	Compare two strings
String	Return a string consisting of a repeated character
Sub	Define a subprogram

T

Tab	Move print position to the given column
Tan	Return the tangent of an angle
Text	Define a line of text in a dialog box
TextBox	Define a text box in a dialog box
Time Function	Return the current time
Time Statement	Return the current time
Timer	Return the number of seconds since midnight
TimeSerial	Return the time value for hour, minute, and second specified
TimeValue	Return the time value for string specified
Trappable Errors	A list of errors trapped by VCBasic code
Trim	Remove leading and trailing spaces from a string
Type	Declare a user-defined data type
Typeof	Check the class of an object

U

	UBound	Return the upper bound of an array's dimension
	UCase	Convert a string to upper case
	Unlock	Control access to some or all of an open file by other processes
V		
	Val	Convert a string to a number
	VarType	Return the type of data stored in a variant
W		
	WaitCrtCursor	Wait for cursor to appear at specific position.
	WaitCrtUnlock	Wait for keyboard to unlock.
	WaitDCD	Wait for carrier detect.
	WaitKeystrokes	Wait for specified number of keystrokes.
	WaitSilent	Wait for inactivity (I/O idle).
	WaitStr	Wait for strings in the emulation data stream
	WaitTime	Wait for a specific time period.
	Weekday	Return the day of the week for the specified date-time value
	While ... Wend	Control repetitive actions
	Width	Set output-line width for an open file
	With	Execute statements on an object or a user-defined type
	Write	Write data to a sequential file
Y		
	Year	Return the year component of a date-time value

Events

You can define the behavior of VCBasic graphical controls for the following events:

Activate
Change
Click
Common
DblClick
Deactivate
DragDrop
DragOver
EditChange
GotFocus
KeyDown
KeyPress
KeyUp
Load
LostFocus
MouseMove
MouseUp
Resize
RightClick
Scroll
Timer
Unload

Methods

You can use the following VCBasic methods to define the behavior of VCBasic graphical controls:

- AddItem
- CanUndo
- Clear
- DeleteString
- Directory
- Drag
- EmptyUndoBuffer
- FindString
- FindStringExact
- FormatLines
- GetData
- GetFormat
- GetLineFromChar
- GetLineText
- GetSel
- GetSelCount
- GetText
- InsertString
- Load
- LoadCursor
- LoadPicture
- Move
- Refresh
- ReplaceSelection
- ScrollText
- SelectString
- SelItemRange
- SetCaretIndex
- SetData
- SetFocus
- SetReadOnly
- SetSel
- SetSelection
- SetText
- Undo
- UnloadForm
- ZOrder

Properties

You can use the following VCBasic properties to define the appearance and behavior of VCBasic graphical controls

- Alignment
- AutoSize
- BackColor
- BorderStyle
- Cancel
- Caption
- Columns
- ColWidth
- CurSel
- FontItalic
- FontName
- FontSize
- FontStrikeThru
- FontUnderline
- ForeColor
- FormHeight
- FormWidth
- HasCaptio
- Left
- Max
- MaxButton
- MaxLength
- Min
- MinButton
- MultiLine
- MultiSelect
- Name
- Style
- SysMenu
- TabIndex
- TabStop
- Tag
- Text
- Tiled
- Timer
- Top

	n		
Cursor	Height	PasswordChar	TopIndex
Default	HelpFileName	Picture	Value
DragCursor	HelpID	PictureCrop	Visible
DragMode	HideSelection	PictureJustify	Width
Enable	Hwnd	ScrollBars	WindowState
ExpandTabs	Icon	SmallChange	WordWrap
FontBold	LargeChange	Sorted	

Data Types and Expressions

Application Data Types (ADTs)

Application Data Types are specific to each application that embeds VCBasic. ADT variables have the appearance of standard VCBasic records. The main difference is that they can be dynamic; creating, modifying, or querying the ADT or its elements will cause application-specific actions to occur. ADT variables and arrays are declared just like any other variable, using the **Dim** or **Global** statement.

Data Type Conversions

Visual CommBasic 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 might 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 (e.g., when converting from **Double** to **Single**, or from either float type to either integer type).

Visual CommBasic 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 contents as the fixed string will be created. When converting from a dynamic string to a fixed string, some adjustment might be required. If the dynamic string is shorter than the fixed string, the resulting fixed string will be extended with spaces. If the dynamic string is longer than the fixed string, the resulting fixed string will be a truncated version of the dynamic string. No runtime errors are caused by string conversions.

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

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

Dynamic Arrays

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

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

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

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

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

Expressions

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

Expressions are evaluated according to precedence order. Operators with higher precedence are evaluated before operators with lower precedence. Operators with equal precedence are evaluated from left to right. Parentheses can be used to override the default precedence; operators within parentheses will be evaluated before those outside the parentheses.

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

Numeric Operators

String Operators

Comparison Operators

Logical Operators

The following table lists the operators in precedence order from high to low.

Operator	**Description**
^	Exponentiation.
-,+	**Unary minus** and **plus**.
*, /	**Numeric multiplication** or **division**. For division, the result is a **Double**.
\	Integer division. The operands can be Integer or Long.
Mod	**Modulus** or **Remainder**. The operands can be **Integer** or **Long**.
-, +	**Numeric addition** and **subtraction**. The + operator can also be used for string concatenation.
&	**String** concatenation.
>, <, =, <=, >=, <>	**Numeric** or **String comparison**. For numbers, the operands will be 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.
Not	**Unary Not**. Operand can be **Integer** or **Long**. The operation is performed bitwise (one's complement).
And	**And o**perands 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).
.	**Record member**. The left operand must be a record variable, and the right operand must be the name of a field.
( )	Array element.

Variant Data Type

The variant data type can 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 can contain a value of any of the following types:

Type/Name	**Size of Data**	**Range**

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

Any newly-defined Variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty Variant converts to zero when used in a numeric expression, or an empty string in a string expression. You 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.

Formatting Data for Display

Formatting Numbers

See Also

When you use the **Format$** function, numeric values may be formatted as either numbers or date/times. If a numeric expression is supplied and the *fmt* argument is omitted or null, the number will be converted to a string without any special formatting.

Format	Description
Currency	Display the number using a currency symbol as defined in the International section of the Control Panel. Use the thousands separator and display two digits to the right of the decimal separator. Enclose negative value in parentheses. For example: Format$(1234,"Currency") returns "$1,234.00".
Fixed	Display the number with at least one digit to the left and at least two digits to the right of the decimal separator.
General Number	Display the number without thousands separator.
On/Off	Display Off for zero, On for any other number.
Percent	Multiply the number by 100 and display with a percent sign appended to the right; display two digits to the right of the decimal separator.
Scientific	Display the number using standard scientific notation.
Standard	Display the number with the thousands separator and two digits to the right of the decimal separator.
True/False	Display False for zero, True for any other number.
Yes/No	Display No for zero, Yes for any other number.

Formatting Date/Times

See Also

When you use the **Format$** function, both numeric values and variants may be formatted as dates. When formatting numeric values as dates, the value is interpreted according the standard VCBasic 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.

Format	Description
General Date	If the number has both integer and real parts, display both date and time.

Example: 11/8/93 1:23:45 PM
If the number has only integer parts, display it as a date. If the number has only fractional parts, display it as time.

Long Date — Display a Long Date. Long Date is defined in the International section of the Control Panel.

Medium Date — Display the date using the month abbreviation and without the day of the week. Example: 08-Nov-93

Short Date — Display a Short Date. Short Date is defined in the International section of the Control Panel.

Long Time — Display Long Time. Long Time is defined in the International section of the Control Panel and includes hours, minutes, and seconds.

Medium Time — Do not display seconds; display hours in 12-hour format and use the AM/PM designator.

Short Time — Do not display seconds; use 24-hour format and no AM/PM designator.

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

A complete date may be output using the following tokens:

Token	Output
c	The date time as if the *fmt* was: "ddddd ttttt". See the definitions below.
ddddd	The date including the day, month, and year according to the machine's current Short Date setting. The default Short Date setting for the United States is m/d/yy.
dddddd	The date including the day, month, and year according to the machine's current Long Date setting. The default Long Date setting for the United States is mmmm dd, yyyy.
ttttt	The time including the hour, minute, and second using the machine's current time settings. The default time format is h:mm:ss AM/PM.

Finer control over the output is available by including *fmt* tokens that deal with the individual components of the date/time:

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

d	
w	The day of the week as a number (Sunday as 1, Saturday as 7).
ww	The week of the year as a number (1-53).
m	The month of the year or the minute of the hour as a one- or two-digit number. The minute will be output if the preceding token was an hour, otherwise the month will be output.
mm	The month of the year or the minute of the hour as a two-digit number. The minute will be output if the preceding token was an hour, otherwise the month will be output.
mmm	The month of the year as a three-letter abbreviation (Jan-Dec).
mmmm	The month of the year without abbreviation (January-December).
q	The quarter of the year as a number (1-4).
y	The day of the year as a number (1-366).
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (1900-9999).
h	The hour as a one- or two-digit number (0-23).
hh	The hour as a two-digit number (00-23).
n	The minute as a one- or two-digit number (0-59).
nn	The minute as a two-digit number (00-59).
s	The second as a one- or two-digit number (0-59).
ss	The second as a two-digit number (00-59).

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

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

m	between noon and 11:59 PM
A/P	An uppercase A with any hour before noon; an uppercase P with any hour between noon and 11:59 PM.
a/p	A lowercase a with any hour before noon; a lowercase p with any hour between noon and 11:59 PM.
AM PM	The contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM.

Note: ampm is equivalent to AMPM.

Note that any set of characters may be inserted into the output by enclosing them in double quotes. Any single character may be inserted by preceding it with a backslash, "\". See Inserting Characters into the Output String for more details.

Formatting Numbers in Scientific Notation

See Also

When you use the **Format$** function, numbers may be formatted in scientific notation by including one of the following exponent strings in the *fmt* specification: E-, E+, e-, e+.

E	Uppercase E appears in the output.
e	Lowercase e appears in the output.
-	Only negative exponents in the output are preceded by the appropriate sign.
+	All exponents in the output are preceded by the appropriate sign.

The exponent string should be preceded by one or more digit characters. The number of digit characters following the exponent string determines the number of exponent digits in the output.

Examples:

Number	**Fmt**	**Result**
123 456 7.89	###.# #E-00	123.4 6E04
123 456 7.89	###.# #e+#	123.4 6e+4
0.12 345	0.00E -00	1.23E -01

Formatting Strings

See Also

When you use the **Format$** function, strings are formatted by examining the *fmt* specification and transferring one character at a time from the input *expression* to the output string.

By default, formatting will transfer characters working from left to right. The exclamation point (!) format character may be used to change this default. Its presence in the *fmt* specification will cause characters to be transferred from right to left.

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

Character transfer is controlled by the at sign (@) and ampersand (&) characters in the *fmt* specification. These operate as follows:

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

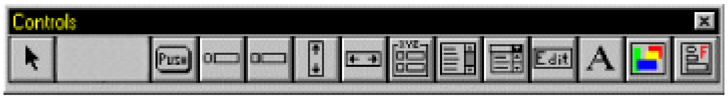
A *fmt* specification for strings can have one or two sections, with the sections separated by a semicolon.

One section   Format specification applies to all string data.

Two sections   The first section applies to string data.
The second section applies to null values and zero-length strings.

Controls and Dialogs

Visual CommBasic Control Reference



Visual CommBasic comes with a set of standard Windows controls that you can use in your applications. Each control has a set of properties, events, and methods that applies to it. For information on a control, select it from the list below.

Check Box

Clipboard

Combo Box

Edit Control

Form

Group Box

Label

List Box

Option Button

Picture Box

Push Button

Scroll Bars

Creating and Modifying Controls

Creating a control

When working with controls, you use the mouse to create, size, and move, and to align controls on a form.

You can also change the properties of a control using the Property Sheet (in design mode) and scripts (in run mode).

To create a control:

1. Select the form on which you want to create the control.

2. Click the appropriate control on the Control Palette. The mouse pointer changes to crosshairs, indicating you are in drawing mode.

3. Use the crosshairs to draw a rectangle that defines the shape of the control on the form. To do this, position the crosshairs and click to create one corner of the rectangle, then drag the mouse to size the control's rectangle, and release the mouse button.

4. The control appears in the rectangle you drew, and the mouse returns to selection mode.

Modifying a control

You can size and move a control in several ways:

♦ Drag it to a new location with the mouse. A grid is available to help you in the alignment of controls (such as buttons).

♦ Drag its handles with the mouse to change its size.

- Use alignment tools on the Toolbar or the Edit menu to change the positions or sizes of several controls relative to each other.

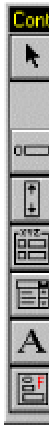
- Change its properties on the Property Sheet.

Use the Script Editor to write a script that changes the properties of a control when an event occurs.

Control Palette

The Control Palette is a set of tools you use when you are designing forms. Each tool, except the mouse-pointer tool, puts the mouse in drawing mode, so you can draw one Visual CommBasic control of the type represented by the tool. Use the mouse-pointer tool to return to selection mode when you are in drawing mode.

The tools on the Control palette are:



To use the control palette to create a control on the active form:

1. Click the appropriate control. The mouse pointer changes to crosshairs, indicating you are in drawing mode.

2. Use the crosshairs to draw a rectangle that defines the shape of the control on the form. To do this, position the crosshairs and click to create one corner of the rectangle, then drag the mouse to size the control's rectangle, and release the mouse button.

3. The control appears in the rectangle you drew, and the mouse returns to selection mode.

Dialog Boxes

See Also

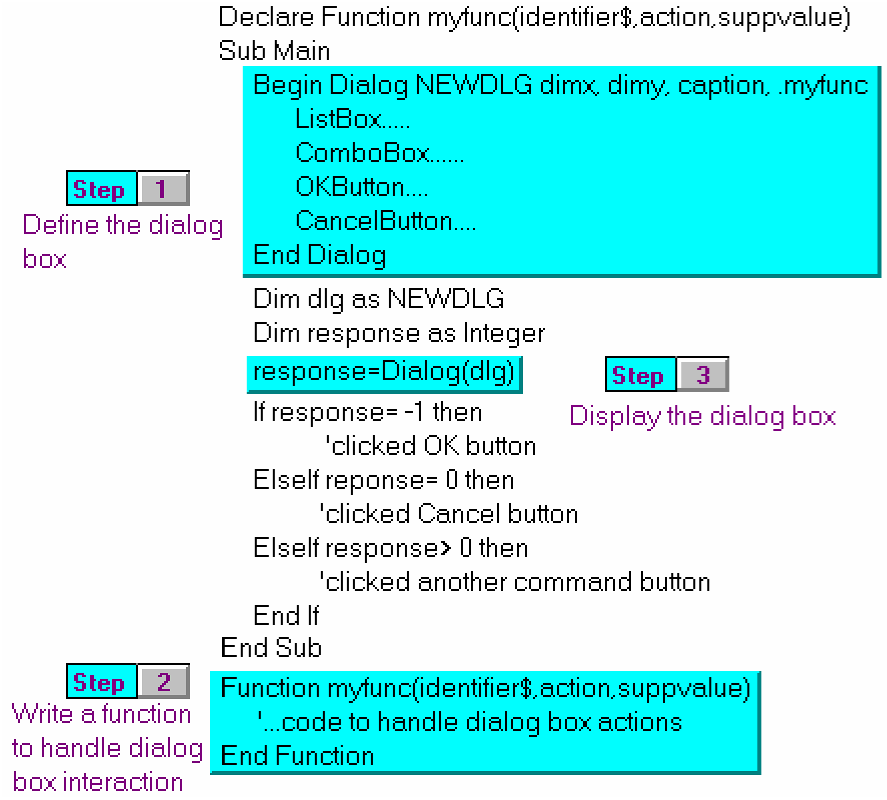
To create and run a dialog box, follow these three steps:

1. Define a dialog box record using the **Begin Dialog...End Dialog** statements and the dialog box definition statements such as **TextBox**, **OKButton**.

2. Create a function to handle dialog box interactions using the **Dialog Functions and Statements**. (Optional)

3. Display the dialog box using either the **Dialog Function** or **Dialog Statement**.

   The example code skeleton below illustrates these steps. Click your mouse over the blue hotspots in this graphic to find out more details.



```
Declare Function myfunc(identifier$,action,suppvalue)
Sub Main
  Begin Dialog NEWDLG dimx, dimy, caption, .myfunc
     ListBox.....
     ComboBox......
     OKButton....
     CancelButton....
  End Dialog
  Dim dlg as NEWDLG
  Dim response as Integer
  response=Dialog(dlg)
  If response= -1 then
       'clicked OK button
  ElseIf reponse= 0 then
      'clicked Cancel button
  ElseIf response> 0 then
      'clicked another command button
  End If
End Sub
Function myfunc(identifier$,action,suppvalue)
  '...code to handle dialog box actions
End Function
```

Dialog Functions and Statements

The function you create uses the "Dlg" dialog functions and statements to manipulate the active dialog box. This is the *only* function that can use these functions and statements.
Dialog functions and statements can be used only when there is an active dialog on the screen; in other words, only the function that was associated with the active dialog in the **BeginDialog** statement (or the VCBasic procedure it called) may call these functions.

This is the list of dialog functions and statements:

Dialog Function	Display a dialog box and return the button pressed
Dialog Statement	Display a dialog box
DlgControlId	Return numeric ID of a dialog control.
DlgEnable Function	Tell whether a control is enabled or disabled.
DlgEnable Statement	Enable or disable a dialog control.
DlgEnd	Close the active dialog box
DlgFocus Function	Return ID of the dialog control having input focus
.DlgFocus Statement	Set focus to a dialog control.

DlgListBoxArray Function	Return contents of a list box or combo box.
DlgListBoxArray Statement	Set contents of a list box or combo box.
DlgSetPicture	Change the picture in the Picture control
DlgText Function	Return the text associated with a dialog control.
DlgText Statement	Set the text associated with a dialog control.
DlgValue Function	Return the value associated with dialog control.
DlgValue Statement	Set the value associated with a dialog control.
DlgVisible Function	Tell whether a control is visible or hidden.
DlgVisible Statement	Show or hide a dialog control.

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

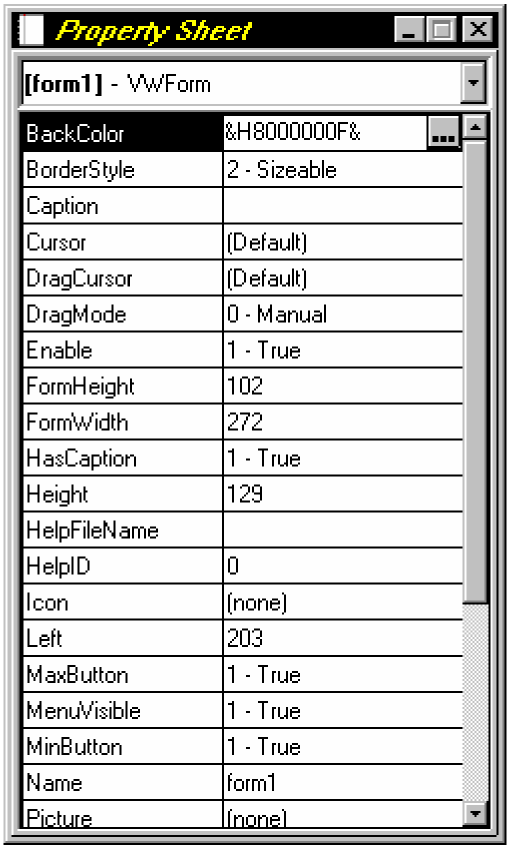
> **CheckBox** 20, 30, 50, 15, "My check box", **.**Check1

Then the **DlgEnable** "**Check1**"**, 1** statement enables the check box, and the **DlgValue(**"**Check1**"**)** function returns 1 if the check box is currently checked, 0 if not. Note that the IDs are case-sensitive and do not include the dot, which appears before the ID. Dialog functions and statements can also work with numeric IDs. Numeric IDs depend on the order in which dialog controls are defined.
For example, if the check box that we considered was the first control defined in the dialog record, the **DlgValue(0)** would be equivalent to **DlgValue(**"**Check1**"**)**. (The control numbering begins from 0, and the Caption control does not count.)

> 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.

Property Sheet

The VCBasic Property Sheet shows the name of the currently selected control on the active form, lists the properties that are available for that control, and shows their current settings. When in Design mode, you can use the Property Sheet to change the property settings of a control.



BackColor	&H8000000F&
BorderStyle	2 - Sizeable
Caption	
Cursor	(Default)
DragCursor	(Default)
DragMode	0 - Manual
Enable	1 - True
FormHeight	102
FormWidth	272
HasCaption	1 - True
Height	129
HelpFileName	
HelpID	0
Icon	(none)
Left	203
MaxButton	1 - True
MenuVisible	1 - True
MinButton	1 - True
Name	form1
Picture	(none)

Note: You can change some of the positional property settings of a control when in Design mode by using its sizing handles, moving it with the mouse, or aligning it with the alignment tools. Use the Script Editor to change the properties of a control during run time.

To change a control's properties by using the Property Sheet:

1. If the Property Sheet is not open, click the control to select it, then click the Property Sheet tool on the Toolbar, or use the menu and select View:Property Sheet.

If the Property Sheet is already open, select the control from the drop-down list box at the top of the Property Sheet window.

2. On the Property Sheet, highlight the property you want to change. A button on the far right side of the highlighted row appears and indicates how you set the property:

... To set one of these properties, click on this button to open a dialog in which you can select the property value. If the property setting is a file, the dialog is a file browser. In other cases, the dialog shows your options or some subset of them (such as a color palette for the BackColor property).

Or, type appropriate text in the edit portion of the row, then click outside the row to establish the new setting. If your entry is invalid, an error message appears.

⬇ To set these properties, click on this button to display a drop-down list of your choices. To select the one you want, click it. Or, remove the drop-down list by clicking outside it.

☑ To set one of these properties, type appropriate text in the edit portion of the row. Then click the checkmark, or move to another field, or press the Enter key to establish the new setting. If your entry is invalid, an error message appears.

Error Trapping and Handling

Error Handling

See Also

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

In addition to the errors trapped by VCBasic, you might 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 VCBasic-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 might 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.

For more information, refer to one of the topics below:

Trapping Errors Returned by VCBasic

Trapping User-defined (Non-VCBasic) Errors

Assert	Trigger an error if a condition is false
Erl	Return the line number where a run-time error occurred
Err Function	Return a run-time error code
Err Statement	Set the run-time error code
Error	Generate an error condition
Error Function	Return a string representing an error
On ErrorControl run-time error handling	

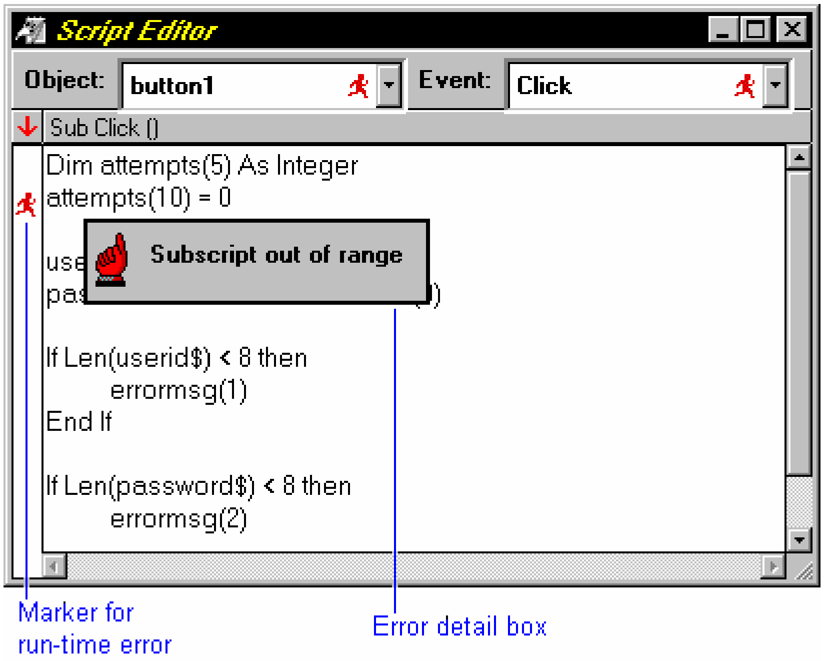
Resume	End an error-handling routine
Trappable Errors	Errors that can be trapped by VCBasic code

Encountering Run-Time Errors

Once a script has been compiled and is running, errors may still occur. Overstepping array bounds, bad file handles, or other such inadvertent or unexpected errors can halt or adversely affect execution. These are called run-time errors.

Run-time errors can be caught through the scripting language with the use of ON-ERROR-GOTO statements, allowing you to define appropriate error handling instructions. You can determine the cause of an error and decide if and how script execution should continue.

Errors that are not caught cause script execution to stop and cause the form to be placed in Design mode. When this happens, the Script Editor window opens, indicating the error line with a red "run" icon, as shown in the following example:



```
Dim attempts(5) As Integer
attempts(10) = 0

use
pas                              )

If Len(userid$) < 8 then
        errormsg(1)
End If

If Len(password$) < 8 then
        errormsg(2)
```

Subscript out of range

Trappable Errors

The following table lists the run-time errors that VCBasic returns. These errors can be trapped by **On Error**. The **Err** function can be used to query the error code, and the **Error$** function can be used to query the error text.

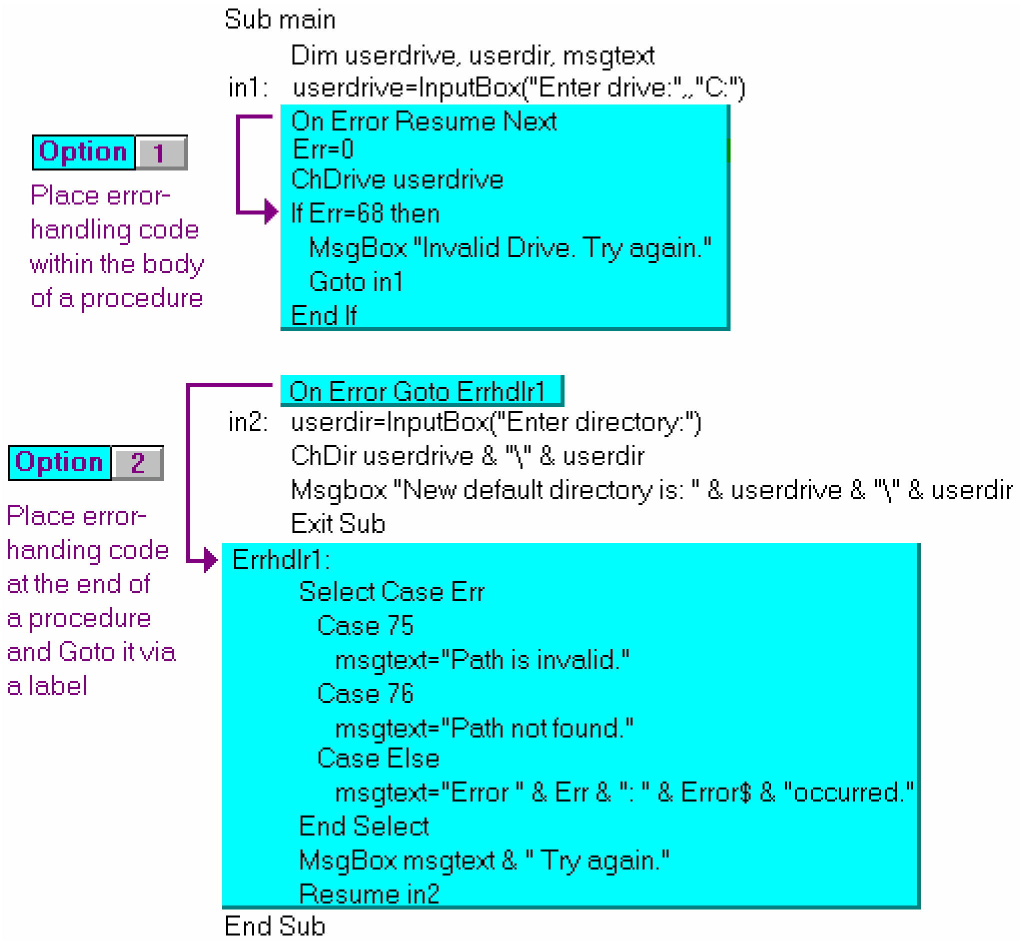
Error Code	Error Text
5	Illegal function call

6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition
11	Division by zero
13	Type mismatch
14	Out of string space
19	No resume
20	Resume without error
28	Out of stack space
35	Sub or function not defined
48	Error in loading DLL
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
58	File already exists
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad file name
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable set to Nothing
93	Invalid pattern
94	Illegal use of NULL
102	Command failed
429	Object creation failed
438	No such property or method
439	Argument type mismatch

440	Object error
901	Input buffer would be larger than 64K
902	Operating system error
903	External procedure not found
904	Global variable type mismatch
905	User-defined type mismatch
906	External procedure interface mismatch
907	Pushbutton required
908	Module has no MAIN
910	Dialog box not declared

Trapping Errors Returned by VCBasic

This code example shows the two ways to trap errors. Option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any error. Click on the blue highlights to get more details.



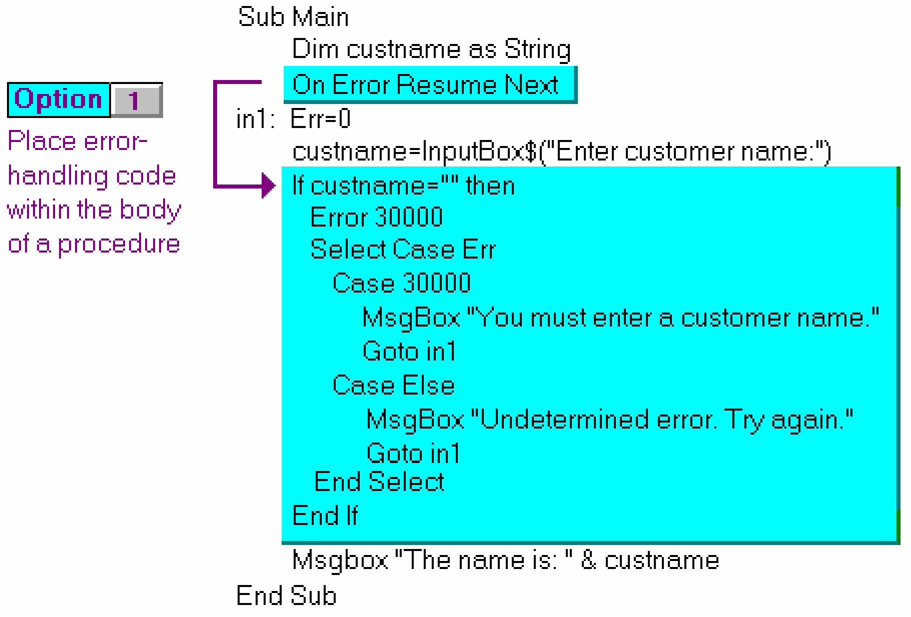
Place error-handling code within the body of a procedure

Place error-handing code at the end of a procedure and Goto it via a label

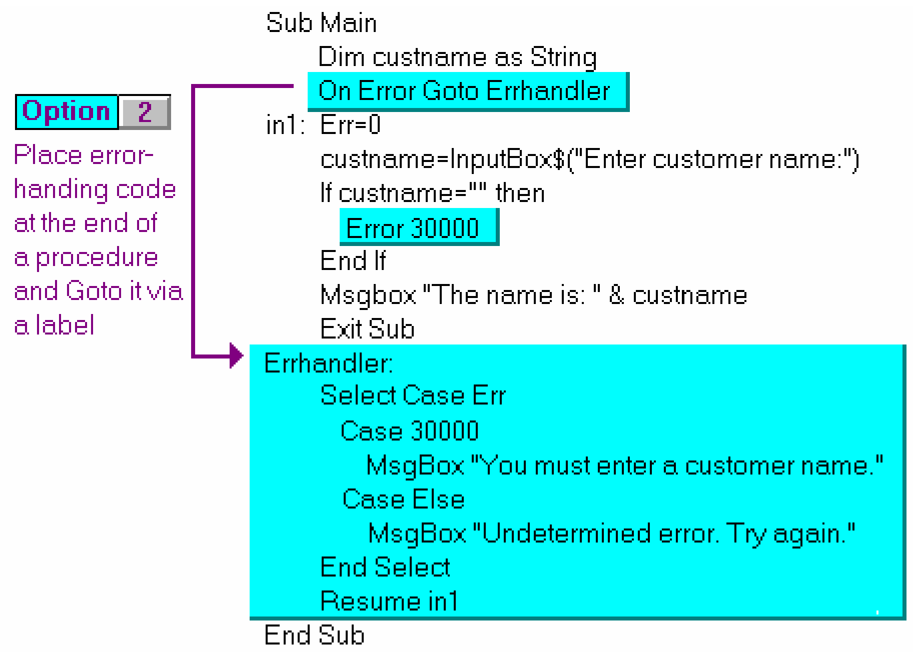
```
Sub main
       Dim userdrive, userdir, msgtext
in1:   userdrive=InputBox("Enter drive:",,"C:")
       On Error Resume Next
       Err=0
       ChDrive userdrive
       If Err=68 then
         MsgBox "Invalid Drive. Try again."
         Goto in1
       End If

       On Error Goto Errhdlr1
in2:   userdir=InputBox("Enter directory:")
       ChDir userdrive & "\" & userdir
       Msgbox "New default directory is: " & userdrive & "\" & userdir
       Exit Sub
Errhdlr1:
       Select Case Err
         Case 75
           msgtext="Path is invalid."
         Case 76
           msgtext="Path not found."
         Case Else
           msgtext="Error " & Err & ": " & Error$ & "occurred."
       End Select
       MsgBox msgtext & " Try again."
       Resume in2
End Sub
```

Trapping User-Defined (Non-VCBasic) Errors

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



```
Sub Main
    Dim custname as String
    On Error Resume Next
in1:  Err=0
    custname=InputBox$("Enter customer name:")
    If custname="" then
      Error 30000
      Select Case Err
        Case 30000
           MsgBox "You must enter a customer name."
           Goto in1
        Case Else
           MsgBox "Undetermined error. Try again."
           Goto in1
      End Select
    End If
    Msgbox "The name is: " & custname
End Sub
```


Option 2
Place error-handing code at the end of a procedure and Goto it via a label

```
Sub Main
    Dim custname as String
    On Error Goto Errhandler
in1:  Err=0
    custname=InputBox$("Enter customer name:")
    If custname="" then
      Error 30000
    End If
    Msgbox "The name is: " & custname
    Exit Sub
Errhandler:
    Select Case Err
      Case 30000
         MsgBox "You must enter a customer name."
      Case Else
         MsgBox "Undetermined error. Try again."
    End Select
    Resume in1
End Sub
```

Visual CommBasic Editor

Menus and Toolbars

Menus and Toolbars

Top level menus

Use the menu commands (or the corresponding tools on the Toolbar) to create an application interface, add controls and specify their properties, write scripts for the controls, test the interface, and debug it.

The menus are:

File	Edit	View	Run	Debug	Window	Help

Toolbar

The tools on the Toolbar are a convenient way to select commonly-used menu commands. The tools are:

File Functions	
🗋	New
📂	Open
Edit Functions	
✂	Cut
📋	Copy
Run Functions	
🏃	Start Run Mode
	End Run Mode
Debug Functions	
	Start Debug Mode
	End Debug Mode
	Single Step
	Show Variables
View Functions	
i	Property Sheet

	Script Editor
Alignment Functions	
	Align Left
	Align Right
	Align Top
	Align Bottom
	Align Horizontally
	Align Vertically
	Stack Vertical
	Stack Horizontal
	Same Width
	Same Height
	Same Size

Debug Menu

Use these commands, or their corresponding Toolbar buttons, to help you debug your scripts.

Toolbar	**Menu commands**
	Start
	End
	Resume
	Single Step
	Show Variables
	Clear Breakpoints

Edit Menu

Use the commands on the Edit menu or the corresponding Toolbar buttons to help you modify the form and its controls:

Toolbar	Menu command
	Undo
	Redo
✂	Cut
⎘	Copy
📋	Paste
	Paste Special...
	Delete
	Find
	Replace
	Bring to Front
	Send to Back
	Alignment Menu

Edit:Alignment Menu

Select two or more controls and use these commands to change their positions relative to each other. These alignment commands can be quickly performed with the Toolbar.

Toolbar	Menu command
⇤	Align Left
⇥	Align Right

	Align Top
	Align Bottom
	Horizontal Space
	Vertical Space
	Stack Vertical
	Stack Horizontal
	Same Height
	Same Width
	Same Size

File Menu

Use the commands on this menu to manage your Visual CommBasic files and choose which control files you want as part of the VCBasic environment. New, Open, and Save can be quickly performed with the Toolbar.

Toolbar	Menu command
	New
	Open
	Close
	Save

Save As...

Save as Text...

Import

Print Form

Recent file list

Exit

Help Menu

Use these commands to display help information about VCBasic.

Index

Using Help

About Visual CommBasic

Run Menu

Use the commands on this menu to switch between Run mode and Design mode.

These functions can be quickly performed with the Toolbar.

Toolbar	**Menu command**
🏃	Start
✎	End

View Menu

Use the commands on this menu to hide or show any of the following.

A checkmark appears beside an item when it is enabled (visible).

Toolbar	**Menu command**
🗎	Script Editor

i	Properties
	Grid Settings...
	Toolbox
	Toolbar
	Status bar
	Always On Top

Window Menu

Use these commands to control the display of the forms and icons in the working window.

Cascade

Tile

Arrange Icons

Close All

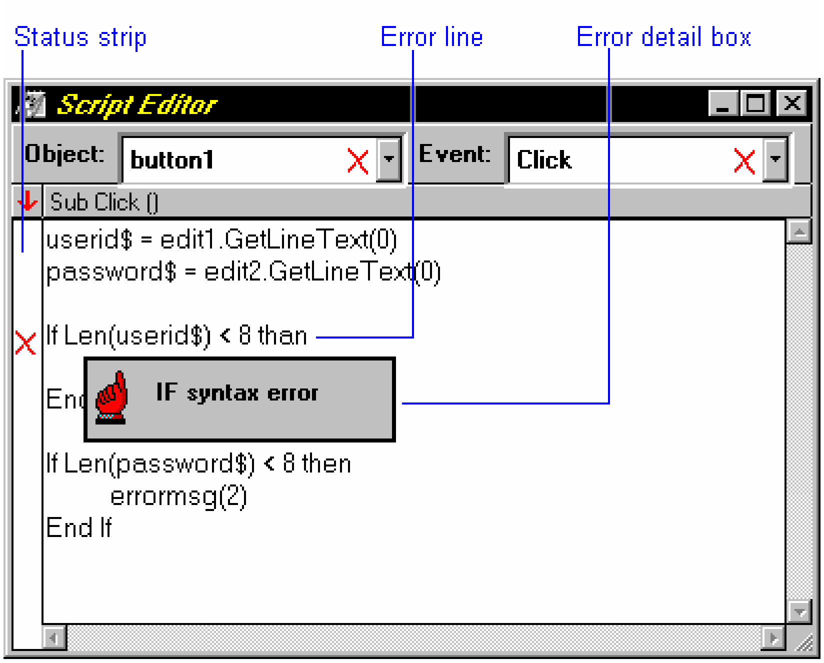
Open Forms List

Debugging

Testing and Debugging an Interface

When you run a form that you are designing, any compile-time errors that are encountered will keep the form in Design mode.

When such errors occur, the Script Editor opens and automatically displays the first script that generated an error and scrolls until the first error-generating line is in view. Error lines are flagged with red X's in the Script Editor window's status strip, as shown in the following example:



Object: button1

Event: Click

```
Sub Click ()
userid$ = edit1.GetLineText(0)
password$ = edit2.GetLineText(0)

If Len(userid$) < 8 than
En
If Len(password$) < 8 then
      errormsg(2)
End If
```

IF syntax error

You can display detailed information about a specific error by clicking on the corresponding X with the right mouse button.

The misspelling of "Next" as "Nxt" in the above example causes two errors. The first error is identified by the error-detail box above. The second error is caused when the script terminates because an expected Next statement is not found. These errors are also identified by error-detail boxes although they may not appear to have an associated script line.

For more information, see Debug Tools.

Debug Tools

Once all compile errors in a script are identified and fixed, the script may still not perform as expected. Errors in control logic and calculations are not caught during the compilation process.

To help you find out why scripts are not performing the way they are expected to, VCBasic provides the following debugging tools:

- Breakpoints
- The STOP Statement
- The Variable Window

Also refer to Runtime Errors.

Setting Breakpoints

You can set or reset breakpoints on any executable script line. A breakpoint is a flag that tells VCBasic to stop at that line while a form is running. When the line is encountered, execution stops and the Script Editor is opened to that line.

To continue line-by-line execution, click the Toolbar button at left or use the Debug:Single Step menu command.

To resume execution (stopping at the next STOP statement or breakpoint), click the Toolbar button at left or use the Debug:Resume menu command.

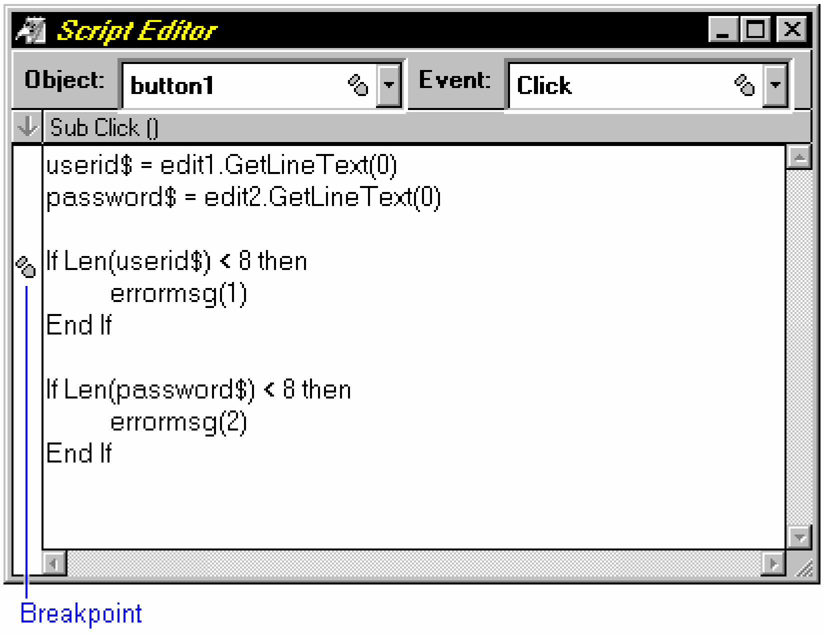
To resume execution in Run mode, click the Toolbar button at left or use the Run:Start menu command.

Breakpoints may be set only on executable lines, not on comments, blank lines, or variable definitions. Breakpoints are only active when the form is run in Debug mode. Forms running in normal Run mode ignore breakpoint settings.

You can set or reset breakpoints on any line of executable code in a script. To do so, position the cursor in the Status strip next to the desired line. When the cursor changes its shape to resemble a fly swatter, click the left mouse button. This toggles the breakpoint on and off for that line.

If you want to clear all breakpoints in all scripts for the active form, use the Debug:Clear Breakpoints menu command.

The following example shows a breakpoint in a script:



Object: button1
Event: Click

```
Sub Click ()
userid$ = edit1.GetLineText(0)
password$ = edit2.GetLineText(0)

If Len(userid$) < 8 then
        errormsg(1)
End If

If Len(password$) < 8 then
        errormsg(2)
End If
```

The STOP Statement

The STOP statement in a script acts as a terminator, halting execution of the script, although no cleanup is performed (the macro program remains in memory, any open files remain open, and variable values are unchanged).

The STOP statement is normally used for compatibility with macros developed under previous versions of CommBasic.

It is recommended that you do not use the STOP statement for new development; if you are attempting to debug the flow and operation of scripts, you should use breakpoints.

Examples

Sample Macros

A number of sample macros are included with Visual CommBasic. These macros are available for your use and show some of the functionality you can provide with VCBasic. Since you have access to the scripts and objects in these sample macros, you can dissect them and see how each is constructed, and perhaps gain a better understanding of macro programming. You may also modify a sample macro to customize it for your environment or operation.

The following macros are included in the MACRO subdirectory:

Macro Name	Description
IOInTest.VCB	This macro uses the IOInput command to intercept a session's I/O stream. • When IOInTest starts, the session's I/O stream will be captured by the macro and not displayed to the screen. • A TACL command (e.g., FILEINFO) may be entered in the edit box labeled "TACL Command". • Pressing the "Get TACL" button will send that command to the session, and any response will be displayed in the large edit box below. The session window will show only the command entered. • The "Clear" button clears the data display. The "Exit" button releases control of the I/O stream and closes the macro.
TACLLog.VCB	This macro allows a TACL logon; logging on to a host is the most-commonly automated task. It waits for ten seconds for the user to enter the first two of the three characters preceeding the prompt and then the ">" character. The macro should work when included on the command line as well as at a TACL prompt.
FTPXfer.VCB	This macro executes a file transfer using FTP at a specified time of day. Only OutsideView needs to be running; the macro creates the necessary FTP session for transferring the specified file. This macro has been rewritten to ensure proper "handshaking" with an NT server and to illustrate the ME alias. A binary

transfer option has also been added.

DDEtoXL.VCB

This example has been rewritten and updated to include examples of error checking and correct use of DDE functions. This macro uses DDE to transfer data between OutsideView and Microsoft Excel. Before you run this macro, start Excel.

Macro Operation:

- Loads and verifies that Excel is running. If Excel is not running, displays error message to user and exits.
- When user clicks Send or Retrieve buttons, sends/retrieves data to/from defined cell in defined sheet.
- If sheet is not loaded, attempts to load sheet.
- If defined sheet cannot be loaded, displays warning message and continues.
- When macro is exited, it closes all open DDE channels.

These macros are shipped with OutsideView, but we are always adding more macros to our library. To see the latest additions, visit our web page or ftp site:

World-Wide Web: http://www.crystalpoint.com/

FTP site: ftp://ftp.crystalpoint.com

Program Examples

We have included several small programs, listed below, that demonstrate the use of VCBasic functions and statements.

Hello World
Simple Basic program that demonstrates calls to subroutines and functions

Bitmap Viewer
Displays a series of bitmap files (.bmp) in a dialog box

Find Files
Finds a test file containing a specified string

Greatest Common Factor
Updates a dialog box dynamically, based on user input

Quicksort
Basic implementation of recursive version of quicksort

Using the Examples

In addition to the definition of each statement or function, the Help System also offers a small working example of each. You will notice the word **Example** next to the words **See Also** in the upper region of the window (under the topic title).

Clicking on **Example** opens a separate window. The Example window contains a small working example of a VCBasic program that uses the given statement or function. You can simply look at the contents of this window, or you can run the example in VCBasic to see how it works.

To run the example, follow these steps:

1. Open a window containing a working version of VCBasic.

2. From the Example window, copy the example to the clipboard (you can copy either part of the example or all of it).

3. Paste the contents into the VCBasic window. (If you copy the whole example, the lines of description will appear as well; however, since each of these lines is preceded by an apostrophe, they function as comments.)

4. Run the program.

To run the examples that show ODBC functions (those beginning with SQL), you will need to have Microsoft Access installed on your machine.

To run the examples that show Object functions, you will need to have VISIO installed on your machine.

🗋 Creates a new form. A new window opens with a blank form in Design mode.

📂 Allows you to open a previously saved VCBasic form by selecting the file in a standard File Open dialog box. If you choose a valid file, VCBasic opens a new window that contains the corresponding form. The form is opened in Design mode.

Closes the currently active form. If the form has been modified since it was last saved, the system warns you. You may choose to save the form before closing it, close the form without saving the changes, or cancel the operation and leave the form open.

💾 Saves the currently active form. If the form was created with the File New command and has not yet been saved, a File Save As dialog box appears so that you can specify a name for the file.

Allows you to save the currently active form by specifying a name for it in a standard File Save dialog box. If the file name already exists, you are warned. You may choose to overwrite the file or cancel the operation.

Allows you to save information about the currently active form as a text file that can be viewed in any text editor.

You choose which of the following information you want to include in the text file:

Control Names	A list of all the controls in the form.
Control Definitions	A list of all the properties and their values for each control.

Control Scripts A listing of all the scripts for each control.

Prints an illustration of the selected form to the currently selected print device.

This section of the File menu lists the last four Visual CommBasic files that have been saved. You may quickly open any of these files by selecting it from the file list.

Exits Visual CommBasic and closes all currently loaded forms. If any form has been modified since it was last saved, the system warns you. You may choose to save the form before exiting, close the form without saving the changes, or cancel the operation and remain in VCBasic leaving the file open.

In Design mode, you can create, move, and resize controls; change their properties, and write scripts for them that will execute when you start Run mode or Debug mode.

Undoes the last editing action done on a form.

Undoes the last undo.

Removes the selected controls from the form, or text from the Property Sheet or Script Editor, and places it on the Windows clipboard.

Copies the selected controls on the form, or text on the Property Sheet or Script Editor, to the Windows clipboard.

Pastes the most recently cut or copied controls from the Windows clipboard into the active form, or text into the Property Sheet or Script Editor.

When there are several different types of data on the Clipboard, this command allows you to choose which one you want to paste into the form.

Removes the selected controls from the form or the selected text from the Property Sheet or Script Editor.

Edit:Find

When the Script Editor is open, this command allows you to find a string of text in the currently active script or all scripts associated with the form.

1. In the **Find** combo box, type the string or choose from the list of strings you have searched for since the last time the Script Editor was opened.

2. Mark the checkboxes that apply to your search. You can choose to match whole words only, match the upper/lower case of the string you type, and/or search all the scripts associated with the active form. If you mark no checkboxes, only the current script in the Script Editor is searched.

3. Click the **Find Next** button to search for the next occurrence of the string.

If no match is found, a tone sounds.

Edit:Replace

When the Script Editor is open, this command allows you to find a string of text in the currently active script or in all scripts associated with the active form, and replace it with another string.

1. In the **Find** combo box, type the string you want to find, or choose from the list of strings you have searched for since the last time the Script Editor was opened.

2. In the **Replace** combo box, type the string you want to replace the string you find, or choose from the list of replacement strings you have used since the last time the Script Editor was opened.

3. Mark the checkboxes that apply to your search. You can choose to match whole words only, match the upper/lower case of the string you type, and/or search and replace the string all the scripts associated with the active form. If you mark no checkboxes, only the current script in the Script Editor is searched.

4. Click the **Find Next** button to search for the next occurrence of the string.

5. Click the **Replace** button to replace the highlighted text with the replacement text.
Or, click the **Replace Next** button to replace the text and then find the next occurrence.
Or, click the **Replace All** button to replace all occurrence in the script (or in all scripts if you have marked that checkbox).

If no match is found, a tone sounds.

Edit:Bring to Front

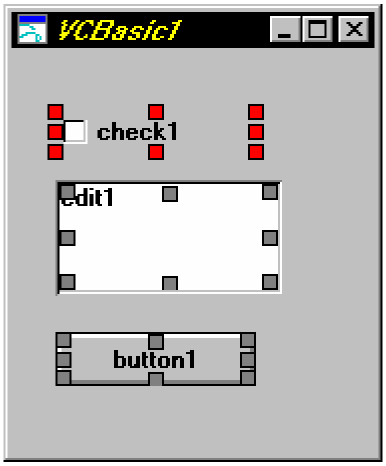
Positions the selected control or controls so they appear to be in front of other controls on the active form. If more than one control is selected, their positions remain the same relative to each other.

Edit:Send to Back

Positions the selected control or controls so they appear to be behind (in back of) other controls on the active form.
If more than one control is selected, their positions remain the same relative to each other.

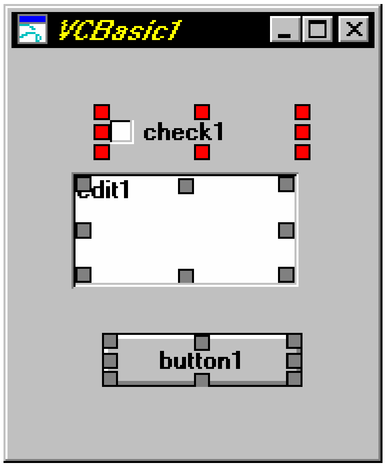
Edit:Alignment:Left

Aligns all the selected controls so their ***left*** edges are even with the left edge of the primary control, as shown in the following example:



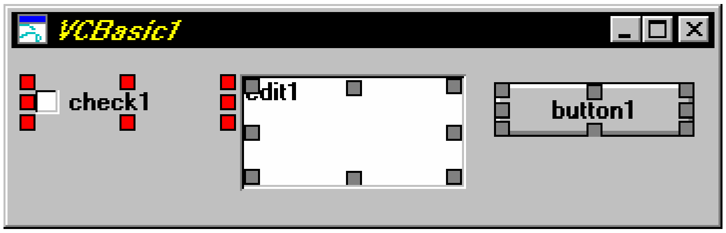
Edit:Alignment:Right

Aligns all the selected controls so their ***right*** edges are even with the right edge of the primary control, as shown in the following example:



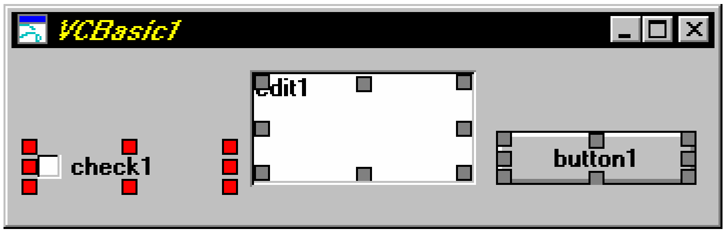
↑↑ Edit:Alignment:Top

Aligns all the selected controls so their ***top*** edges are even with the top edge of the primary control, as shown in the following example:



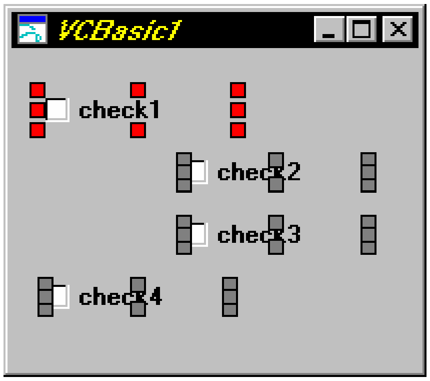
↓↓ Edit:Alignment:Bottom

Aligns all the selected controls so their ***bottom*** edges are even with the bottom edge of the primary control, as shown in the following example:



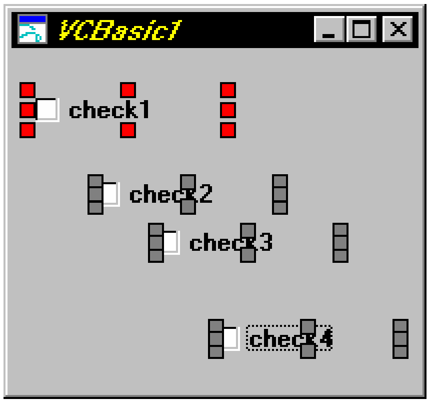
Edit:Alignment:Vertical Space

Positions the selected controls so that they are evenly distributed vertically within the space between the top-most of the selected controls and the bottom-most one, as shown in the following example. The relative order of the controls is unaltered.



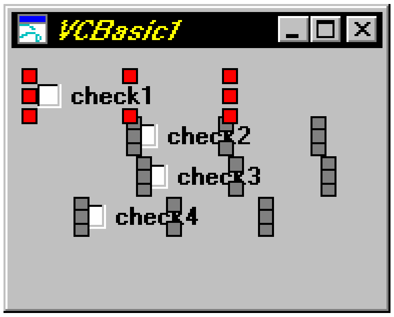
Edit:Alignment:Horizontal Space

Positions the selected controls so that they are evenly distributed horizontally within the space between the left-most of the selected controls and the right-most one, as shown in the following example. The relative order of the controls is unaltered.



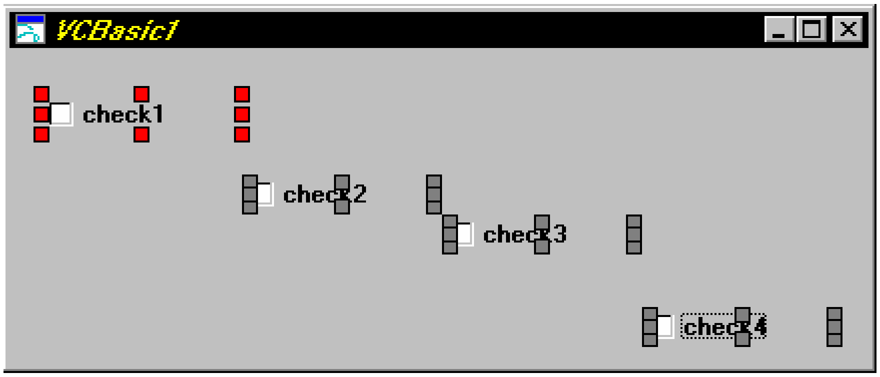
Edit:Alignment:Stack Vertical

Positions the selected controls so that they are stacked vertically one directly on top of the next, with the top of the stack even with the previous position of the top of the primary control, as shown in the following example. The relative order of the controls is unaltered, both horizontally and vertically. The horizontal position of each control is unaltered.



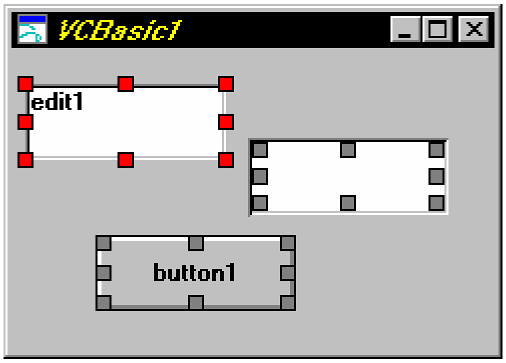
Edit:Alignment:Stack Horizontal

Positions the selected controls so that they are stacked horizontally one directly beside the next, with the left edge of the stack even with the previous position of the left edge of the primary control, as shown in the following example. The relative order of the controls is unaltered, both horizontally and vertically. The vertical position of each control is unaltered.



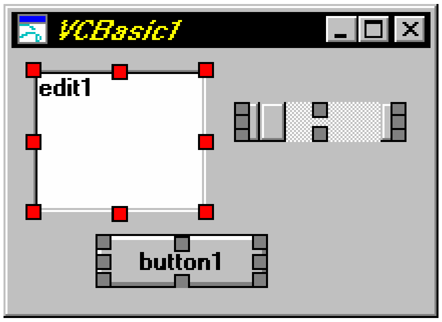
Edit:Alignment:Same Height

In the set of selected controls, changes the height of all secondary controls to the height of the primary control, as shown in the following example:



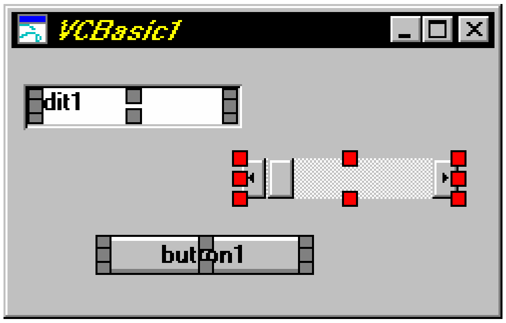
Edit:Alignment:Same Width

In the set of selected controls, changes the width of all secondary controls to the width of the primary control, as shown in the following example:



Edit:Alignment:Same Size

In the set of selected controls, changes the size of all secondary controls to the size of the primary control, as shown in the following example:



View:Grid Settings...

Displays the Grid Settings window in which you can set up a grid of guide-points on the working window. You can use the grid to help you size and align controls when you use the mouse to create or modify them. The grid settings consist of:

X The number of pixels between horizontal grid points.

Y The number of pixels between vertical grid points.

Snap When enabled, controls snap to the grid points when you size or move them with the mouse.

Visible When enabled, grid points are visible in the working window.

View:Toolbar

Use this command to show or hide the bar of command buttons across the top of the window. A Toolbar button is a single click replacement for a menu command.

For more information, see Help topic on the Toolbar.

View:Status Bar

Use this command to show or hide the status bar. The status bar appears at the bottom of the window and gives a brief description of the highlighted menu item, the Toolbar button that is held down, or the control palette button that is held down.

View:Property Sheet

Use this command to show or hide the property sheet, which is a list of properties for the currently selected control.

For more information, see the Help topic on the Property Sheet.

View:Script Editor

Use this command to show or hide the script editor, a window in which you can view or edit a script for the currently selected control.

For more information, see Help topic on the Script Editor.

View:Always On Top

Use this command to keep the Visual CommBasic editor window on top of all other windows on your desktop.

This feature is helpful when testing your macro and you have breakpoints set.

Selection Mode

Selection mode cancels any other palette selection, allowing you to select controls that are already created.

Run:Start

Changes the active form to Run mode, in which you can use the controls on the form just as a user would.

Scripts you have written in Design Mode execute when events occur.

If any scripts contain errors, the Script Editor window opens, and an error message appears pointing to the error.

Run:End

Ends Run mode and switches VCBasic to Design mode.

In Design Mode you can create, move and resize controls; change their properties; and write scripts for them that will execute when you switch to Run mode.

Debug:Start

Starts running the current form in Debug mode.

See Testing and Debugging an Interface for details.

Debug:End

Stops running the current form in Debug mode and returns to Design mode.

Debug:Resume

After execution has stopped at a breakpoint or a STOP statement, this command continues execution and stops at the next breakpoint or STOP statement.

Debug:Single Step

This command allows you to execute the current script a step at a time. Each step proceeds to the next script line in the current script. The line that a script has stopped on is marked with a red exclamation point (!).

Note that this command will step into other scripts that may contain subroutines called by the current script.

To continue running in Debug mode, without stopping at each step, use the Resume command on the Debug menu.

Debug:Show Variables

Turns the Variable Window on or off.

If the Variable Window is turned on, it opens automatically whenever script execution has been suspended due to a breakpoint or a STOP statement. The Variable Window displays:

- All defined script variables.
- The variable type in blue beside the variable name.
- The current value for the variable.

You can edit the values of these variables dynamically in the variable window by typing the new value over the old one. Since the script is still executing when you do, any changes you make to the variables may affect the execution of the script.

The Variable Window only appears when a form is in a suspended state and is processing a breakpoint or a STOP statement. It will always appear during these states until it is turned off.

Debug:Clear Breakpoints

Removes all the breakpoints in all the scripts for the active form. Note that form execution still stops at STOP statements.

Window:Cascade

Arranges the open forms, in front of and offset from the previous form, so you can see the title of each form.

This command also resizes them to a standard size.

Window:Tile

Arranges the open windows in a tiled format in which as much as possible of each form can be seen.

Window:Arrange Icons

If you have minimized any of the open forms, this command arranges their icons in a row along the bottom of the working window.

Window:Close All

Closes all the open forms.

If changes have been made to any of the forms, you are prompted to save them before closing.

Window:Open Forms List

This section of the Window menu is beneath the standard commands and lists all open forms.

You may switch immediately to any of these forms by selecting it from the menu.

Help:Index

Opens the help for Visual CommBasic.

Help:Using Help

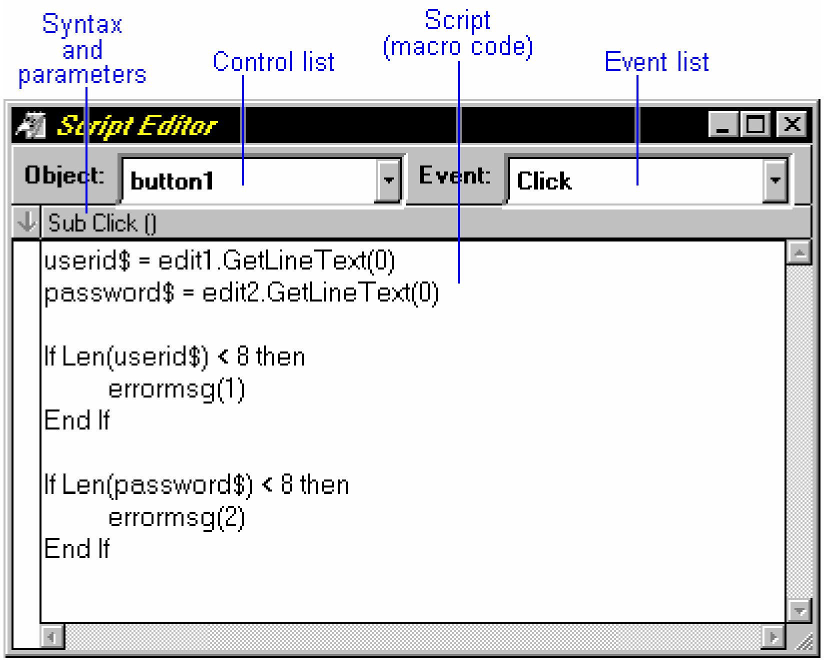
Opens the Windows help file explaining how to use Windows help.

Help:About Visual CommBasic

Opens a window with information about the version of the Visual CommBasic you are using.

Script Editor

The Script Editor is a window in which you can create and edit scripts that execute in run mode.



Object: button1

Event: Click

```
Sub Click ()
userid$ = edit1.GetLineText(0)
password$ = edit2.GetLineText(0)

If Len(userid$) < 8 then
        errormsg(1)
End If

If Len(password$) < 8 then
        errormsg(2)
End If
```

The Script Editor window includes:

- Control and event lists, which allow you to select the control and event for which you want to create a script.

- Syntax and parameter descriptions for the selected event

- A Script Editor for creating and editing scripts. The Script Editor is a standard Windows edit control that supports cut, copy, and paste operations. Search and replace are also supported, either within a script or across all scripts, and are accessible through the Edit menu.

To create or edit a script for a control:

1. If the Script Editor is not open, double-click on the control to open the Script Editor. The control's name is selected in the **Control** list at the top of the editor.

If the Script Editor is already open, select the control from the **Control** drop-down list.

2. In the **Event** drop-down list, select the event for which you want to write or edit a script. The list contains all the events that apply to the selected control. Note that, when you select an event from this list, the syntax and parameters it uses appear in the row just above the edit area of the window.

3. Write the script that you want to execute when the selected event occurs for the selected control. You can include properties (getting and setting), methods, and events in the script.

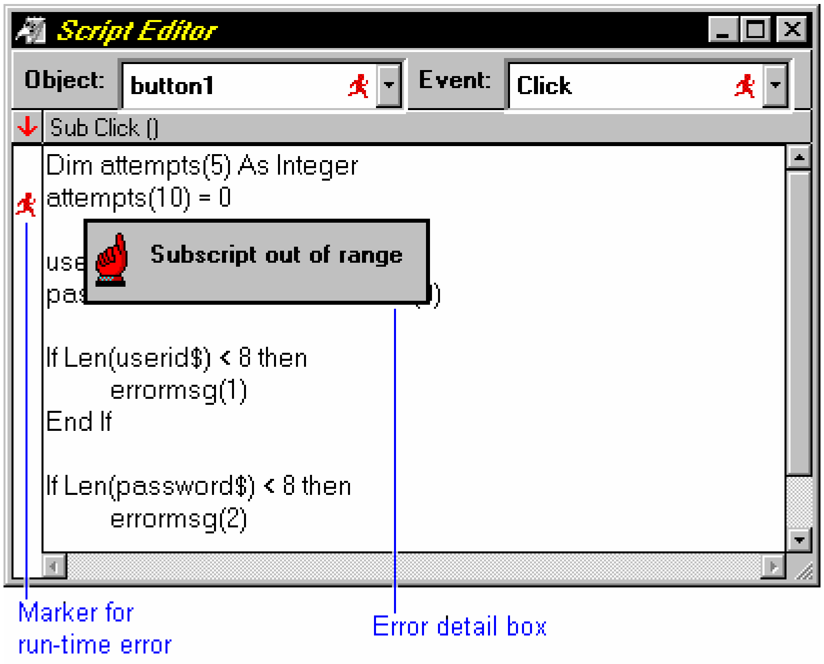
To run the script, change to run mode by clicking this Toolbar button, or use the menu by selecting Run:Start.

If VCBasic finds errors in the script, it returns to Design mode, and marks each error in the script with a red X, and displays an error message about the first error encountered in the script.

The Script Editor Window in Debug Mode

Through the Script Editor Window you can also monitor and control the execution of a procedure when you are running VisualWare in Debug mode.

The status strip on the left side of the window allows you to display and control the debugging process. The status strip lets you set breakpoints, view compile and run errors, and it indicates which is the current line while debugging, as shown in the following example:

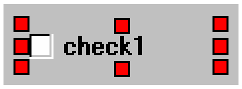


Subscript out of range

```
Dim attempts(5) As Integer
attempts(10) = 0

use
pas                                 )

If Len(userid$) < 8 then
        errormsg(1)
End If

If Len(password$) < 8 then
        errormsg(2)
```

For more information, see Testing and Debugging.

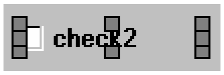
Primary Control



In the set of selected controls, the one that was selected first, and is indicated by red handles.

The secondary controls have brown-grey handles.

Secondary Controls



In a set of selected controls, all those that were not selected first. Secondary controls have brown handles.

The primary control has red handles.

Dialog Box Records

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

See the **Begin Dialog** statement for more information.

Line Continuation

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

Dim *Month* **As Integer _** 'month of transaction

Year **As Integer** ' year of transaction

DDEInitiate Function

See Also **Example**

Opens a dynamic-data exchange (DDE) channel and returns the DDE channel number (1,2, etc.).

Syntax **DDEInitiate**(*appname$* , *topic$*)

Where:	**Is:**
appname$	A string or expression for the name of the DDE application to talk to.
topic$	A string or expression for the name of a topic recognized by *appname$*.

If **DDEInitiate** returns non-zero, the channel was opened. If zero, the channel was unable to be opened.

Appname$ is usually the name of the application's .EXE file without the .EXE filename extension. If the application is not running, **DDEInitiate** cannot open a channel and returns an error. Use **Shell** to start an application.

Topic$* is usually an open filename. If *appname$ doesn't recognize *topic$*, **DDEInitiate** generates an error. Many applications that support DDE recognize a topic named System, which is always available and can be used to find out which other topics are available. For more information on the System topic, see **DDERequest**.

The maximum number of channels that can be open simultaneously is determined by the operating system and your system's memory and resources. If you aren't using an open channel, you should conserve resources by closing it using **DDETerminate**.

User-defined Numeric Formats

See Also

Here are the rules for creating user-defined numeric formats when you use the **Format$** function.

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

Examples:

Number	**Fmt**	**Result**
1234.56	#	1 2 3

		5
1234.56	#.##	1234.56
1234.56	#.#	1234.6
1234.56	######.##	1234.56
1234.56	00000.000	01234.560
0.12345	#.##	.12
0.12345	0.##	0.12

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

Number	Fmt	Result
1234567.8901	#,#.##	1,234,567,89
1234567.8901	#,#.####	1,234,567.8901

Note: While a period (.) is always used in the *fmt* to denote the decimal separator, the output string will contain the appropriate decimal character based upon the current international settings for your machine. Likewise, while a comma (,) is always used in the *fmt* specification, the output will contain the appropriate thousands separator from the current international settings.

Numbers may be scaled 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 (divide) the number by 1000. The commas will not appear in the output string. The percent sign will cause the number to be multiplied by 100. The percent sign will appear in the output string in the same position as it appears in *fmt*.

Number	Fmt	Results
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 being included in the *format* specification. The following characters will be automatically inserted in the output string in a location matching their position in the *format* specification:

- + $ ( ) space : /

Any set of characters can be inserted by enclosing them in double quotes. Any single character can be inserted by preceding it with a backslash, "\". You can use the VCBasic **'$CSTRINGS** metacommand or the **Chr** function if you need to embed quotation marks in a format specification. The character code for a quotation mark is 34.

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

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

E- E+ e- e+

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

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

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

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

Inserting Characters into the Output String

See Also

When you use the **Format$** function, you can have characters inserted into the output string by including them in the *fmt* specification. The following characters are automatically inserted in the output string in locations matching their positions in the *fmt* specification:

(dash)	(close paren)
(plus)	(space)
(dollar sign)	(colon)
(open paren)	(forward slash)

Any set of characters may be inserted by enclosing them in double quotes. Any single character may be inserted by preceding it with a backslash (\). You may wish to use the VCBasic **$CStrings** metacommand or the **Chr** function if you need to embed double quotation marks in a format specification. The character code for double quotes is 34.

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

Sectioning Numeric fmt Specifications

See Also

When you use the **Format$** function, a numeric *fmt* specification can have up to four sections, with the sections separated by semicolons.

One section	It applies to all values.
Two sections	The first applies to positive values and zeros. The second applies to negative values.
Three sections	The first applies to positive values. The second applies to negative values.

	The third applies to zeros.
Four sections	The first applies to positive values. The second applies to negative values. The third applies to zeros. The fourth applies to null values

If you include semicolons with nothing between them, the undefined section is printed using the format of the first section. If the fourth section is omitted and the input expression results in a NULL value, **Format$** will return an empty string.

Examples:

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

GetCurValues Statement

Stores the current values for the application dialog box associated with the specified record

Syntax GetCurValues *recordName*

Where:	**Is:**
recordName	The name of the record. The record must have been previously dimensioned as an application dialog box.

GetObject Function

See Also **Example**

Returns an OLE2 object associated with the file name or the application name.

Syntax A: GetObject(*fileName*)

Syntax B: **GetObject**(*fileName*, *oleClassName*)

Syntax C: GetObject(, *oleClassName*)

Where:	Is:
fileName	The name of the file where the OLE2 object is stored.
oleClassName	The name of the OLE2 object, including the object class in a dot notation.

Check Box Events

Click
DblClick
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
RightClick

Check Box Methods

Drag
LoadCursor
LoadPicture
Move
Refresh
SetFocus
ZOrder

Check Box Properties

Alignment
BackColor
FontName
FontSize
Name
Picture

- Caption
- Cursor
- DragCursor
- DragMode
- Enable
- FontBold
- FontItalic
- FontStrikeThru
- FontUnderline
- ForeColor
- Height
- HelpID
- Hwnd
- Left
- TabIndex
- TabStop
- Tag
- Top
- Value
- Visible
- Width

Clipboard Methods

- Clear
- GetData
- GetFormat
- GetText
- SetData
- SetText

Combo Box Events

- Click
- DblClick
- DragDrop
- DragOver
- EditChange
- GotFocus
- KeyDown
- KeyPress
- KeyUp
- LostFocus
- MouseDown
- MouseMove
- MouseUp
- RightClick

Combo Box Methods

- AddItem
- Clear
- DeleteString
- Directory
- Drag
- FindString
- FindStringExact
- GetText
- InsertString
- LoadCursor
- Move
- Refresh
- SelectString
- SetFocus
- ZOrder

Combo Box Properties

- BackColor
- BorderStyle
- Cursor
- DragCursor
- DragMode
- Enable
- FontBold
- FontItalic
- FontName
- FontSize
- FontStrikeThru
- FontUnderline
- ForeColor
- Height
- HelpID
- Hwnd
- Left
- Name
- Sorted
- Style
- TabIndex
- TabStop
- Tag
- Text
- Top
- Visible
- Width

Edit Control Events

- Change
- GotFocus
- MouseDow

n

Click

DblClick

DragDrop

DragOver

KeyDown

KeyPress

KeyUp

LostFocus

MouseMove

MouseUp

RightClick

Edit Control Methods

CanUndo

Drag

EmptyUndoBuffer

FormatLines

GetLineFromChar

GetLineText

LoadCursor

Move

Refresh

ReplaceSelection

ScrollText

SetFocus

SetReadOnly

SetSelection

Undo

ZOrder

Edit Control Properties

Alignment

BackColor

BorderStyle

Cursor

DragCurso

FontSize

FontStrikeThru

FontUnderline

ForeColor

Height

Name

PasswordChar

ScrollBars

TabIndex

TabStop

r

Drag Mode

Enable

ExpandTabs

Font Bold

FontItalic

Font Name

HelpID

HideSelection

Hwnd

Left

MaxLength

MultiLine

Tag

Text

Top

Visible

Width

Form Events

Activate

Common

Deactivate

DragDrop

DragOver

GotFocus

Load

LostFocus

MouseDown

MouseMove

MouseUp

Resize

Right Click

Timer

Unload

Form Methods

Drag

Load

LoadCursor

LoadPicture

Move

Refresh

SetFocus

ZOrder

UnloadForm

Form Properties

BackColor

BorderStyle

HasCaption

Height

Picture

SysMenu

Caption

Cursor

DragCursor

DragMode

Enable

FormHeight

FormWidth

HelpFileName

HelpID

Icon

Left

MaxButton

MinButton

Name

Tag

Tiled

Timer

Top

Visible

Width

WindowState

Group Box Events

Click

DblClick

DragDrop

DragOver

MouseDown

MouseMove

MouseUp

RightClick

Group Box Methods

Drag

LoadCursor

LoadPicture

Move

Refresh

ZOrder

Group Box Properties

BackColor

BorderStyle

Caption

Cursor

FontName

FontSize

FontStrikeThru

FontUnderline

Picture

PictureCrop

PictureJustify

TabIndex

DragCursor
DragMode
Enable
FontBold
FontItalic
ForeColor
Height
Hwnd
Left
Name
Tag
Tiled
Top
Visible
Width

Label Control Events

Click
DblClick
DragDrop
DragOver
MouseDown
MouseMove
MouseUp
RightClick

Label Control Methods

Drag
LoadCursor
Move
Refresh
ZOrder

Label Control Properties

Alignment
AutoSize
BackColor
Bor
FontBold
FontItalic
FontName
FontSi
Hwnd
Left
Name
TabI

derStyle	ze	ndex
Caption	FontStrikeThru	Tag
Cursor	FontUnderline	Top
DragCursor	ForeColor	Visible
DragMode	Height	Width
Enable	HelpID	WordWrap

List Box Events

Change	GotFocus	MouseDown
Click	KeyDown	MouseMove
DblClick	KeyPress	MouseUp
DragDrop	KeyUp	RightClick
DragOver	LostFocus	

List Box Methods

AddItem	GetSel	Refresh
Clear	GetSelCount	SelectString
DeleteString	GetText	SelItemRang

e

- Directory
- Drag
- FindString
- FindStringExact
- InsertString
- LoadCursor
- LoadPicture
- Move
- SetCaretIndex
- SetFocus
- SetSel
- ZOrder

List Box Properties

- Alignment
- BackColor
- BorderStyle
- Columns
- ColWidth
- CurSel
- Cursor
- DragCursor
- DragMode
- Enable
- FontBold
- FontItalic
- FontName
- FontSize
- FontStrikeThru
- FontUnderline
- ForeColor
- Height
- HelpID
- Hwnd
- Left
- MultiSelect
- Name
- Picture
- Sorted
- TabIndex
- TabStop
- Tag
- Top
- TopIndex
- Visible
- Width

Option Button Events

Click
DblClick
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
RightClick

Option Button Methods

Drag
LoadCursor
LoadPicture
Move
Refresh
SetFocus
ZOrder

Option Button Properties

Alignment
BackColor
Caption
Cursor
DragCursor
FontName
FontSize
FontStrikeThru
FontUnderline
ForeColor
Name
Picture
TabIndex
TabStop
Tag

DragMode

Enable

FontBold

FontItalic

Height

HelpID

Hwnd

Left

Top

Value

Visible

Width

Picture Box Events

Click

DblClick

DragDrop

DragOver

MouseDown

MouseMove

MouseUp

RightClick

Picture Box Methods

Drag

LoadCursor

LoadPicture

Move

Refresh

ZOrder

Picture Box Properties

BackColor

BorderStyle

Cursor

DragCursor

Dra

HelpID

Hwnd

Left

Name

Pictur

TabIndex

Tag

Tiled

Top

Visibl

- gMode
- Enable
- Height
- e
- PictureCrop
- PictureJustify
- e
- Width

Push Button Events

- Click
- DragDrop
- DragOver
- GotFocus
- KeyDown
- KeyPress
- KeyUp
- LostFocus
- MouseDown
- MouseMove
- MouseUp
- RightClick

Push Button Methods

- Drag
- LoadCursor
- LoadPicture
- Move
- Refresh
- SetFocus
- ZOrder

Push Button Properties

- Alignment
- BackColor
- Cancel
- Caption
- Cursor
- Default
- DragCursor
- DragMode
- FontItalic
- FontName
- FontSize
- FontStrikeThru
- FontUnderline
- ForeColor
- Height
- HelpID
- Left
- Name
- Picture
- TabIndex
- TabStop
- Tag
- Top
- Visible

Enable
FontBold
Hwnd
Width

Scroll Bar Events

DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
RightClick
Scroll

Scroll Bar Methods

Drag Method
Load Cursor
Move
Refresh
SetFocus
ZOrder

Scroll Bar Properties

Cursor
DragCursor
DragMode
Enable
Height
HelpID
Hwnd
LargeChange
Left
Max
Min
Name
SmallChange
TabIndex
TabStop
Tag
Top
Value
Visible
Width

Activate Event

Applies to...

When the form becomes active, this event fires.

Syntax Sub Control_Activate()

A form becomes active if it is not currently active and the user clicks on any portion of it, or if it gains focus due to some other application closing or giving it focus.

Change Event

Applies to...

This event fires when the Text of the edit control changes or when the current selection of the list box changes.

Syntax Sub Control_Change()

This event is fired only if the change is the result of user interaction. If the change is the result of a property being set, this event is not fired.

Click Event

Applies to...

When the control is clicked with the mouse, this event fires.

Syntax Sub Control_Click()

Common Event

Applies to...

A non-executable event.

Syntax Not applicable.

The **Common** event's script is used as an area to store all subroutines, functions, and data declarations and definitions; if stored here, they are accessible to all scripts for the form.

DblClick Event

Applies to...

When the control is double-clicked, this event fires.

Syntax Sub Control_DblClick()

Deactivate Event

Applies to...

When the form becomes inactive, this event fires.

Syntax Sub Control_Deactivate()

A form becomes inactive if another window (another form or another application) becomes active.

DragDrop Event

Applies to...

When the user releases the mouse button over the control during a drag and drop operation, this event fires.

Syntax Sub Control_DragDrop (*source* as Control, *x* As Single, *y* As Single)

Where:	**Is:**
source	The control that initiated the event.

x,*y* Coordinates that specify the location of the cursor. These coordinates are in pixels relative to the upper left corner of the control that was dropped on.

DragOver Event

Applies to...

This event fires every time the cursor changes its position over the control while a drag and drop operation is in progress.

Syntax Sub Control_DragOver (*source* as Control, *x* As Single, *y* As Single)

Where:	**Is:**
source	The control that initiated the event.
x,*y*	Coordinates that specify the location of the cursor. These coordinates are in pixels relative to the upper left corner of the control that was dropped on.
state	One of the following values indicating the current drag and drop state:

Value of ***state***	**Description**
0	The cursor is entering the control.
1	The cursor is leaving the control.
2	The cursor is over the control.

EditChange Event

Applies to...

When the Text in the edit portion of the combo box changes as a result of user input, this event fires.

Syntax Sub Control_EditChange()

GotFocus Event

Applies To

When the control gains the input focus, this event fires.

Syntax Sub Control_GotFocus()

A control can gain the input focus under any of these conditions:

- When the user clicks on the control
- When a form becomes active and the control is the first control to gain focus on the form.
- When it is set by the script language.
- When it is set by another application.

KeyDown Event

Applies to...

This event fires when the user presses a key other than SHIFT, ALT, or CTRL while the control has the input focus.

As long as the key is held down, this event will continue to fire at a rate determined by the keyboard repeat rate.

Syntax Sub Control_KeyDown (*KeyCode* As Integer, *ShiftState* As Integer)

Where:	**Is:**
KeyCode	The ASCII code for the key.
ShiftState	A value indicating the state of the SHIFT, CTRL, and ALT keys.

If more than one of these keys is down, ShiftState is the sum of their values.

Value	**Description**
0	No modifier key is down.
1	SHIFT is pressed.
2	CTRL is pressed.
4	ALT is pressed.

KeyPress Event

Applies to...

When the user presses a key while the control has the input focus, this event fires.

As long as the key is held down, this event will continue to fire at a rate determined by the keyboard repeat rate.

Syntax Sub Control_KeyPress (*KeyCode* As Integer)

Where:	**Is:**
KeyCode	The ASCII code for the key.

KeyUp Event

Applies to...

When the user releases a pressed key while the control has the input focus, this event fires.

Syntax Sub Control_KeyUp (*KeyCode* As Integer, *ShiftState* As Integer)

Where:	**Is:**
KeyCode	The ASCII code for the key.
ShiftState	A value indicating the state of the SHIFT, CTRL, and ALT keys. If more than one of these keys is down, ShiftState is the sum of their values.

Value of ***ShiftState***	**Description**
0	No modifier key is down.

1	SHIFT is pressed.
2	CTRL is pressed.
4	ALT is pressed.

Load Event

Applies to...

When the form is first loaded, this event fires.

Syntax **Sub Form_Load**

This event occurs before the form is visible. The position and properties of the controls on the form can be changed during this event without causing flicker.

LostFocus Event

Applies to...

When the control loses the input focus, this event fires.

Syntax Sub Control_LostFocus()

A control can lose the input focus if any other control gains the input focus or if a window in another application gains the input focus.

MouseDown Event

Applies to...

When the user presses the mouse button down on a control, this event fires.

Syntax Sub Control_MouseDown (*MouseButton* As Integer, *ShiftState* As Integer, *X* As Integer, *Y* As Integer)

Where: **Is:**

X,Y Coordinates that specify the location of the cursor. These coordinates are in pixels relative to the upper left corner of the control.

MouseButton Indicates which mouse button or buttons are pressed down. If more than one button is pressed, MouseButton is the sum of their values.

ShiftState A value indicating the state of the SHIFT, CTRL, and ALT keys. If more than one of these keys is down, ShiftState is the sum of their values.

Value of ***MouseButton***	**Description**
1	Left mouse button pressed.
2	Right mouse button pressed.
4	Middle mouse button pressed.

Value of ***ShiftState***	**Description**

0	No modifier key is down.
1	SHIFT is pressed.
2	CTRL is pressed.
4	ALT is pressed.

MouseMove Event

Applies to...

When the mouse pointer moves over a control, this event fires.

Syntax Sub Control_MouseMove (*MouseButton* As Integer, *ShiftState* As Integer, *X* As Integer, *Y* As Integer)

Where: **Is:**

X,*Y* Coordinates that specify the location of the cursor. These coordinates are in pixels relative to the upper left corner of the control.

MouseButton Indicates which mouse button or buttons are pressed down. If more than one button is pressed, MouseButton is the sum of their values.

ShiftState A value indicating the state of the SHIFT, CTRL, and ALT keys. If more than one of these keys is down, ShiftState is the sum of their values.

Value of *MouseButton*	**Description**
1	Left mouse button pressed.
2	Right mouse button pressed.
4	Middle mouse button pressed.

Value of *ShiftState*	**Description**
0	No modifier key is down.
1	SHIFT is pressed.
2	CTRL is pressed.
4	ALT is pressed.

MouseUp Event

Applies to...

This event fires when the mouse button is released after it was pressed down over the control. Note that this event can be fired for a control even if the cursor is not over that control.

Syntax Sub Control_MouseUp (*MouseButton* As Integer, *ShiftState* As Integer, *X* As Integer, *Y* As Integer)

Where: **Is:**

X,Y Coordinates that specify the location of the cursor. These coordinates are in pixels relative to the upper left corner of the control.

MouseButton Indicates which mouse button or buttons are pressed down. If more than one button is pressed, MouseButton is the sum of their values.

ShiftState A value indicating the state of the SHIFT, CTRL, and ALT keys. If more than one of these keys is down, ShiftState is the sum of their values.

Value of ***MouseButton***	**Description**
1	Left mouse button pressed.
2	Right mouse button pressed.
4	Middle mouse button pressed.

Value of ***ShiftState***	**Description**
0	No modifier key is down.
1	SHIFT is pressed.
2	CTRL is pressed.
4	ALT is pressed.

Resize Event

Applies to...

When the form is resized, this event fires.

Syntax

Sub Form_Resize (*LeftSide* As Integer, *TopSide* As Integer, *RightSide* As Integer, *BottomSide* As Integer)

Arguments are defined by the number of pixels from the upper left corner of the screen to the:

LeftSide	Left side of the form.
TopSide	Top of the form.
RightSide	Right side of the form.
BottomSide	Bottom of the form.

RightClick Event

Applies to...

When the control is clicked on with the right mouse button, this event fires.

Syntax Sub Control_RightClick()

Scroll Event

Applies to...

When a scroll bar control is scrolled by the user, this event is fired.

Syntax Sub Scrollbar_Scroll (*Value* As Integer)

Where:	**Is:**
Value	The new Value property for the scroll bar. It is the position of the scroll bar "thumb".

Timer Event

Applies to...

When the timer for the form counts down to zero, this event fires.

Syntax Sub Form_Timer()

The Timer property for the form can be set to some number of milliseconds. This value is decremented until it reaches zero, and then this event is fired. To have a recurring timer event, you must reset the Timer property for the form from within this event.

Unload Event

Applies to...

When a form is being unloaded, this event fires.

Controls on the form are still instantiated and can be accessed.

Syntax Sub Form_Unload()

AddItem Method

Applies to...

Adds a specified text string to a combo box or list box.

Syntax *control*.AddItem *text$*

Where:	**Is:**
control	The control ID.
text$	The text string that is to be added to the control's list. There is a 64K limit on all strings in a combo box or list box.

If the items in the list are sorted, the string is added to the list in alphabetical order. If the items are not sorted, the string is added to the end of the list.

CanUndo Method

Applies to...

Checks whether there is anything in the undo buffer and whether the most recent change to an edit control can be undone.

Syntax *value%* = *control*.CanUndo

Where:	**Is:**
control	The control ID.
value%	This method returns one of the following values:

Value of *value%*	**Description**
0	FALSE ; can't undo.
<>0	TRUE ; can undo.

Clear Method

Applies to...

Clears the contents of the control.

Syntax *control*.Clear

Where:	**Is:**
control	The name of the control to clear.

DeleteString Method

Applies to...

Deletes an entry in a combo box or list box at the specified index.

Syntax *control*.DeleteString *index%*

Where:	**Is:**
control	The control ID.
index%	The index number of the entry to be deleted. If the index is not in range, this command has no effect.

Directory Method

Applies to...

Fills the combo box or list box with a list of files that match the indicated file-specification pattern.

Syntax *control*.Directory *fileattributes%*, *pattern$*

Where:	**Is:**
control	The control ID.
fileattributes%	Defines the type of files to be included in the control's list. The *fileattributes%* argument may contain the following values. You may sum the values to get combinations of file types.
pattern$	A string that serves as a file-specification pattern for identifying the files to be included in the control's list. The pattern$ argument can contain wildcard characters such as asterisk (*) and percent (%).

Constant	**Value**	**Meaning**
DIR_STANDARD	0x0000	Normal files
DIR_READWRITE	0x0000	Read/write files

DIR_READONLY	0x0001	Read only files
DIR_SYSTEM	0x0004	System files
DIR_HIDDEN	0x0002	Hidden files
DIR_DIRECTORY	0x0010	System files
DIR_ARCHIVE	0x0020	Volume label
DIR_DRIVES	0x4000	Directory
DIR_EXCLUSIVE	0x8000	Exclusivity

To get only files of a specific type, use the DIR_EXCLUSIVE flag in addition to the others. Otherwise, read/write files will also be included. For example, DIR_EXCLUSIVE + DIR_DRIVES will return only the drive letters.

This method does not clear the control's current contents. To do so, use the Clear method.

Drag Method

Applies to...

Controls the drag status of the cursor.

Syntax control.Drag action%

Where:	**Is:**
control	The control ID.
action%	An integer value that indicates the action to perform.

Value of ***action%***	**Description**
0	Cancel drag operation; no drop occurs.
1	Place the cursor in drag mode and start a drag and drop operation. The cursor changes into the control's DragCursor and remains in drag and drop mode until the user releases the mouse or this method is used again with a parameter of "2".
2	End dragging and perform a drop.

EmptyUndoBuffer Method

Applies to...

Clears an edit control's undo buffer and causes the CanUndo method to return a value of "false". The undo buffer contains the text that was in the edit control prior to the most recent change. **EmptyUndoBuffer** prevents the user from undoing a change to the edit control.

Syntax *editcontrol*.EmptyUndoBuffer

Where:	**Is:**
editcontrol	The edit control ID.

FindString Method

Applies to...

Returns the index of the first entry that fully or partially matches a specified pattern in a combo box or list box.

Syntax *index%* = *control*.FindString (*startindex%*, *pattern$*)

Where:	**Is:**
index%	The index of the matching entry. If no match is found, -1 is returned.
control	The control ID.
startindex%	The index where the search is to begin.
pattern$	The pattern to be matched. Characters normally used as wildcards are treated as normal characters.

FindStringExact Method

Applies to...

Returns the index of the first entry that fully matches the specified pattern in a combo box or list box.

Syntax *index%* = *control*.FindStringExact (*startindex%*, *pattern$*)

Where:	**Is:**
index%	The index of the matching entry. If no match is found, -1 is returned.
control	The control ID.
startindex%	The index where the search is to begin.
pattern$	The pattern to be matched. Characters normally used as wildcards are treated as normal characters.

FormatLines Method

Applies to...

Determines how text is returned from an edit control.

Syntax *editcontrol*.FormatLines *value%*

Where:	**Is:**
editcontrol	The edit control ID.
value%	May be one of the following:

Value of *value%*	**Description**
0	Returns text with hard breaks.
Non-0	Returns text without hard breaks.

GetData Method

Applies to...

Returns picture data from the clipboard control and assigns it to the picture property of any control that supports a picture property. To get text data from the clipboard, use the **GetText** method on the clipboard control.

Syntax *control*.Picture = clipboard.GetData (*format%*)

Where:	**Is:**

control Any control that has a picture property.

format% Specifies the type of data to be retrieved from the clipboard. Note that the clipboard can contain multiple formats. You must specify the format that you wish to retrieve. The GetFormat method will allow you to determine if data in a specific format is available on the clipboard. May be one of the following:

Value of ***format%***	**Description**
2	Bitmap (.BMP) file.
3	Metafile (.WMF) file.
8	Device-independent bitmap (DIB) file.

GetFormat Method

Applies to...

Returns an integer indicating whether or not there is an item in the clipboard control that matches the specified format.

Syntax *value%* = clipboard.GetFormat (*format%*)

Value of ***value%***	**Description**
0	FALSE ; there is not a matching format.
Non-0	TRUE ; there is a matching format.

Value of ***format%***	**Description**
1	Text file.
2	Bitmap (.BMP) file.
3	Metafile (.WMF) file.
8	Device-independent bitmap (.DIB) file.

GetLineFromChar Method

Applies to...

Returns the number of the line in the edit control that contains the specified character position.

Syntax *line% = editcontrol*.GetLineFromChar(*charindex%*)

Where:	**Is:**

line% The number of the line with the specified character.

editcontrol The edit control ID.

charindex% An integer that identifies the relative position of the character to be located. *charindex%* is relative to the start of the edit control's contents.

For example, a *charindex%* of "39" specifies the thirty-ninth character from the beginning of the text in the edit control. If the thirty-ninth character occurred on the third line of text, **GetLineFromChar** would return "3".

GetLineText Method

Applies to...

Returns the text in the specified line of the edit control.

Syntax *text$* = *editcontrol*.GetLineText (*line%*)

Where:	Is:
text$	The text on the specified line.
editcontrol	The edit control ID.
line%	The line number of the text to be captured.

GetSel Method

Applies to...

Returns the selection state of the specified list-box item.

Syntax *state%* = *listbox*.GetSel(index%)

Where:	Is:
listbox	The list box ID.
index%	The number of the selection for which the selection state is to be returned.
state%	May be one of the following:

Value	Description
0	The item is not selected.
1	The item is selected.
-1	*index%* is out of range.

GetSelCount Method

Applies to...

Returns the number of selected items in the list box.

Syntax *count%* = *listbox*.GetSelCount

Where: **Is:**

count% For a single-select list box, **GetSelCount** returns:

Value of *count%*	**Description**
0	No item is selected.
<>0	An item is selected.

For a multi-select list box, **GetSelCount** returns:

Value of *count%*	**Description**
0	No item is selected.

n The total number of currently selected items.

listbox The list box ID.

GetText Method

Applies to...

Returns text from an item in a combo box or list box.

Syntax *text$* = *control*.GetText(*index%*)

Where: **Is:**

text$ The returned text. A return value of a null string indicates either that *index%* is out of range or that the specified item actually contains a null string.

control The control ID.

index% The index number of the item within the combo box or list box. *index%* can range from zero to "*listbox*.GetCount-1".

GetText Method

Applies to...

Returns a text string from the clipboard object.

Syntax *string$* = clipboard.GetText

Where:	**Is:**
string$	The returned string.

InsertString Method

Applies to...

Syntax

control.InsertString *index%*, *text$*

Description

Inserts a text string at a specified position within the combo box or list box.

Details

Argument	**Description**
control	The control ID.
index%	The index number of the position at which *text$* is to be inserted. If *index%* is out of range, **InsertString** has no effect.
text$	The text to be inserted at the position specified by *index%*.

Load Method

This method is typically used to load a new form in the current macro program. However, the **Load** method is not supported by Visual CommBasic.

The recommended method of presenting multiple forms is to use the RunMacro statement.

LoadCursor Method

Applies to...

Loads a new cursor from a file into a control's cursor property.

Syntax A: *control*.Cursor = LoadCursor(*filename$*)

Syntax B: *control*.DragCursor = LoadCursor(*filename$*)

Where:	**Is:**
control	The control ID.
filename$	The filename should be a valid icon (.ICO), bitmap (.BMP), or cursor (.CUR) file.

Specifying an invalid file type will clear the cursor. If you want to safely clear the cursor, use **LoadCursor(**" "**)**.

Cursor is the standard cursor display; DragCursor is the cursor displayed while dragging.

LoadPicture Method

Applies to...

Loads a new picture from a file into a control's Picture property.

Syntax *control*.Picture = LoadPicture(*filename$*)

Where:	**Is**
Control	The control ID.
filename$	The file must be in a valid bitmap (.BMP), metafile (.WMF), or icon file (.ICO) format.

Move Method

Applies to...

Moves the control to the specified location and sizes it.

Syntax *control*.Move *left%, top%, width%, height%*

Where:	Is:
control	The control ID.
left%	*Controls other than forms:* The number of pixels between the left side of the control and the left side of the form. *Forms:* The number of pixels between the left side of the form and the left side of the screen.
top%	*Controls other than forms:* The number of pixels between the top of the control and the top of the form. *Forms:* The number of pixels between the top of the form and the top of the screen.
width%	The width of the control in pixels.
height%	The height of the control in pixels.

Refresh Method

Applies to...

Updates the control with any property changes that have been made to it during script execution.

Syntax *control*.Refresh

Where:	Is:
control	The control ID.

Controls are updated automatically after the current script finishes. While a script is executing, controls will not automatically repaint themselves if their properties are changed.

Menus **must** be refreshed after items are added or deleted.

The **Refresh** method, which allows you to update controls during script execution, can be useful when a script runs for an extended period of time.

ReplaceSelection Method

Applies to...

Replaces the currently selected text in the edit control with the specified text.

If no text is selected, the specified text is inserted at the current cursor position.

Syntax *editcontrol*.ReplaceSelection *text$*

Where:	**Is:**
editcontrol	The edit control ID.
text$	The text to be substituted for the currently selected text.

ScrollText Method

Applies to...

Scrolls text in the edit control horizontally and vertically. Positive values scroll down or to the right, negative values scroll up or to the left.

Syntax *editcontrol*.ScrollText *vertical%,horizontal%*

Where:	**Is:**
editcontrol	The edit control ID.
vertical%	The number of lines that text is to be scrolled vertically.
horizontal%	The number of characters that text is to be scrolled horizontally.

SelectString Method

Applies to...

Selects the first item in a combo box or list box that partially or fully matches a specified pattern string. If no match is found, returns -1.

Syntax *control*.SelectString *startindex%, pattern$*

Where:	**Is:**
control	The control ID.
startindex%	The index where the search is to begin.
pattern$	The pattern to be matched. Characters normally used as wildcards are treated as normal characters.

SelItemRange Method

Applies to...

Sets all items in a specified range within a multi-select list box to a specified state. This method is valid only for multi-select list boxes, that is, those whose MultiSelect property is non-zero.

Syntax *listbox*.SelItemRange *state%,startindex%,endindex%*

Where:	**Is:**
listbox	The list box ID.
startindex%	The starting index in the range of items to be changed.
endindex%	The ending index in the range of items to be changed.
state%	The state to which all items within the range are to be set.

Value of *state%*	**Description**
0	Items are not selected.
Non-0	Items are selected.

SetCaretIndex Method

Applies to...

Places the focus rectangle on a specified list-box item. If the specified item is out of range, **SetCaretIndex** has no effect.

Syntax *listbox*.SetCaretIndex *index%*

Where:	**Is:**
listbox	The list box ID.
index%	The index of the item to receive the focus rectangle.

SetData Method

Applies to...

Puts a picture in the clipboard control. You must also specify the format of the data. Note that multiple formats are supported, and that this method may be used multiple times to put up different formats on the clipboard at the same time.

Syntax A: clipboard.SetData *control*.Picture, *format%*

Syntax B: clipboard.SetData LoadPicture(*filename$*), *format%*

Where:	Is:
control	Any control that has a picture property.
filename$	If you use Syntax B to load a picture from a file into the clipboard, the format of *filename$* must match the format indicated by *format%*.
format%	The format of the picture. May be one of the following:

Value of *format%*	Description
2	Bitmap (.BMP) file.
3	Metafile (.WMF) file.
8	Device-independent bitmap (.DIB) file.

SetFocus Method

Applies to...

Places the focus on the specified control. If the Enable property for the control is zero, **SetFocus** has no effect.

Syntax *control*.SetFocus

Where:	Is:
control	The control ID.

SetReadOnly Method

Applies to...

Specifies whether or not the status of an edit control is read-only.

Syntax *editcontrol*.SetReadOnly *readonly%*

Where:	**Is:**
editcontrol	The edit control ID.
readonly%	May be one of the following:

Value of *readonly%*	**Description**
0	Sets the edit control's status to read-write.
Non-0	Sets the edit control's status to read-only.

SetSel Method

Applies to...

Sets a specified item in a multiple-select list box to a specified selection state.

If the specified item is out of range, **SetSel** has no effect. This property is **only** valid for multiple-select list boxes.

Syntax *listbox*.SetSel *index%, state%*

Where:	**Is:**
listbox	The list box ID.
index%	The index of the item to be affected.
state%	The state to which the specified item is to be set. May be one of the following:

Value of *state%*	**Description**
0	De-selects the item.
Non-0	Selects the item.

SetSelection Method

Applies to...

Selects text in an edit control.

Syntax *editcontrol*.SetSelection *start%, stop%, flag%*

Where:	**Is:**
editcontrol	The edit control ID.
start%	The relative position in the edit control of the first text character to be selected. If *start%* is "-1", no text is selected.
stop%	The relative position in the edit control of the last text character to be selected.
flag%	Specifies whether the caret (which is placed at the end of the selection) will be scrolled into view.

If *start%* is "0" and *stop%* is "-1", **all** text is selected.

SetText Method

Applies to...

Puts a text string into the clipboard control.

Syntax clipboard.SetText(*text$*)

Where:	**Is:**
text$	The text string.

Undo Method

Applies to...

Undoes the last change to the text in the edit control. If the undo buffer is empty, **Undo** has no effect.

Syntax *editcontrol*.Undo

Where:	**Is:**
editcontrol	The edit control ID.

UnloadForm Method

Applies to...

Unloads the current form, ending the macro.

Syntax UnloadForm me

The argument "me" is the alias of the current form.

This method does not work with any **WAIT** statement.

An example use of the UnloadForm method is to set it as the Click event for an Exit push button.

ZOrder Method

Applies to...

Positions a control in front of or behind other controls on a form.

Syntax *control*.ZOrder *value%*

Where:	Is:
control	The control ID.
value%	Specifies the direction of the change. May be one of the following:

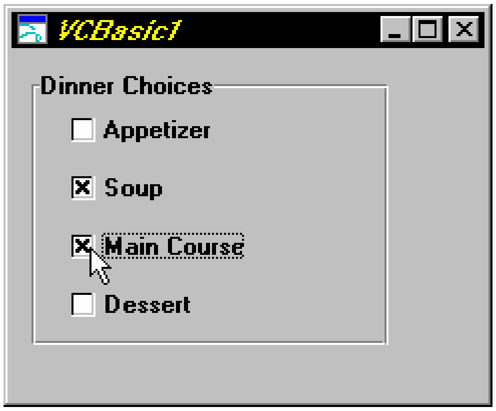
Value of *value%*	Description
0	Brings the control to the front.
Non-0	Sends the control to the back.

Check Box

Properties **Events** **Methods**

Description

The check box control provides you with a non-mutually exclusive "yes or no" choice about a particular option in your application. The following example allows a user to select various dinner choices by grouping together a set of related check box controls. For allowing selection of mutually exclusive options, see the Option Button control.



Dinner Choices

☐ Appetizer

☒ Soup

☒ Main Course

☐ Dessert

You can select a check box by clicking it with the mouse or by moving the cursor to the check box and pressing the space bar. When a check box is selected, an X appears inside the box. When a check box is not selected, the box is empty.

Clipboard

Methods

Description

The clipboard control allows a user to copy, cut, and paste text or graphics into an application.

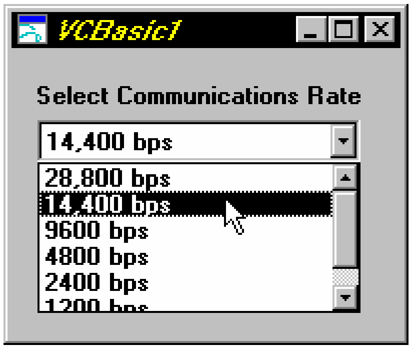
The clipboard control can contain multiple pieces of data and/or text at the same time if each piece of data is in a **different** format. Data of the same format will replace data already on the clipboard.

Combo Box

Properties **Events** **Methods**

Description

A combo box combines the features of an edit box and a list box. Use a combo box to give the user the choice of typing as in an edit box or selecting an item from as in a list box.



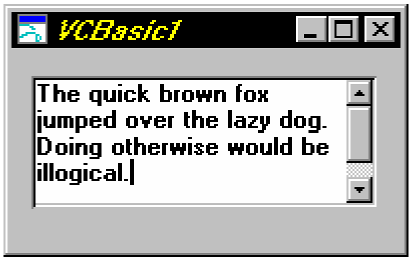
You can select an item in a combo box by clicking on the drop down arrow and using the mouse to scroll to the item. You can also use the keyboard to select an item. Combo boxes can also have other styles, which include the ability to type new entries into the edit portion or having the drop-down area always in its down state.

Edit Control

Properties Events Methods

Description

An edit control allows the user to view and edit textual information. The edit control can be configured to handle single-line or multi-line text, with an array of options to control color, font, word wrapping and password input.



The quick brown fox jumped over the lazy dog. Doing otherwise would be illogical.

The above example edit control has been set up to handle multi-line word-wrapped text.

Form Control

Properties **Events** **Methods**

Description

The form control is always created with each form. It provides an interface to the overall behavior of the form. It also acts as a container control so that other controls may be placed on the form.

The use of the following properties for a form control are highly dependent on the developer. If the form is a child window, you should query the value of these properties and apply them to the parent window of the form.

Property	Recommendation If Used
Caption	Set the parent window's caption (VWM_SETWINDOWTEXT).
HasCaption	Create a parent window with a caption bar.
Icon	Set the icon to the parent window.
MaxButton	Create a parent window with a max button.
MinButton	Create a parent window with a min button.
SysMenu	Create a parent window with a system menu.

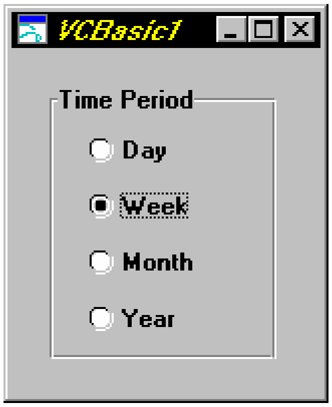
Note: In the VCBasic Editor, the HasCaption, MinButton, and MaxButton properties are ignored because the form's parent is an MDI window (these always have caption, min and max buttons, and system menus).

Group Box

Properties **Events** **Methods**

Description

A group box allows a group of related controls to be visually and physically grouped together on a form. As a physical container, the group box, when moved, will also move all of its children. As a logical container, its child options buttons will coordinate their behavior, allowing only one child option button to be on at any one time.



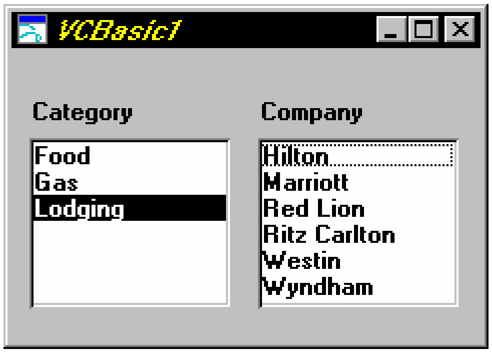
The above example group box allows the user to select a time period. Only one of the options can be selected at a time.

Label Control

Properties **Events** **Methods**

Description

The label control allows read-only text to be displayed on a form. The label provides control over font, color, and word wrapping, but it allows no editing, nor does it allow focus to be set to it.



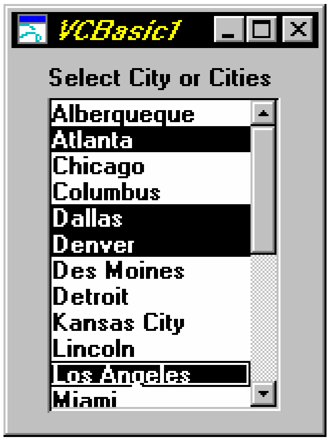
The above example shows how label controls add clarity to a form. Since each list box is labeled, the user of the form can understand what is being selected.

List Box

Properties Events Methods

Description

A list box provides an easy way to present the user with a set of choices. Use a list box to display a list of items from which the user can select one or more. If the number of items exceeds what can be displayed, a scroll bar is automatically added to the list box.



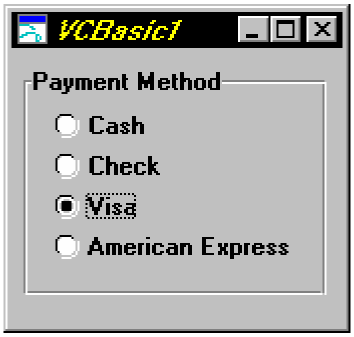
The above example list box displays the set of available cities. The user can use either the mouse or the keyboard to navigate through the selection. This list box is set to handle multiple selections.

Option Button

Properties **Events** **Methods**

Description

The option button control provides you with a mutually exclusive "yes" or "no" choice about a particular option in your application. All option buttons within a group can have, at most, one choice selected at one time. By selecting one option, all other options are turned off. The following example allows a user to select various payment choices by grouping together a set of related option button controls. For allowing selection of non-mutually exclusive options, see the Check Box control.



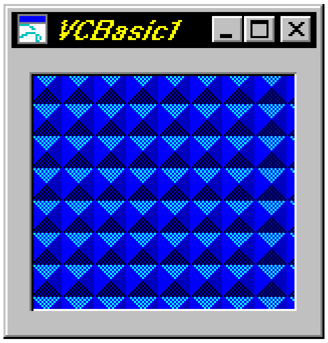
You can select an option button by clicking it with the mouse or by moving the cursor to the associated text and pressing the space bar. When an option button is selected, its circle is filled in. When an option button is not selected, the circle is empty.

Picture Box

Properties **Events** **Methods**

Description

The picture box control is used to display bitmaps or metafiles on a form. These pictures can be set at design time through the Property Sheet, or at run time through a script statement. A variety of formatting techniques are available, including tiling, cropping, and stretching. The picture box control is also a container control, allowing other controls to be nested inside.



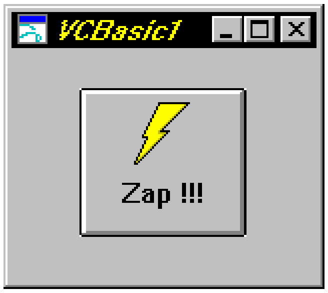
The above example picture control has a small bitmap, which is then tiled to fill its space. Since this control is also a container control, this technique can be used to create a background for a group of controls.

Push Button

Properties **Events** **Methods**

Description

A push button allows the user to click on the button to initiate an action. A push button can also contain bitmaps or icons to clarify the type of action it represents.



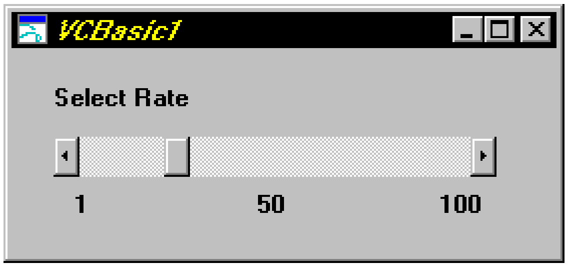
The above example is a standard push button decorated with a bitmap.

Scroll Bar Controls

Properties **Events** **Methods**

Description

The scroll bar controls provide a numeric input/output device that allows users to select parameters without typing in values. The scroll bar can either be horizontal or vertical, and it can have its minimum and maximum values configured.



The above example scroll bar allows the user to select a rate by grabbing the scroll bar "thumb" and dragging it.

Alignment Property

Applies to...

The alignment property gets a value that indicates how a control's text, caption, and/or bitmap are aligned in the control. The specific type of alignment differs for different controls. For the edit control, this property is read-only at run time.

Syntax *align%* = *control*.Alignment

For check box, edit control (multi-line only), list box, and option button, *align%* can be:

Constant	Value	Description
ALIGN_LEFT	0	Text is aligned left.
ALIGN_RIGHT	1	Text is aligned right.

For label, *align%* can be:

Constant	Value	Description
ALIGN_LEFT	0	Text is aligned left.
ALIGN_RIGHT	1	Text is aligned right.
ALIGN_CENTER	2	Text is centered.

For push button, *align%* can be:

Constant	Value	Description
ALIGN_LEFT		If the control contains a bitmap, the bitmap is aligned left, text right. If there is no bitmap, text is aligned left.
ALIGN_RIGHT		If the control contains a bitmap, the bitmap is aligned right, text left. If there is no bitmap, text is aligned right.
ALIGN_CENTER		Bitmap and text are centered.
ALIGN_TOP		If the control contains a bitmap, the bitmap is aligned top, text bottom. If there is no bitmap, text is aligned top.
ALIGN_BOTTOM		If the control contains a bitmap, the bitmap is aligned bottom, text top. If there is no bitmap, text is aligned bottom.

AutoSize Property

Applies to...

Gets or sets the action taken when the size, text, font name, or font size changes for a label control.

This property allows users to match the size of the label control to the size of the label's text. This property is valid whether or not the label has a border.

Syntax A: *value%* = *control*.AutoSize

Syntax B: *control*.AutoSize = *value%*

Where:	Is:
control	The control ID.
value%	May be one of the following:

Constant	Value	Description

AUTOSIZE_OFF	0	FALSE; Do not autosize.
AUTOSIZE_ON	<>0	TRUE; autosize.

BackColor Property

Applies to...

Gets or sets the background color for a control.

Syntax A: colorValue& = *control*.BackColor

Syntax B: *control*.BackColor = colorValue&

Where:	**Is:**
control	The control ID.
colorValue&	A long integer that consists of four bytes. This value is interpreted in one of two ways:

Physical Colors: If the high bit is 0 (&H00000000), the lower three bytes represent an RGB value. Each color (red, green, and blue) is in the range of 0x00 to 0xFF. The Property Sheet presents a palette of predefined RGB values. The actual color displayed will be the one that is closest to the specified color, based on the specific color resolution and palette of the user's system.

System Colors: If the high bit is not zero (&H800000000), the lower three bytes are an index into the Windows system color table. This table depends on the specific user configuration and will be dynamically updated to reflect any change to the system color palette. Using these values will allow your form to take on the user color preferences as defined in the control panel:

Color Value	Constant	**System Color for:**
&H80000000	SCROLL_BARS	Scroll-bars gray area.
&H80000001	DESKTOP	Desktop.
&H80000002	ACTIVE_TITLE_BAR	Active window caption.
&H80000003	INACTIVE_TITLE_BAR	Inactive window caption.
&H80000004	MENU_BAR	Menu background.
&H80000005	WINDOW_BACKGROUND	Window background.
&H80000006	WINDOW_FRAME	Window frame.
&H80000007	MENU_TEXT	Text in menus.
&H80000008	WINDOW_TEXT	Text in windows.
&H80000009	TITLE_BAR_TEXT	Text in caption, size box, scroll-bar arrow box.

&H8000000A	ACTIVE_BORDER	Active window border.
&H8000000B	INACTIVE_BORDER	Inactive window border.
&H8000000C	APPLICATION_WORKSPACE	Background color of multiple document interface (MDI) applications.
&H8000000D	HIGHLIGHT	Items selected item in a control.
&H8000000E	HIGHLIGHT_TEXT	Text of item selected in a control.
&H8000000F	BUTTON_FACE	Face shading on command buttons.
&H80000010	BUTTON_SHADOW	Edge shading on command buttons.
&H80000011	GRAY_TEXT	Grayed (disabled) text. This color is set to 0 if the current display driver does not support a solid gray color.
&H80000012	BUTTON_TEXT	Text on push buttons.

BorderStyle Property

Applies to...

Gets the style of the border for the control. The meaning of this property is different for each control that it applies to.

This property is read-only at run time.

For form controls, this style determines the Windows border style for the form. If the form appears in an MDI window, this property has no effect because MDI windows define their own frame characteristics.

Syntax *style%* = *control*.BorderStyle

Where:	**Is:**
Control	The control ID.
style%	May be one of the following:

For forms:.

Value of ***style%***	**Description**
0	No border.

1	Single-pixel border.
2	Sizable border.
3	Dialog border.

For combo box, edit control, group box, label, list box, and picture box controls:

Value of ***style%***	**Description**
0	No border (not valid for combo box).
1	Single-pixel border.
2	Indented into the screen.

Cancel Property

Applies to...

Identifies a button as the one to be activated when the user presses the Escape key.

Syntax A: *state%* = *control*.Cancel

Syntax B: *control*.Cancel = *state%*

Where:	**Is:**
control	The control ID.
state%	May be one of the following:

Value of ***state%***	**Description**
0	Control is not activated when the user presses the ESC key.
Non-0	When an ESC is pressed, the control's Click event is fired.

If the Cancel property is set for a button, the button will be activated when the user presses the ESC key, which is an indication that the user wishes to cancel the form. You must implement a script to perform the actual Cancel operation.

Only one push button may have the Cancel property set to non-zero at one time. Setting the Cancel property to non-zero for one control will reset all the other push button controls.

Caption Property

Applies to...

Gets or sets the text that is used for the title, caption, or label portion of a control.

Syntax A: *text$* = *control*.Caption

Syntax B: *control*.Caption = *text$*

For check box, form, group box, label, option button, and push button controls:

Syntax A	**:**	**Is:**
control	The control ID.	
text$	Appears as the caption or label of the control.	

Columns Property

Applies to...

Gets or sets whether a list box displays a single column or multiple columns. This property is read-only at run time.

For multiple-column list boxes, column width is determined by the ColWidth property value.

Syntax A: *cols%* = *listbox*.Columns

Syntax B: *listbox*.Columns = *cols%*

Where:	**Is:**
listbox	The list box ID.
cols%	A value indicating the column setting of a list box.

Value of *cols%*	**Description**
0	Control is a single-column list box.
Non-0	Control is a multiple-column list box.

ColWidth Property

Applies to...

Gets or sets the width, in pixels, of columns in a list box. This is only used for list boxes that have their Columns property set to true.

Syntax A: *width%* = *listbox*.ColWidth

Syntax B: *listbox*.ColWidth = *width%*

Where:	**Is:**
listbox	The list box ID.
width%	The number of pixels specifying the width of each column of a list box.

CurSel Property

Applies to...

Gets or sets the currently selected item in a listbox. A value of -1 is used to indicate that no item is selected.

This property is only valid for single-select list boxes. If the style of the list box is set to multiple-select, use the GetSel and SetSel methods.

Syntax A: *item%* = *listbox*.CurSel

Syntax B: *listbox*.CurSel = *item%*

Where:	**Is:**
Listbox	The list box ID.
item%	The index of the currently selected item.

List box

Cursor Property

Applies to...

Sets the cursor to be displayed as the pointer when the mouse is placed over a control.

This property can be set at design time using the Property Sheet. The desired cursor can be selected with a file browser, allowing the cursor to be previewed before it is selected.

Syntax: set *control*.Cursor = LoadCursor(*filename$*)

set *control1*.Cursor = *control2*.Cursor

set *control1*.Cursor = *control2*.DragCursor

Where:	**Is:**
control, control1, control2	The control ID.
filename$	The filename should be a valid icon (.ICO), bitmap (.bmp) or cursor (.CUR) file. An invalid file type will clear the cursor. If you want to clear the cursor during run time, use **LoadCursor(" ")**.

Check box

Comb

List box

Option

o box	button
Edit control	Picture box
Form	Push button
Group box	Scroll bar
Label control	

Applies to...

Gets or sets whether a button is to be activated when the user presses the ENTER key.

Syntax A: *state%* = *control*.Default

Syntax B: *control*.Default = *state%*

Where:	**Is:**
control	The control ID.
state%	A value setting or indicating whether the ENTER key activates the button.

Value of *state%*	**Description**
0	Control is not activated when the user presses the ENTER key.
Non-0	When the ENTER key is pressed, the control's Click event is fired.

If the **Default** property is set for a button, the button will be activated when the ENTER key is pressed during run time (assuming the control with the focus does not process this key). You must implement a script to perform the actual **Default** operation.

Only one push button may have the **Default** property set to non-zero at one time. Setting the **Default** property to non-zero for one control will reset all the other push button controls on the form.

Push button

DragCursor Property

Applies to...

Determines the cursor to be displayed as the pointer during a drag-and-drop operation.

Syntax: set *control*.DragCursor = LoadCursor(*filename$*)

set *control1*.DragCursor = *control2*.Cursor

set *control1*.DragCursor = *control2*.DragCursor

Where:	**Is:**
control, control1, control2	The control ID.
filename$	The filename should be a valid icon (.ICO), bitmap (.bmp) or cursor (.CUR) file. An invalid file type will clear the cursor. If you want to clear the cursor during run time, use **LoadCursor(" ")**.

This cursor can be set at any time, allowing for complex cursor behavior during a drag/drop sequence. For example, the drag cursor can change based on what control it is over, or based on some function of time.

This property can be set at design time using the Property Sheet. The desired cursor can be selected with a file browser, allowing the cursor to be previewed before it is selected.

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Check box

Combo box

Edit control

Group box

Label control

List box

Option button

Push button

Check box

Combo box

Edit control

Group box

Label control

List box

Option button

Push button

Check box

Combo

Label control

List

box	box
Edit control	Option button
Group box	Push button
Check box	Label control
Combo box	List box
Edit control	Option button
Group box	Push button
Check box	Label control
Combo box	List box
Edit control	Option button
Group box	Push button
Check box	Label control
Combo box	List box
Edit control	Option button

Group box	Push button

Color Value	Constant	**System Color for:**
&H80000000	SCROLL_BARS	Scroll-bars gray area.
&H80000001	DESKTOP	Desktop.
&H80000002	ACTIVE_TITLE_BAR	Active window caption.
&H80000003	INACTIVE_TITLE_BAR	Inactive window caption.
&H80000004	MENU_BAR	Menu background.
&H80000005	WINDOW_BACKGROUND	Window background.
&H80000006	WINDOW_FRAME	Window frame.
&H80000007	MENU_TEXT	Text in menus.
&H80000008	WINDOW_TEXT	Text in windows.
&H80000009	TITLE_BAR_TEXT	Text in caption, size box, scroll-bar arrow box
&H8000000A	ACTIVE_BORDER	Active window border.
&H8000000B	INACTIVE_BORDER	Inactive window border.
&H8000000C	APPLICATION_WORKSPACE	Background color of multiple document interface (MDI) applications.
&H8000000D	HIGHLIGHT	Items selected item in a control.
&H8000000E	HIGHLIGHT_TEXT	Text of item selected in a control.
&H8000000F	BUTTON_FACE	Face shading on command buttons.
&H80000010	BUTTON_SHADOW	Edge shading on command buttons.
&H80000011	GRAY_TEXT	Grayed (disabled) text. This color is set to 0 if the current display driver does not support a solid gray color.
&H80000012	BUTTON_TEXT	Text on push buttons.

Check box	Label contr

		ol
	Combo box	List box
	Edit control	Option button
	Group box	Push button
Check box		List box
	Combo box	Option button
	Edit control	Picture box
	Form	Push button
	Group box	Scroll bar
	Label control	
Check box		List box
	Combo box	Option button
	Edit control	Picture box
	Form	Push button
	Label cont	Scroll bar

rol

HideSelection Property

Gets or sets whether

	Is:
editcontrol	The edit control ID.
state%	May be one of the following:

Value of *state%*	Description
0	FALSE. The text remains highlighted when the edit control loses focus.
Non-0	TRUE. The text does not remain highlighted when the edit control loses focus.

Normally, when text in an edit control is highlighted and the edit control loses focus, the highlight is removed until the focus returns to the edit control. Setting this property to "0" prevents the highlight from being removed.

Hwnd Property

Applies to...

Gets the Windows HWND handle for the control. This property is read-only at run time.

Syntax *hwnd%* = *control*.Hwnd

Where:	Is:
control	The control ID.
hwnd%	The control's HWND handle. Refer to the Windows SDK on what the HWND handle is and how it can be used.

Check box

Combo box

Edit control

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Applies to...

Gets or sets the largest value a scroll bar will represent.

Syntax A: amount% = scrollcontrol.Max

Syntax B: *scrollcontrol*.Max = *amount%*

Where:	Is:
scrollbar	The scroll bar ID.
amount%	The maximum value. The difference in value between the Max and Min values for a scroll bar cannot be greater than 65535.

Scroll bar

MaxButton Property

Applies to...

Gets the state of a form's **MaxButton** property.
[Gets a value indicating whether a form with a caption bar has a standard Windows maximize button.]

Syntax A: *state%* = *form*.MaxButton

Syntax B: *state%* = me.MaxButton

Where:	**Is:**
form	The form ID.
state%	If this property is TRUE, the form has a standard Windows maximize button.

Value of *state%*	**Description**
0	FALSE. The form has no maximize button.
Non-0	TRUE . The form has a maximize button (forms with title bars only).

The HasCaption property must be set to True for this property to have an effect. [The form must have a title bar that holds a caption.]

MaxButton is read-only at run time.

Use the form ID or the operator **"me"**.

Forms that are MDI windows always have a caption and a maximize button.

You can choose to ignore this property and change the setup of the form.

The TRUE constant can be used in a form to indicate a non-zero [True] condition.

The FALSE constant can be used in a form to indicate a zero [False] condition.

Form

MaxLength Property

Applies to...

Gets or sets the maximum number of characters that can be entered into an edit control. A setting of zero places no limit on the number of characters, aside from the limitations set by the operating system.

Syntax A: *length%* = *editcontrol*.MaxLength

Syntax B: *editcontrol*.MaxLength = *length%*

Where:	**Is:**
editcontrol	The edit control ID.

length% Must be from zero to the maximum imposed by the operating system. If the user attempts to enter more characters than this value, a beep will be generated and the excess characters will be ignored.

Edit control

Min Property

Applies to...

Gets or sets the smallest value a scroll bar will represent.

Syntax A: *amount%* = *scrollbar*.Min

Syntax B: *scrollbar*.Min = *amount%*

Where:	**Is:**
scrollbar	The scroll bar ID.
amount%	The minimum value. The difference in value between the Max and **Min** values for a scroll bar cannot be greater than 65535.

Scroll bar

MinButton Property

Applies to...

Gets the state of a form's **MinButton** property.

[Gets a value indicating whether a form with a caption bar has a standard Windows minimize button.]

Syntax A: *state%* = *form*.MinButton

Syntax B: *state%* = me.MinButton

Where:	**Is:**
form	The form ID.
state%	If this property is TRUE, the form has a standard Windows minimize button.

Value of *state%*	**Description**
0	FALSE. The form has no minimize button.
Non-0	TRUE. The form has a minimize button (forms with title bars only).

The HasCaption property must be set to True for this property to have an effect. [The form must have a title bar that holds a caption.]

MinButton is read-only at run time.

Forms that are MDI windows always have a caption and a minimize button.

You can choose to ignore this property and change the setup of the form.

Form

MultiLine Property

Applies to...

Gets the state of an edit control's multi-line property.

Syntax *state%* = *editcontrol*.MultiLine

Where:	**Is:**
editcontrol	The edit control ID.
state%	The state of the edit control.

Value of ***state%***	**Description**
0	FALSE. The edit control is single-line.
Non-0	TRUE. The edit control is multi-line.

This property is read-only at run time.

Multi-line edit controls can have the Alignment property set to a value other than left-justified.

Single-line edit controls ignore the **Alignment** property.

Multi-line edit controls ignore the PasswordChar property.

Edit control

MultiSelect Property

Applies to...

Gets the type of selection list that the list box presents to the user.

Syntax *state%* = *listbox*.MultiSelect

Where:	**Is:**
listbox	The list box ID.
state%	May be one of the following:

Value of *state%*	**Description**
0	Single selection only.
1	Simple multiple selection. The user can switch the selection state of each item in the list box by clicking on it.
2	Extended multiple selection. The user can click on an item and drag, selecting a set of items at one time. Users can also CTRL+click to select several individual items

Either a list box can have just one item highlighted at a time, or it can have multiple items highlighted simultaneously.

This property is read-only at run time.

This property can be set through the Property Sheet at design time.

List box

Name Property

Applies to...

Gets the name of the control.

Syntax *name$* = *control*.Name

Where:	**Is:**
Control	The control ID.
name$	The control name

The control name may be any valid name recognized by the scripting language. It must, however, be unique within a given form. For a form, in addition to the name set by the user, the keyword **"me"** can also be used in place of the name of the form.

You can create arbitrary names by enclosing the name in brackets. For example, **[This is a test control!]** can be used as the name of the control. When, in a script, you refer to controls that have names of this type, always include the brackets.

This property is read-only at run time. The name of the control can only be set through the Property Sheet during design time.

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

Option button

Password Char Property

Applies to...

Gets or sets the password character used in an edit control.

Syntax A:
chr$ = *editcontrol*.PasswordChar

Syntax B:
editcontrol.PasswordChar = *chr$*

Where:	**Is:**
chr$	When *chr$* contains a string, password mode is turned on, and the first character in this string is used as the placeholder character. To turn off password mode, set the password character to the null string ("").

A password field allows the user to enter text without having it appear on the screen. The text that is entered is the **Text** property of the control, but the **PasswordChar** is the text that is displayed for each character typed.

The first character of the *chr$* string is the password character, typically an asterisk (*), which is displayed in the edit control to hide the actual text for the edit control.

The Text property contains the string that is the actual text for the edit control.

This property is only valid for single-line edit controls.

Edit control

Picture Property

Applies to...

Gets or sets the picture for the control. The picture can be loaded from a file or copied from the **Picture** property of another control.

Syntax A: set *control*.Picture = LoadPicture(*filename$*)

Syntax B: set *control2*.Picture = *control1*.Picture

Where:	**Is:**
control	The control ID.
control1, *control2*	Using **Syntax B**, you can set the **Picture** property of *control2* to the **Picture** property value of *control1*.
filename$	The name of the picture file. If it is not a valid picture file, if it does not exist, or if *filename$* is blank, the picture will be cleared.

Value of ***filename$***	**Description**
.BMP	All applicable controls.
.ICO or .WMF	Group box; picture; push button controls.

For check box, list box, and option button controls, the bitmap for these controls is divided into four even sections horizontally, and each section is used for the different states of the control. From left to right, these states are: ON, OFF, ON pressed, and OFF pressed.

For check boxes and option buttons, the picture property is used instead of the standard square or circle.

For list boxes, the picture property is used to indicate the selection state of each item.
For group box, picture, and push button controls, the picture is displayed on the background of the control.

Form
Group box
List box
Picture box
Push button

Check box
Combo box
Edit control
Group box
Label control
List box
Option button
Picture box
Push button
Scroll bar

Check box
Combo box
Edit control
List box
Option button
Push button
Scroll bar

Check box
Combo box
Edit control
Form
List box
Option button
Picture box
Push button

Group box	Scroll bar
Label control	

Text Property

Applies to . . .

Gets or sets

Edit control

Tiled Property

Applies to...

Gets or sets a value indicating whether the bitmap picture is displayed once in the control or tiled throughout the control.

Syntax A: *state%* = *control*.Tiled

Syntax B: *control*.Tiled = *state%*

Where:	Is:
control	The control ID.
state%	May be one of the following:

Value of *state%*	Description
0	FALSE. The bitmap is displayed according to its PictureJustify and PictureCrop properties.
Non-0	TRUE. The bitmap is tiled so that it fills the control.

If the **Tiled** property is true, the PictureJustify property has no effect (its value is ignored).

If the Picture property for a form, group box, or picture is a bitmap, Tiling fills the control with multiple copies of the picture assigned to the control.

If the Picture property is not a bitmap, the **Tiled** property has no effect.

Form

Group box

Picture box

Timer Property

Applies to...

This property sets the number of milliseconds before the Timer event is triggered.

Syntax A: *form*.Timer = *milliseconds%*

Syntax B: me.Timer = *milliseconds%*

Where:	Is:

form	The form ID.
milliseconds%	The number of milliseconds to wait before triggering the Timer event.

This property is write-only at run time.

Use the form ID or the operator "**me**".

Form

Top Property

Applies to...

Gets or sets the top position of the control, in pixels.

Syntax A: *top%* = *control*.Top

Syntax B: *control*.Top = *top%*

Where:	**Is:**
control	The control ID.
top%	**For a form**, **Top** is the number of pixels between the top edge of the form window and the top of the parent window of the form.

For controls other than forms, Top is the number of pixels between the control and the top edge of the form or parent control.

Check box

Combo box

Edit control

Form

Group box

Label control

List box

Option button

Picture box

Push button

Scroll bar

BINARY	Can be either ON or OFF.

DELETETABS.	Can be either ON or OFF
HOSTNAME	The host name.
NOEXTENSIONS	Can be either ON or OFF.
PACKETDEPTH	The packet depth (<= 16).
PCNAME	The PC name
PRIMARYEXTENT	The primary extension (<= 65535).
PRINTFILE	Can be either ON or OFF.
RECORDLENGTH	The record length (<= 4096).
SECONDARYEXTENT	The secondary extension (<= 65535).
SKIPPERF	Can be either ON or OFF.
SHOWSTATUSDIALOG	Can be either ON or OFF.
STATUSPAUSE	Can be either ON or OFF.
STRIPHIBIT	Can be either ON or OFF.
TABINTERVAL	The tab interval (<= 80)
WAIT	Can be either ON or OFF.

If function is successful, it returns *item$*. If it is not successful, it returns a null string.

FtTrigger$ Function [VCBasic Extension]

See Also Example

Requests special actions for file transfer.

Syntax FtTrigger$ (*command$*, *secondary_command$*)

Where:	**Is:**
command$	A string specifying the file transfer command to trigger.
secondary_command$	When required, a string providing the *command$* argument. Commands that do not require this argument will ignore any value passed.

If *command$* Value is:	***secondary_command$* Value must be:**
SEND	<filename> Invalid for ftp transfers. For IXF, <filename(s)> will be appended to PCNAME list.

RECEIVE	<filename> Invalid for ftp transfers. For IXF, <filename> will not overwrite PCNAME setting.
ABORT	None required. Not valid for ftp transfers.
ASCII	None required. (ftp only)
BINARY	None required. (ftp only)
BYE	None required. (ftp only)
CONNECT	None required. (ftp only)
CD	<path> (ftp only)
CLOSE	None required. (ftp only)
DIR	<path> (ftp only)
ERASE	<filename> (ftp only)
GET	<host_filename local_filename> (ftp only)
INPUT	<input string> (ftp only)
LS	<path> (ftp only)
OPEN	<IP address> (ftp only)
PUT	<local_filename host_filename> (ftp only)

For IXF, **FtTrigger** returns "OK" if successful or null string on failure.

For FTP, returns the next status that would apply for **FtQuery$.**

This function is normally used for controlling file transfers under program control.

FtTypeSet$ Function

See Also **Example**

Queries the current file transfer protocol, or changes to a new file transfer protocol.

Syntax FtTypeSet$ (*protocol$*)

Where:	**Is:**
null	Requests the current file transfer protocol in effect.
protocol$	Specifies the file transfer protocol to change to.

Valid *protocol$* values are "FTP" and "IXF".

If *protocol$* is null, **FtTypeSet** returns the current file transfer protocol in effect. Values are the same as for *protocol$* above.

IoInput$ Function [VCBasic Extension]

See Also **Example**

Suspends the emulation module, allowing the macro program to receive data from the I/O module.

Syntax IoInput$ (*timeout%*, *char_count%*, *input_flags%* [, *terminate$*])

Where: **Is:**

timeout% The idle timeout period in seconds. If zero, then there is no timeout.

char_count% Specifies the number of characters to receive before ending the **IoInput$** function. The maximum value for *char_count%* is 32767.

input_flags% Flags that control the behavior while the function is active. The flags may be added together for multiple functionality.

Value of *input_flags%*	**Description**
4	Include termination character in return data string.
8	Translate CR or LF to CRLF combination in return data string.
32	Echo the received I/O data to the local CRT screen and the emulator. **See Note Below**

terminate$ Special termination string to match against. If this optional argument is omitted, the default termination characters are either CR or LF.

IoInput$ returns the data received from the I/O module.

When the **IoInput$** function is executed, terminal emulation is suspended. All I/O is then routed through the executing macro program. The **IoInput$** function, along with **Emit** and **EmitBrk** , allow you to create custom remote applications such as a mini-BBS or a front-end user verification system.

If the amount of data received is greater than *char_count%*, only the first *char_count%* characters are returned and the rest are discarded.

The **IoInput$** function can be terminated by issuing another **IoInput$** function with a *timeout%* of zero and a *char_count%* of zero.

If flag value 32 is set (Echo) then OutsideView will buffer SendKey-ed and/or Emit-ted characters until the host indicates it is able to accept additional character input. This is the same buffering that occurs during cut-and-paste processing.

If flag value 32 is not set (Do not Echo) then the macro programmer must ensure that the host is in the proper state to accept additional character input before data is sent. There is effectively no buffering and WaitCrtUnlock(timeout%) will always return immediate success.

> **When transitioning between echoing and not-echoing** actions, the macro programmer should assume that any pending or following data will be immediately sent to the host. Furthermore, WaitCrtUnlock(timeout%) will return immediate success if it were called during such a transition.

IoQuery$ Function

See Also Example

Queries the current I/O settings.

Syntax IoQuery$ (*item$*)

Where:	**Is:**
item$	A string specifying the type of query to perform.

Value of *item$*	**Description**
* (asterisk)	Returns list of all current configuration settings for the I/O module.
? (question mark)	Returns list of all valid keywords acceptable as *item$*.
\ (backslash)	Returns list of commands accepted by the **IoTrigger$** function.

Async Value	**Description**
CARRIER	Can be TRUE or FALSE
COMPORT	Can be COM1, COM2, COM3 or COM4.
BAUD	Can be 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000 or 256000.
FLOW	Can be NONE, XON/XOFF, RTS/CTS or DSR/TSR.
COMTARGET	Can be HOST or MODEM.
INITSTRING	The string value for modem initialization.
CHARSIZE	Can be 5, 6, 7 or 8.
STOPBITS	Can be 1, 1.5 or 2.
PARITY	Can be E (even), M (mark), N (no), or O (odd).
BREAKPERIOD	Can be from zero to 3000. Value indicates milliseconds.
DUPLEX	Can be NONE, HALF or HALF-LF.
SHOWERRORS	Can be TRUE (writes I/O errors to the session log) or FALSE.
TIMETOLIVE	Can be from zero to 3000. Value indicates milliseconds.

TAPI Value	Description
LINENAME	Descriptive name as defined in Control Panel Modems applet (such as "USRobotics Sportster 14400")
LINEID	Numeric ID of the item defined by LINENAME (zero for the first entry, 1 for the second, etc.).
PHONENUMBER	Number to dial, including any "outside line" and area code digits.

TCP/IP ***or*** SPX Value	Description
CARRIER	Can be TRUE or FALSE
TELTARGET	<host name or IP address> [<port number>]
ENABLENVT	Can be TRUE or FALSE.
ECHO	Can be TRUE or FALSE.
BINARY	Can be TRUE or FALSE.
TERMINAL	Can be TRUE or FALSE.
LINEMODE	Can be TRUE or FALSE.

IoQuery returns a string containing the requested information.

When multiple items (lines) of information are returned, each item is separated by a CR-LF combination.

IoSet$ Function

See Also Example

Prepares an I/O setting to be changed. The change goes into effect when an **IoTrigger$**("CONNECT") function is executed.

Syntax IoSet$ (*item$*, *value$*)

Where:	Is:
item$	A string specifying the I/O setting to change.
value$	The value to use for the specified setting. Allowed values depend on *item$*, as described below.

Async Value	Description
COMPORT	Can be COM1, COM2, COM3 or COM4.
BAUD	Can be 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000 or 256000.

FLOW	Can be NONE, XON/XOFF, RTS/CTS or DSR/TSR.
COMTARGET	Can be HOST or MODEM.
INITSTRING	Any string value for modem initialization.
CHARSIZE	Can be 5, 6, 7 or 8.
STOPBITS	Can be 1, 1.5 or 2.
PARITY	Can be E (even), M (mark), N (no), or O (odd).
BREAKPERIOD	Can be from zero to 3000. Value indicates milliseconds.
DUPLEX	Can be NONE, HALF or HALF-LF.
SHOWERRORS	Can be TRUE (writes I/O errors to the session log) or FALSE.

TAPI Value	**Description**
LINENAME	Descriptive name as defined in Control Panel Modems applet (such as "USRobotics Sportster 14400")
LINEID	Numeric ID of the item defined by LINENAME (zero for the first entry, one for the second, etc.).
PHONENUMBER	Number to dial, including any "outside line" and area code digits.

TCP/IP *or* SPX Value	**Description**
TELTARGET	<host name or IP address> [<port number>]
ENABLENVT	Can be TRUE or FALSE.
ECHO	Can be TRUE or FALSE.
BINARY	Can be TRUE or FALSE.
TERMINAL	Can be TRUE or FALSE.
LINEMODE	Can be TRUE or FALSE.

If *value$* is within range, it is returned. Otherwise, the current setting is returned.

New settings specified by this function do not take effect until a **IoTrigger$**("CONNECT") function is executed.

IoTrigger$ Function [VCBasic Extension]

See Also Example

Invokes the change to the I/O settings previously requested with an **IoSet$** function, or requests a special action from the I/O module.

Syntax IoTrigger$ (*command$*)

Where:	Is:
command$	A string specifying the I/O action to invoke.

Valid strings are BREAK, CONNECT and HANGUP.

If **IoTrigger** is successful, it returns the string "OK". If it is not successful, it returns a null string.

Typical uses for **IoTrigger$** are *command$* values of CONNECT to establish a connection (using any settings specified by the **IoSet$** function) and HANGUP to terminate a connection.

IoTypeSet$ Function

See Also Example

Queries the type of the current I/O connection or changes to a new I/O connection type.

Syntax IoTypeSet$ (*item$*)

Where:	Is:
item$	If null, queries the current connection type.

When set to a connection string, the new connection type goes into effect when an **IoTrigger$**("CONNECT") function is executed. Valid strings are "Asynchronous", "TCP/IP: Windows Sockets" or "NonStop IPX/SPX".

IoTypeSet will return *return_value$* containing the name of the current or set IO. Valid strings are the same as for *item$* above, and are case-sensitive.

When **IoTypeSet$** is called with the proper *item$*, the I/O connection type is changed immediately, but does not connect with a host until an **IoTrigger$**("CONNECT") function is executed.

RunMacro Statement

Runs another macro program from within the current macro program.

Syntax **RunMacro** $*macroname* [, associationFlag%]

Where:	Is:
$macroname	Filename of a macro program in the OutsideView MACRO subdirectory.
AssociationFlag	TRUE (non-zero) or FALSE (zero)

The called (child) macro begins execution when loaded. The calling (parent) macro, however, continues execution independently after invoking the child macro. The parent macro DOES NOT wait for the child macro to complete before continuing its execution.

Since OutsideView can have multiple sessions open simultaneously, you may specify the association of a macro with a particular session.
The value of *associationFlag* allows you to specify this association.

TRUE (non-zero)	This is the default. Associates the new macro with the session containing the parent macro. If the parent macro does not have any session associated with it, the macro will be associated with the currently active session.
FALSE (zero)	Associates the new macro with the currently active session.

The **RunMacro** statement allows you to develop specific, modular macros and then include them by reference within another macro. For example, you might develop a macro that logs onto a specific host or service, and then call that macro from different macros that use different connection types to reach the host or service.

Caution: Take care not to have a macro call itself, as this can result in an endless loop.

Shutdown Statement [VCBasic Extension]

The **Shutdown** statement shuts down (terminates) OutsideView.

Syntax Shutdown

The **Shutdown** statement is used primarily when a macro is started by another application. Another Windows application can start OutsideView and run a macro. The **Shutdown** statement lets the macro exit and returns control to the initial application.

Caution: You should not normally include a **Shutdown** statement in a macro you run directly from OutsideView, as OutsideView itself would then be closed when the macro is through running.
Should you wish to manipulate OutsideView via macros, AppActivate will not switch focus to OutsideView; you must use AppClassActivate.

WaitCrtCursor Function

See Also Example

Waits a specified amount of time for the cursor to be positioned at a specific CRT screen position

Syntax **WaitCrtCursor** (*row%*, *column%*, *timeout%*)

Where:	**Is:**

row%	The CRT screen row of the position to wait for the cursor.
column%	The CRT screen column of the position to wait for the cursor.
timeout%	The number of seconds to wait for the cursor to arrive at the specified position.

WaitCrtCursor will return zero if the timeout period expires prior to the cursor's arrival, and non-zero if the cursor arrives at the specified position before the timer expires.

To convert a cell position into a row and column value, use the **CrtCol** and **CrtRow** functions.

WaitCrtCursor waits for the cursor to reach a particular position on the screen, but makes no demands on the state of the session (i.e., is the keyboard locked?). Some connection methods, such as Async, can take longer than others to unlock the keyboard. The solution is to perform a WaitCrtUnlock following the **WaitCrtCursor**.

WaitCrtUnlock Function

See Also Example

The **WaitCrtUnlock** function waits a specified amount of time for the keyboard to unlock.

Syntax WaitCrtUnlock (*timeout%*)

Where:	Is:
timeout%	The number of seconds to wait for the keyboard to unlock.

WaitCrtUnlock will return zero if the timeout expires before the keyboard unlocks. Non-zero indicates the keyboard is unlocked, and the value indicates the number of seconds remaining in the set timeout period.

Some non-conversational (i.e., Tandem block mode or 3270) modes automatically lock the keyboard after each transmit action. The **WaitCrtUnlock** function allows you to easily synchronize with the host.

WaitDCD Function [VCBasic Extension]

See Also Example

Waits for the Data Carrier Detect signal to go high in a specified time period.

Syntax WaitDCD (*timeout%*)

Where:	Is:

timeout%	The number of seconds to wait for the DCD signal.

WaitDCD returns zero if the timeout expires before carrier is detected. Non-zero indicates a carrier signal, with the value being the number of seconds remaining in the set timeout period.

The **WaitDCD** function polls the I/O module to check if the logical DCD status has changed to a TRUE condition. For an async connection, **WaitDCD** is typically used to sense if there is an active modem connection to a host. For network connection methods, **WaitDCD** can be used to verify that an outstanding call request has completed.

While **WaitDCD** waits a specific time period, the status of carrier can be queried at any time by executing an **IoQuery$**("CARRIER") function.

WaitKeystrokes Function

See Also Example

Waits a specified period of time for the specified number of keystrokes to be entered by the user.

Syntax **WaitKeystrokes** (*timeout%*, *count%*)

Where:	**Is:**
timeout%	The number of seconds to wait for the keystrokes. If *timeout%* is zero, the function waits indefinitely for the specified number of keystrokes.
count%	The number of keystrokes to wait for.

WaitKeystrokes returns zero if the timeout expires before *count%* keystrokes are received. Non-zero indicates the specified number of keystrokes have been entered, with the value being the number of seconds remaining in the set timeout period.

The **WaitKeystrokes** function is typically used for logon information.

If the keystrokes are entered on the CRT emulation screen, use the **CrtGet$**, **CrtCol**, **CrtRow**, and **CrtPosition** functions.

WaitSilent Statement

See Also Example

The **WaitSilent** statement is used to wait for inactivity (idle) over the I/O connection.

Syntax WaitSilent (*timeout%*)

Where:	Is:
timeout%	The number of seconds to wait for no activity on the I/O connection.

The **WaitSilent** statement is typically used to wait for the host to stop sending data after a user action that triggers a response of unknown length.

WaitStr Function [VCBasic Extension]

See Also Example

Waits a specified period of time for one or more specified strings in the data stream passed to the emulation module.

Syntax **WaitStr** (*timeout%, text1$* [, *text2$, text3$,* ... , *text16$*])

Where:	Is:
timeout%	The number of seconds to wait for any of the specified strings to appear in the emulation data stream. If *timeout%* is zero, the function waits indefinitely.
text n*$*	A string to search for in the emulation data stream.

WaitStr returns zero if the timeout expires before any of the specified strings are received. Non-zero indicates the specific *text* n*$* that has been matched in the emulation data stream. The value is 1 for *text1$*, 2 for *text2$*, up to 16 for *text16$*.

The data stream passed to the emulation module is searched for strings matching those specified in the function call. Up to 16 search strings can be specified. For multiple search strings, **WaitStr** terminates on the first match. The function terminates on a match or when the timeout period expires, whichever comes first.

Note: The **WaitStr** function evaluates the raw data stream that is passed to the emulation module. Some hosts or host applications may vary the order of screen writes, or send cursor positioning sequences instead of blanks between data items being displayed. In such instances, it may be easier to use the **WaitSilent** and **CrtSearch** functions.

WaitTime Function

See Also Example

The **WaitTime** function waits for a specific period of time to pass.

Syntax WaitTime (*timeout*)

Where:	**Is:**
timeout	The time of the wait period in 1/100s of a second.

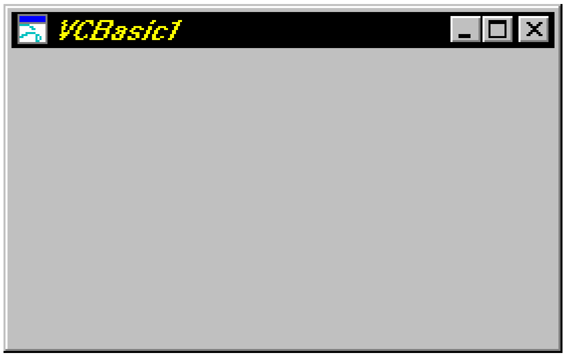
The return value of **WaitTime** is always zero. This call should be treated as a statement; the function call is retained for backward compatibility.

Terminology

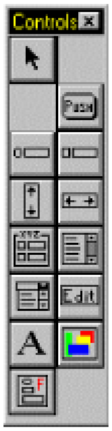
Forms and Controls

Forms and controls are the basic building blocks of VCBasic

A **form** is the canvas that you use to construct a dialog box. Here is the blank form that comes up when you first start the VCBasic Editor:



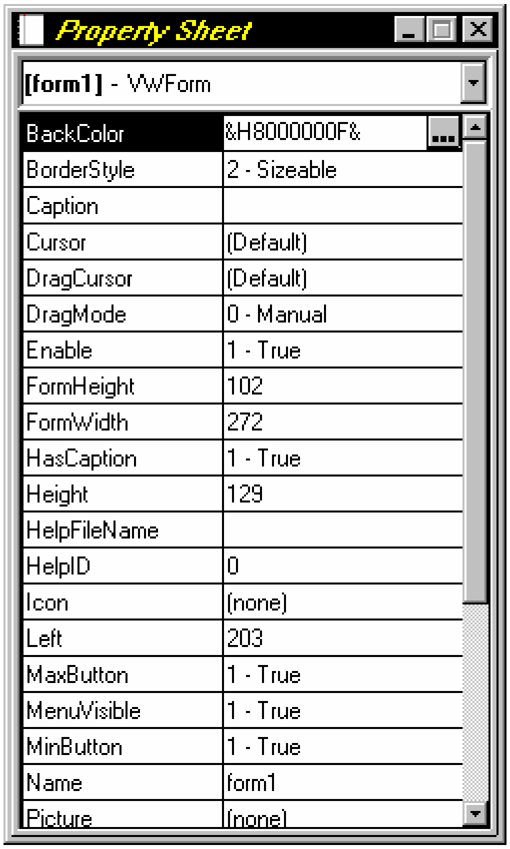
Controls are the objects added to a form that allow interaction with the form. Examples of controls are push buttons, edit fields, check boxes, and scroll bars. The Control Palette (also called the Toolbox) contains the controls you can add to a form. Click on the default Control Palette below to see the meaning of each control icon.



Properties, Events and Methods

Each object is defined by its properties, the events it acts upon, and the methods it uses.

Properties are the attributes of a particular control. All properties of a control are listed, and can be edited, in the control's Property Sheet. A portion of an example Property Sheet, in this case for our blank form, is shown below:



[form1] - VWForm

BackColor	&H8000000F&
BorderStyle	2 - Sizeable
Caption	
Cursor	(Default)
DragCursor	(Default)
DragMode	0 - Manual
Enable	1 - True
FormHeight	102
FormWidth	272
HasCaption	1 - True
Height	129
HelpFileName	
HelpID	0
Icon	(none)
Left	203
MaxButton	1 - True
MenuVisible	1 - True
MinButton	1 - True
Name	form1
Picture	(none)

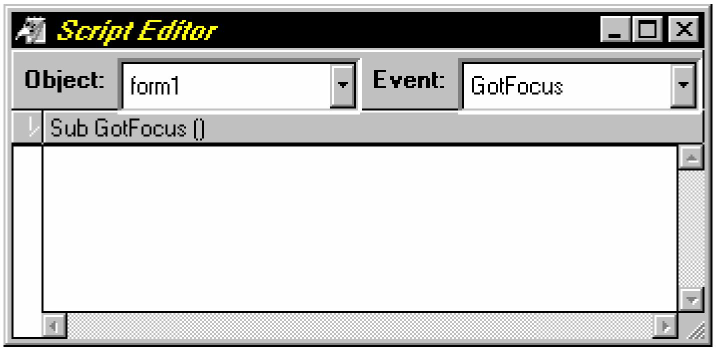
These properties are for our blank form. If you would like to view the Property Sheet, select View:Properties from the menu. **Note:** Not all controls have the same properties. Properties in the Property Sheet will change based on the selected control.

Events are just that: things that happen, and that the control may or may not act upon. Examples of events are a control gaining focus, clicking the mouse, pressing a key or editing text. The events that apply to a selected object are listed in the Script Editor (described later).

Methods are actions the control can use to perform tasks. Examples of methods are determining the selected item in a list, changing the cursor over a control, or refreshing a control's contents. Some methods are actions themselves, while others change certain properties of a control, affecting the look or operation of the control. The methods that apply to a control are listed in the main online help; a method is applied by writing "script" code in the Script Editor.

Scripts

The Script Editor is used to write **scripts**, which are self-contained subroutines of VCBasic code. Scripts are tied to an object and an event. For example, you would write a single script for what should happen when the OK button on a dialog is clicked. The Script Editor for our blank form looks like this:



form1

GotFocus

```
Sub GotFocus ()
```

Back to the Tutorial.

Your First Macro

In this tutorial, you will learn:

- How to add controls to a form to create a user interface.
- How to define the properties of controls.
- How to use the script editor to create the macro code.

What the Macro Will Do

For the purposes of this tutorial, we will create a macro that:

- Prompts the user for a Group Name, User Name and Password.
- Logs on to a TACL session.
- Starts the Tandem ViewSys application.
- Reads the display to determine the length of the CPU 00 CPU Busy bargraph, then display it as a percentage.
- Terminates ViewSys, logs off the session and ends the macro.

Writing the Macro

Step 1. Creating the user interface.

Step 2. Setting the controls' properties.

Step 3. Writing the scripts.

Step 4. Running the macro.

Step 1. Creating the user interface.

1. In OutsideView, go to the Macro menu and select the Macro Editor option. This starts the Visual CommBasic development environment. The macro editor displays, with a blank form (titled "VCBasic1") and the Control Palette visible.

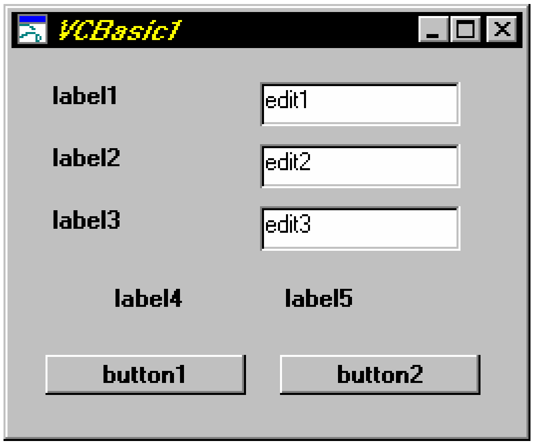
2. Click the A Label control in the Control Palette, then click near the upper left of the blank form to drop a label control. Repeat this until you have five label controls, identified as "label1" through "label5".

3. Click the Edit Edit control in the Control Palette, then click to the right of "label1" to drop an edit control. Repeat this until you have three edit controls, identified as "edit1", "edit2" and "edit3", to the right of the first three label controls.

4. Click the PUSH Push Button control in the Control Palette, then click below the other controls to drop a pushbutton control. Repeat this to add a second button to the right of the first. The buttons are identified as "button1" and "button2".

5. Position the controls to the left and top of the form, then resize the form to correspond to the controls. If you wish, you can use the alignment buttons in the toolbar to align selected controls.

For example, hold down the Control key and click on the the first three labels, then click the ⇤ Toolbar button to align the labels to the left; you can then drag the aligned labels to the desired location on the form. Your form should now look similar to this:



6. Save your work so far by selecing File:Save from the menu. For the filename, type "CPU Busy". Click Save. You'll see the filename replace "VCBasic1" in the form's title bar.

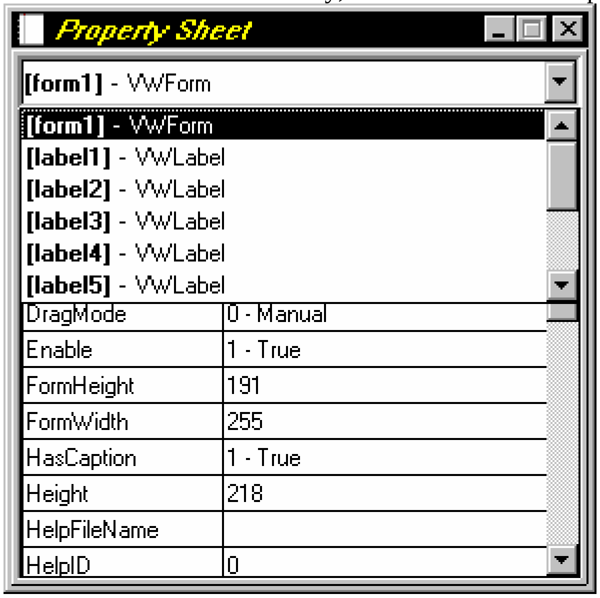
Step 2. Setting the control's properties.

Step 2. Setting the controls' properties.

This step sets the properties of the controls added in the previous step.

One of the properties we will be setting is the control's Name. While the automatically-assigned names of controls can be used, it is good programming practice to use more descriptive names, which makes the script code easier to write and understand.

1. Select View:Properties from the menu. The Property Sheet displays, which is used to set the properties of the controls. The drop-down list at the top of the Property Sheet allows you to select each control individually, as shown in this example:



[form1] - VWForm

[form1] - VWForm
[label1] - VWLabel
[label2] - VWLabel
[label3] - VWLabel
[label4] - VWLabel
[label5] - VWLabel

DragMode	0 - Manual
Enable	1 - True
FormHeight	191
FormWidth	255
HasCaption	1 - True
Height	218
HelpFileName	
HelpID	0

The name of the control is shown at left in bold, while the control type is shown at the right.

2. Select **form1**. Click on the Caption property and type "CPU Busy" (without the quotes) into the edit field and press Enter. You'll see the title of the form change as you enter the caption.

3. Select **label1**. Notice that the red rectangles indicate the active selection. In the Name property, select the "label1" text, change it to "lblGrpName" and press Enter. In the Caption property, select the existing "label1" text, change it to "Group Name" and press Enter. You'll see the text on the form change to reflect the new caption.

4. Select **label2**. Change the Name to "lblUsrName". Change the Caption to "User Name".

5. Select **label3**. Change the Name to "lblPassword". Change the Caption to "Password".

6. Select **label4**. Change the Name to "lblBusyCaption". Change the Caption to "CPU Busy". Change the Visible property to "0 - False" by clicking the drop-down arrow and selecting the False entry.

7. Select **label5**. Change the Name to "lblCPUBusy". Select the Caption and press the Delete key to blank out the label so it has no caption. Change the Visible property to "0 - False". (This is the field where the percentage of "CPU Busy" is displayed. Don't worry that nothing is visible now, the macro script will generate the necessary information when it runs.)

8. Select **edit1**. Change the Name to "txtGrpName". Select the Text property and delete the "edit1" text so that the edit field is blank by default.

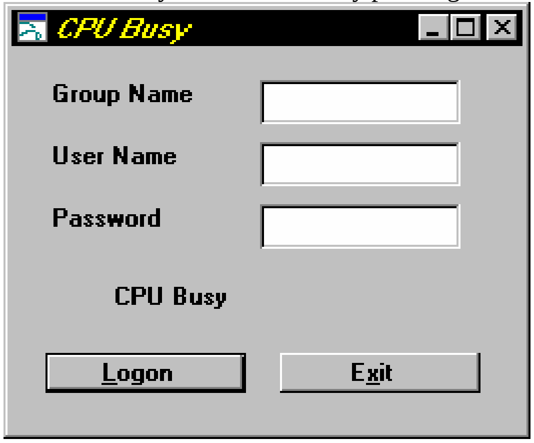
9. Select **edit2**. Change the Name to "txtUsrName". Select the Text property and delete the "edit2" text so that the edit field is blank by default.

10. Select **edit3**. Change the Name to "txtPassword". Select the Text property and delete the "edit3" text so that the edit field is blank by default. In the PasswordChar property, type an asterisk (*) to use as the password character. This causes any text typed into Password field to display as asterisks, rather than the actual text, providing password security.

11. Select **button1**. Change the Name to "cmdLogon". Change the Caption to "&Logon" (the ampersand makes the following character, in this case the "L", the keyboard-accessible accelerator). Change the Default property to "1 - True", making this the default button on the form.

12. Select **button2**. Change the Name to "cmdExit". Change the Caption to "E&xit".

13. Save your work so far by pressing Ctrl+S. Your form should now look something like this:

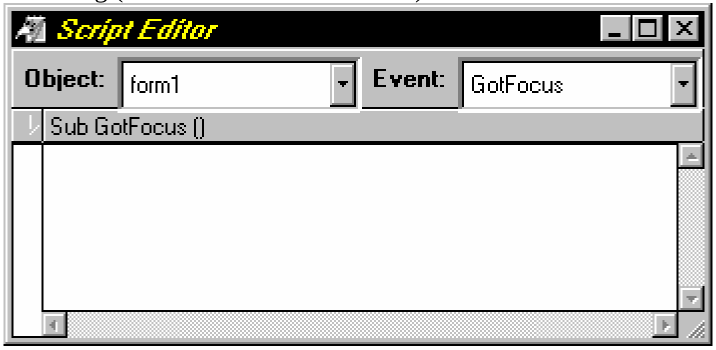


Step 3. Writing the scripts.

Step 3. Writing the scripts.

This step adds the scripts that control the macro's operation.

1. Select View:Script Editor from the menu. The Script Editor displays, similar to the following (which has been sized down):



Object: form1

Event: GotFocus

```
Sub GotFocus ()
```

The two drop-down lists at the top of the Script Editor allow you to select any Object (such as controls on your form) or Event (such as clicking a button). The edit area beneath is where you enter the macro script for the selected Object and Event.

2. In the Event list, select Common. This is a special event that is "common" to all scripts in your macro. In the edit area, type

```
Option Explicit
```

This command forces explicit declaration of all variables used in the macro. Although it is not necessary, explicit variable declarations help avoid some very troublesome errors in larger macros.

3. In the Object list, select "cmdLogon", which is the name of the Logon button. The Event list automatically changes to "Click". Now we will add the script that executes when the user clicks the Logon button. Type the following code, or copy it from this Help window and paste it into the Script Editor's edit area:

```
dim retInt as integer
' Logon to TACL
emit "Logon " & txtGrpName.text & "." & txtUsrName.text
' Wait for the "Password:" prompt from TACL
retInt = WaitStr(5, "Password:")
emit txtPassword.text
' Wait for TACL prompt
retInt= WaitStr(5, ">")
' Start ViewSys
emit "viewsys"
' Show the CPU Busy controls
lblBusyCaption.visible = TRUE
lblCPUBusy.visible = TRUE
' Disable the logon controls
txtGrpName.enable = FALSE
```

```
txtUsrName.enable = FALSE
txtPassword.enable = FALSE
cmdLogon.enable = FALSE
' Initialize timer
me.timer = 1000
```

This script uses the values entered by the user for Group Name, User Name and Password to logon to a TACL session. Note that the WaitStr commands are used to wait for the proper responses from the host. After the TACL prompt is detected, the script starts a ViewSys process. The edit controls and Logon button are also disabled to prevent the user changing any information and sending invalid data to the now-logged-in session.

4. In the Object list, select "form1". In the Event list, select "Timer". The Timer event is used for macro processes that you want to execute multiple times. In this case, we'll use the Timer event to periodically check the "cpu busy" item for the CPU 00 processor from ViewSys. Type the following code, or copy it from this Help window and paste it into the Script Editor's edit area:

```
dim intCol as integer

const CrtAttrReverse = 2
const BarLength = 28  ' Maximum length of CPU Busy bar
' Find the length of the CPU busy bar by checking
' for the reverse character attribute
inCol = 3
do while CrtAttr(6, intCol) AND CrtAttrReverse
    intCol = intCol + 1
loop
' Format the value and display
lblCPUBusy.caption = Format$(cSng(intCol - 3) / BarLength, "percent")
' Reinitialize the timer for one-second intervals
me.timer = 1000
```

This script determines the length of the CPU 00 Busy bargraph by checking the character attribute in the bargraph row (row 6, starting in column 3). The character attribute is retrieved with the CrtAttr command, and is compared with the reverse attribute value to determine if the character space is a reverse video bar character. The percentage of "found" reverse video characters is calculated, formatted, and displayed in the "lblCPUBusy" control. To repeat this procedure every second, the timer property for the form ("me" always refers to the form) is reinitialized at the end of the script.

5. In the Object list, select "cmdExit", which is the name of the Exit button. The Event list automatically changes to "Click". Now we will add the script that executes when the user clicks the Exit button. Type the following code, or copy it from this Help window and paste it into the Script Editor's edit area:

```
dim retStr as string, retInt as integer

' Disable timer
me.timer = 0
' Stop ViewSys by sending F16
retStr = CrtTrigger("FUNCKEY", "Tandem F16")
' Wait for TACL prompt
retInt = WaitStr(5, ">")
' Logoff from TACL
```

```
emit "logoff"
' Terminate the macro
UnloadForm me
```

This script is used to close down the ViewSys process, log off the TACL session, and close down the macro. Note the use of the CrtTrigger command to send the F16 function key (see the online help of this command for details on specifying the correct function key). Also, "UnloadForm me" is how you terminate a macro. Since Visual CommBasic is an event-driven language, you must explicitly terminate execution

6. Save your work so far by pressing Ctrl+S.

Step 4. Running the macro.

Step 4. Running the macro.

Congratulations! The macro is complete and now can be run. But before we start it, keep in mind that Visual CommBasic macros are bound to the session that is active when they are executed. This applies whether you are running a macro from OutsideView (Options: Run Macro) or from the VCB Macro Editor. Our tutorial macro requires that a Tandem session be connected, active, and running TACL but not logged in.

To run your first macro:

1. Click the Run button on the Toolbar. The Macro Editor enters run mode, and executes the macro (If there are any syntax errors in your script code, the Script Editor displays with the detected error marked.).

2. Enter your Group Name, User Name and Password in the edit fields. Click the Logon button and watch the results.

3. If you make any changes to your macro, make sure to save them. To watch the macro operate the session, close the Macro Editor and select Macro:Run Macro from the OutsideView menu.

That's the end of our tutorial.
You can continue on to an Alphabetical List of all commands available in Visual CommBasic or a list of all the commands grouped by their function. You may also go back to the Table of Contents or to the Visual CommBasic Overview.

Abs Function

See AlsoExample

Returns the absolute value of a number.

Syntax Abs(*number*)

Where:	**Is:**
number	Any valid numeric expression.

The data type of the return value matches the type of the *number* except for types 8 and 0.

If *number* is a **Variant** string (vartype 8), the return value will be converted to vartype 5 (Double).

If the absolute value evaluates to vartype 0 (Empty), the return value will be vartype 3 (Long).

AppActivate Statement
See AlsoExample

Activates an application window.

Syntax AppActivate *title*

Where:	**Is:**
title	A string expression for the title-bar name of the application window to activate.

Title must match the name of the window character for character, but comparison is not case-sensitive, e.g., "File Manager" is the same as "file manager" or "FILE MANAGER".
If there is more than one window with a name matching *title*, a window is chosen at random.

AppActivate changes the focus to the specified window but does not change whether the window is minimized or maximized.
Use **AppActivate** with the **SendKeys** statement to send keys to another application.
If you want to open an application that dynamically changes its title bar, such as OutsideView, you must use the AppClassActivate statement instead.

Asc Function

See AlsoExample

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

Syntax Asc(*string$*)

where:	**is:**
string$	A string expression of one or more characters.

To obtain the first byte of a string, use **AscB**.

To change a character code to a character string, use **Chr$**.

Atn Function

See AlsoExample

Returns the angle (in radians) for the arc tangent of the specified number.

Syntax Atn(*number*)

where:	**is:**
number	Any valid numeric expression.

The **Atn** function assumes *number* is the ratio of two sides of a right triangle: the side opposite the angle to find and the side adjacent to the angle.
The return value is a single-precision value for a ratio expressed as an integer, a currency, or a single-precision numeric expression.
The return value is a double-precision value for a long, Variant or double-precision numeric expression.

To convert radians to degrees, multiply by (180/PI). The value of PI is approximately 3.14159.

Beep Statement
See AlsoExample

Produces a single tone through the computer speaker.

Syntax **Beep**

The frequency and duration of the tone depends on the hardware.

Begin Dialog ... End Dialog Statement
See AlsoExampleOverview

Begins and ends a dialog-box declaration.

Syntax **Begin Dialog** *dialogName* [*x* , *y* **,**] *dx* , *dy* [, *caption$*] [, **.***dialogfunction*]
' dialog box definition statements
End Dialog

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

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

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

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

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

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

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

id$ The text string that identifies the dialog control that triggered the call to the dialog function (usually because the user changed this control).

action% An integer from 1 to 5 identifying the reason why the dialog function was called.

suppvalue& Gives more specific information about why the dialog function was called.

As with any Basic function, these arguments can have different names. The arguments of the dialog function can also be Variants. (Click the underlined argument above to see more about it.)

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

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

Button Statement

See AlsoExample

Defines a custom push button.

Syntax A **Button** x , y , dx , dy , text$ [, **.**id]

Syntax B **PushButton** *x , y , dx , dy , text$* [, **.***id*]

where:	**is:**
x , y	The position of the button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
text$	The name of the push button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

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

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

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

ButtonGroup Statement

See AlsoExample

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

Syntax **ButtonGroup** *.field*

where:	**is:**
.field	The field to contain the user's custom button selection.

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

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

Call Statement

See AlsoExample

Transfers control to a subprogram or function.

Syntax A **Call** *subprogram-name* [**(** *argumentlist* **)**]

Syntax B subprogram-name argumentlist

where:	**is:**
subprogram-name	The name of the subroutine or function to call.
argumentlist	The arguments for the subroutine or function (if any).

Use the Call statement to call a subprogram or function written in Basic or to call C procedures in a DLL. These C procedures must be described in a **Declare** statement or be implicit in the application.

If a procedure accepts named arguments, you can use the names to specify the argument and its value. Order is not important. For example, if a procedure is defined as follows:

```
Sub mysub(aa, bb, optional cc, optional dd)
```

the following calls to this procedure are all equivalent:

```
call mysub(1, 2, , 4)
mysub aa := 1, bb := 2, dd :=4
call mysub(aa := 1, dd:=4, bb := 2)
mysub 1, 2, dd:=4
```

Note that the syntax for named arguments is as follows:

```
argname  := argvalue
```

where *argname* is the name for the argument as supplied in the **Sub** or **Function** statement and *argvalue* is the value to assign to the argument when you call it. The advantage to using named arguments is that you do not have to remember the order specified in the procedure's original definition, and if the procedure takes optional arguments, you do not need to include commas (,) for arguments that you leave out.

The procedures that use named arguments include:

1. All functions defined with the **Function** statement.

2. All subprograms defined with the **Sub** statement.

3. All procedures declared with **Declare** statement.

1. Many built-in functions and statements (such as **InputBox**).

    1. Some externally registered DLL functions and methods.

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

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

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

CancelButton Statement
See AlsoExample

Sets the position and size of a Cancel button in a dialog box.

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

where:	**is:**
x , *y*	The position of the Cancel button relative to the upper left corner of the dialog box.
dx , *dy*	The width and height of the button.
.*id*	An optional identifier for the button.

A *dy* value of 14 can usually accommodate text in the system font.

.*Id* is used by the dialog statements that act on this control.

If you use the Dialog **statement** to display the dialog box and the user clicks Cancel, the box is removed from the screen and an Error 102 is triggered.

If you use the Dialog **function** to display the dialog box and the user clicks Cancel, the function will return 0 and no error occurs.

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

Caption Statement
See AlsoExample

Defines the text to be used as the title of a dialog box.

Syntax Caption *text$*

where:	**is:**
text$	A string expression containing the title of the dialog box.

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

If no **Caption** statement is specified for the dialog box, a default caption is used.

ChDir Statement
See AlsoExample

Changes the default directory for the specified drive.

Syntax **ChDir** *path$*

where:	**is:**
path$	A string expression identifying the new default directory.

The syntax for *path$* is: [*drive*:] [\] *directory* [*directory*]

If the drive argument is omitted, **ChDir** changes the default directory on the current drive.

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

ChDrive Statement
See AlsoExample

Changes the default drive.

Syntax **ChDrive** *drive$*

where:	**is:**
drive$	A string expression designating the new default drive.

This drive must exist and must be within the range specified by the LASTDRIVE statement in the CONFIG.SYS file.
If a null argument (" ") is supplied, the default drive remains the same.
If the *drive$* argument is a string, **ChDrive** uses the first letter only.
If the *drive$* argument is omitted, an error message is produced.

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

CheckBox Statement
See AlsoExample

Creates a check box control in a dialog box.

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

where:	**is:**
x , y	The upper left corner coordinates of the check box, relative to the upper left corner of the dialog box.
dx	The combined width of the check box and the *text$* field.
dy	The height of *text$*.
text$	The title shown to the right of the check box.
.field	The name of the dialog-record field that will hold the current check box setting (0=unchecked, -1=grey, 1=checked).

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

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

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

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

VCBasic treats any other value of *.field* which isn't -1, 0, or 1 as if the value was 1. The *.field* argument is also used by the dialog statements that act on this control.

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

Class List

Following is a list of classes that can be used in a **Dim** statement, a **Typeof** expression, or with the **New** operator:

Object	Provides access to OLE2 automation.

Clipboard Example

The Windows Clipboard can be accessed directly in your program to enable you to get text from and put text into other applications that support the Clipboard.

Syntax Clipboard.Clear

Clipboard.GetText()

Clipboard.SetText *string$*

Clipboard.GetFormat()

where:	**is:**
string$	A string or string expression containing the text to send to the Clipboard.

The Clipboard methods supported are as follows:

Me tho d:	**What it does:**
Cle	Clears the contents of the Clipboard.

ar	
GetText	Returns a text string from the Clipboard.
SetText	Puts a text string to the Clipboard.
GetFormat	Returns TRUE (non-0) if the format of the item on the Clipboard is text. Otherwise, returns FALSE (0).

Note: Data on the Clipboard is lost when another set of data of the same format is placed on the Clipboard (either through code or a menu command).

CLng Function

See AlsoExample

Converts an expression to the data type **Long** by rounding.

Syntax CLng(*expression*)

where:	**is:**
expression	Any expression that can evaluate to a number.

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

Strings that cannot be converted to a long result in a "Type Mismatch" error.

Variants containing null result in an "Illegal Use of Null" error.

Close Statement

See AlsoExample

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

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

where:	**is:**
#	You may use this symbol or not. It has no effect.
filenumber%	An integer expression identifying the file to close.

Filenumber% is the number assigned to the file in the **Open** statement. If this argument is omitted, all open files are closed.

Once a **Close** statement is executed, the association of a file with *filenumber%* is ended, and the file can be reopened with the same or a different file number.

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

Cos Function

See AlsoExample

Returns the cosine of an angle.

Syntax Cos(*number*)

where:	**is:**
number	An angle in radians.

The return value will be between -1 and 1.

The return value is a single-precision number if the angle has a data type **Integer**, **Currency**, or is a single-precision value.

The return value will be a double precision value if the angle has a data type **Long**, **Variant** or is a double-precision value.

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

CSng Function

See AlsoExample

Converts an expression to the data type **Single** (single-precision floating point.)

Syntax CSng(*expression*)

where:	**is:**
expression	Any expression that can evaluate to a number.

The *expression* must have a value within the range allowed for the **Single** data type, or an error occurs.

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

CStr Function

See AlsoExample

Converts an expression to the data type **String**.

Syntax CStr(*expression*)

where:	**is:**
expression	Any expression that can evaluate to a number.

The **CStr** statement accepts any type of *expression*:

If *expression* is: CStr returns:

Boolean	A String containing "True" or "False".
Date	A String containing a date.
Empty	A zero-length String ("").
Error	A String containing "Error", followed by the error number.
Null	A run-time error.
Other Numeric	A String containing the number.

CurDir Function

See AlsoExample

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

Syntax CurDir[$] [(*drive$*)]

where:	**is:**
drive$	A string expression containing the drive to search.

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

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

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

Date Function

See AlsoExample

Returns a string representing the current date.

Syntax Date[$]

The **Date** function returns a ten character string.

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

DateSerial Function

See AlsoExample

Returns a date value for year, month, and day specified.

Syntax DateSerial(*year% , month% , day%* **)**

where:	**is:**
year%	A year between 100 and 9999, or a numeric expression.
month%	A month between 1 and 12, or a numeric expression.
day%	A day between 1 and 31, or a numeric expression.

The **DateSerial** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999. A value of zero represents December 30, 1899. Times are represented as fractional days.

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

DateValue Function
See AlsoExample

Returns a date value for the string specified.

Syntax DateValue(*date$*)

where:	**is:**
date$	A string representing a valid date.

The **DateValue** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999. A value of zero represents December 30, 1899. Times are represented as fractional days.

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

Day Function

See AlsoExample

Returns the day of the month (1-31) of a date-time value.

Syntax Day(*date*)

where:	**is:**
date	Any expression that can evaluate to a date.

Day attempts to convert the input value of *date* to a date value. If it cannot convert, a run-time error occurs. The return value is a **Variant** of vartype 2 (integer) unless the value of *date* is null
If the value of *date* is null, a Variant of vartype 1 (null) is returned.

DDEAppReturnCode Function
See Also Example

Returns a code received from an application on an open dynamic data exchange (DDE) channel.

Syntax   DDEAppReturnCode()

To open a DDE channel, use **DDEInitiate**. Use **DDEAppReturnCode** to check for error return codes from the server application after using **DDEExecute**, **DDEPoke** or **DDERequest**.

DDEExecute Statement
See AlsoExample

Sends one or more commands to an application via a dynamic-data exchange (DDE) channel.

Syntax   DDEExecute *channel% , cmd$*

where:	**is:**
channel%	An integer or expression for the channel number of the DDE conversation as returned by **DDEInitiate**.
cmd$	One or more commands recognized by the application.

If *channel* doesn't correspond to an open channel, an error occurs.

You can also use the format described under **SendKeys** to send specific key sequences. If the server application cannot perform the specified command, an error occurs.

In many applications that support DDE, *cmd$* can be one or more statements or functions in the application's macro language. Note that some applications require that each command received through a DDE channel be enclosed in brackets and quotation marks.

You can use a single **DDEExecute** instruction to send more than one command to an application.

> Many commands require arguments in the form of strings enclosed in quotation marks. Because quotation marks indicate the beginning and end of a string in VCBasic, you must use Chr$(34) to include a quotation mark in a command string. For example, the following instruction tells Microsoft Excel to open MYFILE.XLS:

DDEExecute channelno, "[OPEN(" + Chr$(34) + "MYFILE.XLS" + Chr$(34) + ")]"

DDERequest Function
See AlsoExample

Returns data from an application through an open dynamic data exchange (DDE) channel.

Syntax DDERequest[$] (*channel%*, *item$*)

where:	**is:**
channel%	An integer or expression for the open DDE channel number.
item$	A string or expression for the name of an item in the currently opened topic to get information about.

If *channel%* doesn't correspond to an open channel, an error occurs.

If the server application doesn't recognize *item$*, an error occurs.

If **DDERequest** is unsuccessful, it returns an empty string ("").

When you open a channel to an application using **DDEInitiate**, you also specify a topic, such as a filename, to communicate with. The *item$* is the part of the topic whose contents you are requesting.

DDERequest returns data as a text string. Data in any other format cannot be transferred, nor can graphics.

Many applications that support DDE recognize a topic named **System**. Three standard items in the **System** topic are described in the following table:

Item:	**Returns:**
SysItems	A list of all items in the **System** topic
Topics	A list of available topics
Formats	A list of all the Clipboard formats supported

Declare Statement
See AlsoExample

Declares a procedure in a module or dynamic link library (DLL).

Syntax A **Declare Sub** *name* **[** *libSpecification* **] [(** *parameter* **[As** *type* **])]**

Syntax B **Declare Function** *name* **[** *libSpecification* **] [(** *parameter* **[As** *type* **])] [As** *functype* **]**

where:	**is:**
name	The subprogram or function procedure to declare.
libSpecification	The location of the procedure (module or DLL).
parameter	The arguments to pass to the procedure, separated by commas.
type	The type for the arguments.
functype	The type of the return value for a function procedure.

The **Declare** statement has two uses: forward declaration of a procedure whose definition is to be found later in this module, and declaration of a procedure that is to be found in an external Windows DLL or external VCBasic module.

A **Sub** procedure does not return a value. A **Function** procedure returns a value, and can be used in an expression. To specify the data type for the return value of a function, end the Function name with a type character or use the **As** *functype* clause shown above. If no type is provided, the function defaults to data type **Variant**.

If the *libSpecification* is of the format:

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

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

If the *libSpecification* is of the format:

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

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

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

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

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

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

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

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

When an empty string ("") is passed **ByVal** to an external procedure, the external procedure will receive a NULL pointer. If you want to send a valid pointer to an empty string, use **Chr$(0)**.

Def*type* Statement
See AlsoExample

Specifies the default data type for one or more variables.

Syntax **DefCur** varTypeLetters
DefInt varTypeLetters
DefLng varTypeLetters
DefSng varTypeLetters
DefDbl varTypeLetters
DefStr varTypeLetters
DefVar varTypeLetters

where:	is:
varTypeLetters	A first letter of the variable name to use.

VarTypeLetters can be a single letter, a comma-separated list of letters, or a range of letters. For example, a-d indicates the letters a, b, c and d.

The case of the letters is not important, even in a letter range.

The letter range a-z is treated as a special case: it denotes all alpha characters, including the international characters.

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

Variables defined using the Global or Dim can override the **Def*type*** statement by using an **As** clause or a type character.

Dialog Function

See AlsoExample Overview

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

Syntax **Dialog (** *recordName* **)**

where:	is:
recordName	A variable name declared as a dialog box record.

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

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

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

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

Dialog Statement

See AlsoExample Overview

Displays a dialog box.

Syntax Dialog *recordName*

where:	**is:**
recordName	A variable name declared as a dialog box record.

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

If the user exits the dialog box by pushing the Cancel button, the run-time error 102 is triggered, which can be trapped using **On Error**.

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

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

Dim Statement

See AlsoExampleOverview

Declares variables for use in a Basic program.

Syntax **Dim** [**Shared**] *variableName* [**As** [**New**] *type*] [,*variableName* [**As** [**New**] *type*]] **...**

where:	**is:**
variableName	The name of the variable to declare.
type	The data type of the variable.

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

Dim *my_1st_variable* **As String**
Dim [one long and strange! variable name] **As String**

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

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

Arrays

Numbers

Objects

Records

Strings

Variants

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

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

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

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

DlgControlID Function

See AlsoExample Overview

Returns the numeric ID of a dialog box control with the specified *Id$* in the active dialog box.

Syntax DlgControlID (*Id$*)

where:	**is:**
Id$	The string ID for a dialog control.

The **DlgControlID** function translates a string *Id$* into a numeric ID.
This function can only be used from within a dialog box function.

The value of the numeric identifier is based on the position of the dialog box control with the dialog; it will be 0 (zero) for the first control, 1 (one) for the second control, and so on.

Given the following example, the statement DlgControlID("doGo") will return the value 1.

```
Begin Dialog newdlg 200, 200

   PushButton 40, 50, 80, 20, "&Stop", .doStop

   PushButton 40, 80, 80, 20, "&Go", .doGo

End Dialog
```

The advantage of using a dialog box control's numeric ID is that it is more efficient, and numeric values can sometimes be more easily manipulated.

Rearranging the order of a control within a dialog box will change its numeric ID. For example, if a PushButton control originally had a numeric value of 1, and a textbox control is added before it, the PushButton control's new numeric value will be 2.

The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements. The string ID does not include the period (.) and is case-sensitive.

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

DlgEnable Function

See AlsoExample Overview

Returns the enable state for the specified dialog control (-1=enabled, 0=disabled).

Syntax DlgEnable (*Id*)

where:	is:
Id	The control ID for the dialog control.

If a dialog box control is enabled, it is accessible to the user. You might want to disable a control if its use depends on the selection of other controls.

Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier.

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

DlgEnable Statement

See AlsoExample Overview

Enables, disables, or toggles the state of the specified dialog control.

Syntax DlgEnable *Id* [, *mode*]

where:	is:
Id	The control ID for the dialog control to change.
mode	An integer representing the enable state (1=enable, 0=disable)

If *mode* is omitted, the **DlgEnable** toggles the state of the dialog control specified by *Id*. If a dialog box control is enabled, it is accessible to the user. You might want to disable a control if its use depends on the selection of other controls.

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

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

DlgEnd Statement

See AlsoExample Overview

Closes the active dialog box.

Syntax DlgEnd *exitCode*

where:	is:
exitCode	The return value after closing the dialog box (-1=OK, 0=Cancel).

ExitCode contains a return value only if the dialog box was displayed using the **Dialog** function. That is, if you used the **Dialog** statement, *exitCode* is ignored.

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

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

DlgFocus Function

See AlsoExample Overview

Returns the control ID of the dialog control having the input focus.

Syntax DlgFocus[$]( )

A control has focus when it is active and responds to keyboard input.

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

DlgFocus Statement

See AlsoExample Overview

Sets the focus for the specified dialog control.

Syntax DlgFocus *Id*

where:	is:
Id	The control ID for the dialog control to make active.

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

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

DlgListBoxArray Function

See Also Example Overview

Returns the number of elements in a list or combo box.

Syntax DlgListBoxArray (*Id*[, *Array$*] **)**

where:	is:

Id The control ID for the list or combo box.

Array$ The entries in the list box or combo box returned.

If *array$* is omitted, the function returns the number of entries in the specified dialog control.

If the *Array$* argument is used, it fills the array with the entries of the list box or the combo box. *Array$* is a one-dimensional array of dynamic strings. If *array$* is dynamic, its size is changed to match the number of strings in the list or combo box. If *array$* is not dynamic and it is too small, an error occurs.

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

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

DlgListBoxArray Statement

See Also **Example** **Overview**

Fills a list or combo box identified by *Id* with the strings from the array.

Syntax **DlgListBoxArray** *Id*, *Array$*

where: **is:**

Id The control ID for the list or combo box.

Array$ The entries for the list box or combo box.

Array$ has to be a one-dimensional array of dynamic strings. One entry appears in the list box for each element of the array. If the number of strings changes depending on other selections made in the dialog box, you should use a dynamic array and **ReDim** the size of the array whenever it changes.

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

DlgSetPicture Statement

See Also **Example** **Overview**

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

Syntax **DlgSetPicture** *Id*, *filename$* , *type*

where: **is:**

Id The control ID for the picture dialog control.

filename$ The name of the bitmap file (.BMP) to use. If *type* =3, then this argument is ignored.

type An integer representing the location of the file (0=*filename$*, 3=Clipboard)

Use the **DlgControlID** function to find the numeric ID for a dialog control based on its string identifier.

The string identifiers come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

If the picture is not available (the file *filename$* doesn't exist, or it doesn't contain a bitmap, or there is no bitmap on the clipboard), the picture control will display the picture frame and the text "(missing picture)". This behavior may be changed by adding 16 to the value of *type*, changing the value of *type* to 16 or 19. If *type* is 16 or 19 and the picture is not available, then a runtime error will be triggered.

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

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

DlgText Function

See Also **Example** **Overview**

Returns the text associated with a dialog control for the current dialog box.

Syntax **DlgText**[**$**] **(** *Id* **)**

where:	is:
Id	The control ID for a dialog control.

If the control is a text box or a combo box, **DlgText** function returns the text that appears in the text box.
If the control is a list box, the function returns its current selection.
If the control is a command button, option button, option group, or a check box, the function returns the control's label.

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

DlgText Statement

See Also **Example** **Overview**

Changes the text associated with a dialog control for the current dialog box.

Syntax **DlgText** *Id*, *text$*

where:	is:
Id	The control ID for a dialog control.
text$	The text to use for the dialog control.

If the dialog control is a text box or a combo box, **DlgText** sets the text that appears in the text box.

If the dialog control is a list box, a string equal to *text$* or beginning with *text$* is selected.

If the dialog control is a text control, **DlgText** sets it to *text$*.

If the dialog control is a command button, option button, option group, or a check box, the statement sets its label to *text$*.

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

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

DlgValue Function

See Also **Example** **Overview**

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

Syntax **DlgValue (** *Id* **)**

where:	is:
Id	The control ID for a dialog control.

The values returned depend on the type of dialog control:

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

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

DlgValue Statement

See Also **Example** **Overview**

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

Syntax **DlgValue** *Id*, *value%*

where:	is:
Id	The control ID for a dialog control.
value%	The new value for the dialog control.

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

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

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

DlgVisible Function

See Also **Example** **Overview**

Returns -1 if a dialog control is visible, 0 if it is hidden.

Syntax **DlgVisible (** *Id* **)**

where:	**is:**
Id	The control ID for a dialog control.

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

DlgVisible Statement

See Also **Example** **Overview**

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

Syntax **DlgVisible** *Id* [, *mode*]

where:	**is:**
Id	The control ID for a dialog control.
mode	Value to use to set the dialog control state:

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

If you omit the *mode*, the dialog box state is toggled between visible and hidden. [If it was hidden, it will become visible. If it was visible, it will be hidden.]

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

Do...While Loop Statement

See Also **Example**

Repeats a series of program lines as long as a **While** condition is TRUE or until an **Until** condition is TRUE.

Syntax A **Do** [{ **While** | **Until** } *condition*]

[*statementblock*]

[**Exit Do**]

[*statementblock*]

Loop

Syntax B **Do**

[*statementblock*]

[**Exit Do**]

[*statementblock*]

Loop [{ **While** | **Until** } *condition*]

where:	**is:**
Condition	Any expression that evaluates to TRUE (nonzero) or FALSE (0).

statementblock(s) Program lines to repeat while (or until) *condition* is TRUE.

When an **Exit Do** statement is executed, control goes to the statement which follows the Loop statement. When used within a nested loop, an **Exit Do** statement moves control out of the immediately enclosing loop.

DoEvents Statement

See Also **Example**

Yields execution to Windows for processing operating system events.

Syntax DoEvents

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

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

DropComboBox Statement

See Also **Example**

Creates a combination of a drop-down list box and a text box.

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

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

where:	is:
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the selections for the combo box.
stringarray$	An array of dynamic strings for the selections in the combo box.
.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 *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

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

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

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

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

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

DropListBox Statement

See Also **Example**

Creates a drop-down list of choices.

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

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

where: is:

x , y The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.

dx , dy The width and height of the list box.

text$ A string containing the selections for the list box.

stringarray$ An array of dynamic strings for the selections in the list box.

.field The name of the dialog-record field that will hold the text string chosen from the list box.

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

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

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

When the user selects OK (or selects any customized button created using the **Button** statement), a number representing the selection's position in the *text$* string is recorded in the field designated by the *.field* argument. The numbers begin at zero. If no item is selected, it is -1. The *.field* argument is also used by the dialog statements that act on this control.

A drop list box is different from a list box. The drop list box only displays its list when the user selects it; the list box also displays its entire list in the dialog box. The droplistbox may overlap other controls or fall outside the dialog box when it drops down.

All dialog functions and statements that apply to the ListBox apply to the **DropListBox** as well.

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

Environ Function

Example

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

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

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

where: is:

Environment-string$ The name of a keyword in the operating system environment.

Numeric expression% A number for the position of the string in the environment table. (1st, 2nd, 3rd, etc.)

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

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

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

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

Eof Function

See Also **Example**

Returns the value -1 if the end of the specified open file has been reached, otherwise it will return 0.

Syntax **Eof(** *filenumber%* **)**

where: **is:**

filenumber% An integer expression identifying the open file to use.

See the **Open** statement for more information about assigning numbers to files when they are opened.

Erase Statement

See Also **Example**

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

Syntax **Erase** *Array* [, *Array*]

where: **is:**

Array The name of the array variable to re-initialize.

The effect of using **Erase** on the elements of a fixed array varies with the type of the element:

Element Type	**Erase Effect**
numeric	Each element set to zero.
variable length string	Each element set to zero length string.
fixed length string	Each element's string is filled with zeros.
Variant	Each element set to **Empty.**
user-defined type	Members of each element are cleared as if the members were array elements, i.e. numeric members have their value set to zero, etc.

Erl Function

See Also **Example** **Overview**

Returns the line number where an error was trapped.

Syntax **Erl**

Using a **Resume** or **On Error** statement after **Erl** will reset the return value for **Erl** to 0. To maintain the value of the line number returned by **Erl**, assign it to a variable.

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

Err Function

See Also **Example** **Overview**

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

Syntax **Err**

If you use a **Resume** or **On Error** statement after **Erl**, the return value for **Err** is reset to 0. To maintain the value of the line number returned by **Erl**, assign it to a variable.

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

Follow this link to the full list of **Trappable Errors** .

Err Statement

See Also **Example** **Overview**

Sets a run-time error code.

Syntax **Err** = *n%*

where:	is:
n%	An integer expression for the error code (between 1 and 32,767) 0 indicates that no run-time error has been trapped.

The **Err** statement is used to send error information between procedures.

Error Function

See Also **Example** **Overview**

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

Syntax **Error**[**$**] [**(** *errornumber%* **)**]

Where:	Is:
errornumber%	An integer between 1 and 32,767 for the error code.

If this argument is omitted, VCBasic returns the error message for the run-time error that has occurred most recently.

If no error message is found to match the errorcode, "" (a null string) is returned.

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

Here is a list of all Trappable Errors.

Error Statement

See Also **Example** **Overview**

Simulates the occurrence of a VCBasic or user-defined error.

Syntax **Error** *errornumber%*

where:	is:
errornumber%	An integer between 1 and 32,767 for the error code.

If an *errornumber%* is one that VCBasic already uses, the **Error** statement will simulate an occurrence of that error.

User-defined error codes should employ values greater than those used for standard VCBasic error codes. To help ensure that non-VCBasic error codes are chosen, user-defined codes should work down from 32,767.

If an **Error** statement is executed, and there is no error-handling routine enabled, VCBasic produces an error message and halts program execution. If an **Error** statement specifies an error code not used by VCBasic, the message "User-defined error" is displayed.

Exit Statement

See Also **Example**

Terminates Loop statements or transfers control to the original calling procedure.

Syntax **Exit** {**Do** | **For**| **Function** | **Sub**}

Use **Exit Do** inside a **Do...Loop** statement.

Use **Exit For** inside a **For...Next** statement.

When the **Exit** statement is executed from within a loop, control transfers to the statement after the Loop or Next statement. When used within a nested loop, an Exit statement moves control out of the immediately enclosing loop.

Use **Exit Function** inside a **Function...End Function** procedure.

Use **Exit Sub** inside a **Sub...End Sub** procedure.

Exp Function

See Also **Example**

Returns the value *e* (the base of natural logarithms) raised to a power.

Syntax **Exp(** *number* **)**

where:	is:
number	The exponent value for *e*.

If the variable to contain the return value has a data type **Integer**, **Currency**, or **Single**, the return value is a single-precision value. If the variable has a data type of **Long**, **Variant**, or **Double**, the value returned is a double-precision number.

[The constant *e* is approximately 2.718282]

FileAttr Function

See Also **Example**

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

Syntax **FileAttr(** *filenumber%* , *returntype* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
returntype	Must be either a 1 or a 2. 1=Returns the file mode* 2=Returns the operating system handle

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

*If *returntype* is 1, the following table lists the return values and corresponding file modes:

If the value is:	**The file Mode is:**
1	**Input**
2	**Output**
8	**Append**

FileCopy Statement

See Also   **Example**

Copies a file.

Syntax **FileCopy** *source$, destination$*

where:	**is:**
source$	A string expression for the name (and path) of the file to copy.
destination$	A string expression for the name (and path) for the copied file.

Wildcards (* or ?) are not allowed for either the *source$* or *destination$*. The *source$* file cannot be copied if it is opened by VCBasic for anything other than **Read** access.

FileDateTime Function

See Also   **Example**

Returns the last modification date and time for the specified file.

Syntax **FileDateTime**(*pathname$*)

where:	**is:**
pathname$	A string expression for the name of the file to query.

Pathname$ can contain path and disk information, but cannot include wildcards (* and ?).

FileLen Function

See Also   **Example**

Returns the length of the specified file.

Syntax **FileLen**(*pathname$*)

where:	**is:**
pathname$	A string expression that contains the name of the file to query.

FileLen returns a result of type Long.

Pathname$ can contain path and disk information, but cannot include wildcards (* and ?).

If the specified file is open, **FileLen** returns the length of the file before it was opened.

Fix Function

See Also **Example**

Returns the integer part of a number.

Syntax **Fix (** *number* **)**

where:	**is:**
number	Any valid numeric expression.

The return value's data type matches the type of the numeric expression. This includes **Variant** expressions, unless the numeric expression is a string (vartype 8) that evaluates to a number, in which case the data type for its return value is double (vartype 5). If the numeric expression is vartype 0 (empty), the data type for the return value is vartype 3 (long).

For both positive and negative *numbers,* **Fix** removes (truncates) the fractional part of the expression and returns the integer part only. For example, **Fix** (6.2) returns 6; **Fix** (-6.2) returns -6.

For...Next Statement

See Also **Example**

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

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

[*statementblock*]

[**Exit For**]

[*statementblock*]

Next [*counter*]

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

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. VCBasic compares the sign of (*start* - *end*) with the sign of *increment*. If the signs are the same, and *end* does not equal *start*, the **For...Next** loop is started. If not, the loop is omitted in its entirety.

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

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

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

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

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

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

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

Format Function

See Also **Example**

Returns a string from an expression; the string is formatted according to a specified format.

Syntax **Format**[**$**]**(** *expression* [, *format*] **)**

where:	**is:**
expression	The value to be formatted. It can be a number, Variant, or string.
format	A string expression representing the format to use. Select one of the topics below for a detailed description of format strings.

Format formats the *expression* as a number, date, time, or string depending upon the *format* argument.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string). As with any string, you must enclose the *format* argument in quotation marks ("").

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

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

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

For more information, see these topics:

Formatting Numbers

Formatting Dates and Times

Formatting Strings

FreeFile Function

See Also **Example**

Returns the lowest unused file number.

Syntax FreeFile

The **FreeFile** function is used when you need to supply a file number and want to make sure that you are not choosing a file number that is already in use.

The value returned can be used in a subsequent **Open** statement.

Function ... End Function Statement

See Also **Example**

Defines a function procedure.

[The purpose of a function is to produce and return a single value of a specified type. Use **Sub** to define a procedure with no return value.]

Syntax [Static] [Private] Function *name* **[(** [**Optional**]*parameter* [**As** *type*] ... **)]** **[As** *functype* **]**

name= *expression*

End Function

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

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 A function name.

parameter The argument(s) to pass to the function when it is called.

type The data type for the function arguments.

functype The data type for the return value.

name=expression The expression that sets the return value for the function.

Recursion is supported.

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

The *parameters* are specified as a comma-separated list of variable names. The data type of a parameter can be specified by using a 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 dimensions are not specified in the **Function** statement. All references to an array parameter within the body of the function must have a consistent number of dimensions.

You specify the return value for the function name using the *name=expression* assignment, where *name* is the name of the function and *expression* evaluates to a return value. If omitted, the value returned is 0 for numeric functions, an empty string ("") for string functions, and vartype 0 (Empty) for a return type of Variant.

If you declare a parameter as **Optional**, a procedure can omit its value when calling the function. Only parameters with **Variant** data types can be declared as optional, and all optional arguments must appear after all required arguments in the **Function** statement. The function **IsMissing** must be used to check whether an optional parameter was omitted by the user or not. Named parameters are described under the **Call** statement heading, but they can be used when the function is used in an expression as well.

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

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

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

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

FV Function

See Also **Example**

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

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

where: is:

rate Interest rate per period.

nper Total number of payment periods.

pmt Constant periodic payment per period.

pv Present value or the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).

due An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

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

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

Get Statement

See Also **Example**

Reads data from a file opened in **Random** or **Binary** mode and puts it in a variable.

Syntax **Get** [#] *filenumber%*, [*recnumber&*], *varname*

where: is:

# You may use this symbol or not. It has no effect.

filenumber% An integer expression identifying the open file from which to read.

recnumber& 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.

varname The name of the variable into which **Get** reads file data. *Varname* can be any variable except **Object, Application Data Type,** or **Array** variables (single array elements can be used).

For more information about how files are numbered when they're opened, see the **Open** statement.

Recnumber& is in the range 1 to 2,147,483,647. If it is omitted, the next record or byte is read.

The commas before and after the *recnumber&* **are required**, even if you do not supply a *recnumber&.*

For **Random** mode, the following rules apply:

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

For variable length String variables, **Get** reads two bytes of data that indicate the length of the string, then reads the data into *varname*.

For **Variant** variables, **Get** reads two bytes of data that indicate the type of the Variant, then reads the body of the Variant into *varname*. Note that Variants containing strings contain two bytes of data type information followed by two bytes of length followed by the body of the string.

User defined types are read as if each member were read separately, except no padding occurs between elements.

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

Get reads variables from the disk without record padding.

Variable length **Strings** that are not part of user defined types are not preceded by the two-byte string length. Instead, the number of bytes read is equal to the length of *varname*.

GetAttr Function

See Also **Example**

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

Syntax **GetAttr**(*pathname$*)

where:	is:
pathname$	A **String** expression for the name of the file, directory, or label to query.

Pathname$ cannot contain wildcards (* and ?).

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

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

GetField Function [VCBasic Extension]

See Also **Example**

Returns a substring from a source string.

Syntax **GetField**[$]**(** *string$* , *field_number%* , *separator_chars$* **)**

where:	is:
string$	A list of fields, divided by separator characters.
field_number%	The number of the field to return, starting with 1.
separator_chars$	The characters separating each field.

Multiple separator characters can be specified. If *field_number* is greater than the number of fields in the string, an empty string ("") is returned.

Global Statement

See Also **Example**

Declare Global variables for use in a VCBasic program.

Syntax **Global** *variableName* [**As** *type*] [,*variableName* [**As** *type*]] **...**

where:	**is:**
variableName	A variable name
type	The data type for *variableName*

Data declared using Global in the Common area of a macro is shared across all loaded macros.

The Global statement should be used only when macros are being designed to share data. If a variable is to be available only to all procedures of a particular macro, the variable should be delcared using the Dim statement in the Common area

If you attempt to load a macro containing a global variable that has the same name but a different data type as an existing global variable, the macro load will fail.

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

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

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

The available data types are:

Arrays

Numbers

Records

Strings

Variants

Objects

[Dialog is a data type, but you cannot use the **Global** statement to declare a dialog record.]

GoTo Statement

See Also **Example**

Transfers program control to the label specified.

Syntax **GoTo** { *label* }

where: **is:**

label A name beginning in the first column of a line of code and ending with a colon (:).

A *label* has the same format as any other Basic name. To be recognized as a *label*, a name must begin in the first column, and be followed immediately by a colon (:). Reserved words are not valid labels.

GoTo cannot be used to transfer control out of the current Function or Subprogram.

GroupBox Statement [VCBasic Extension]

See Also **Example**

Defines and draws a box that encloses sets of dialog box items, such as option boxes and check boxes, within a dialog box.

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

where: **is:**

x , y The upper left corner coordinates of the group box, relative to the upper left corner of the dialog box.

dx , dy The width and height of the group box.

text$ A string containing the title for the top border of the group box.

.id The optional string ID for the groupbox, used by the dialog statements that act on this control.

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

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

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

Hex Function

See Also **Example**

Returns the hexadecimal representation of a number (or numeric expression) as a string.

Syntax **Hex**[**$**]**(** *number* **)**

where: **is:**

number Any numeric expression that evaluates to a number.

If *number* is an integer, the return string contains up to four hexadecimal digits; otherwise, the value will be converted to a **Long** Integer, and the string can contain up to 8 hexadecimal digits.

To represent a hexadecimal number directly, precede the hexadecimal value with **&H**. For example, &H10 equals decimal 16 in hexadecimal notation.

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

Hour Function

See Also **Example**

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

Syntax **Hour(** *time* **)**

where:	**is:**
time	Any numeric or string expression that can evaluate to a date and time.

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

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

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

If ... Then ... Else

See Also **Example**

Executes alternative blocks of program code based on the logical values of one or more expressions.

Syntax A **If** *condition* **Then** *then_statement* **[Else** *else_statement* **]**

Syntax B **If** *condition* **Then**

statement_block

[ElseIf *expression* **Then**

*statement_block***]...**

[Else

statement_block **]**

End If

where:	**is:**
condition	Any expression that evaluates to TRUE (non-zero) or FALSE (zero).
then_statement	Any valid single expression.
else_statement	Any valid single expression.
expression	Any expression that evaluates to TRUE (non-zero) or FALSE (zero).
statement_block	Zero or more valid expressions. Separate expressions by colons (:) or list them on different lines.

When multiple statements are required in either the **Then** or **Else** clauses, use the block version (Syntax B) of the **If** statement.

'$Include Metacommand [VCBasic Extension]*

See Also **Example**

Includes statements from the specified file.

Syntax **'$Include: "***filename***"**

where: **is:**

filename The name and location (drive and path) of the file to include.

It is recommended (although not required) that you use a file extension of .SBH for *filename*.

If no directory or drive is specified, the compiler will search for *filename* on the source file search path.

All metacommands must begin with an apostrophe (') and are recognized by the compiler only if the command starts at the beginning of a line.

For compatibility with other versions of VCBasic, you can enclose the *filename* in single quotation marks (').

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

Input Function

See Also **Example**

Returns a string containing the characters read from a file.

Syntax **Input**[**$**]**(** *number% ,* [#]*filenumber%***)**

where: **is:**

number% The number of characters to be read from the file.

# You may use this symbol or not. It has no effect.

filenumber% An integer expression identifying the open file to read from.

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

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

To return a given number of bytes from a file, use the **InputB** function.

Input Statement

See Also **Example**

Reads data from a sequential file and assigns the data to variables.

Syntax A **Input** [#] *filenumber%* , *variable* [, *variable*]...

Syntax B **Input** [*prompt$,*] *variable* [, *variable*]...

where:	**is:**
#	You may use this symbol or not. It has no effect.
filenumber%	An integer expression identifying the open file to read from.
variable	The variable(s) to contain the value(s) read from the file.
prompt$	An optional string that prompts for keyboard input.

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

The list of *variables* is separated by commas.

If *filenumberr%* is not specified, the user is prompted for keyboard input.

If *prompt$* is omitted, users will be prompted with a "?".

InputBox Function

See Also **Example**

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

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

where:	**is:**
prompt$	A string expression containing the text to show in the dialog box.
title$	The caption to display in the dialog box's title bar.
default$	The string expression to display in the edit box as the default response.
xpos%, *ypos%*	Numeric expressions, specified in dialog box units, that determine the position of the dialog box.

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

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

If either *prompt$* or *default$* is omitted, nothing is displayed in the corresponding area.

Xpos% determines the horizontal distance between the left edge of the screen and the left border of the dialog box. *Ypos%* determines the horizontal distance from the top of the screen to the dialog box's upper edge.

If *xpos%* and *ypos%* are not entered, the dialog box's position defaults to centered roughly one third of the way down the screen. A horizontal dialog box unit is 1/4 of the average character width in the system font; a vertical dialog box unit is 1/8 of the height of a character in the system font.

Note: If you want to specify the dialog box's position, you must enter both of these arguments. If you enter one without the other, the box's position will be set to the default positioning.

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

InStr Function

See Also **Example**

Returns the character position of the first occurrence of one string within another string.
[To obtain the byte position of the first occurrence of one string within another string, use the **InStrB** function.]

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

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

where: is:

start% The position in *string1$* to begin the search. (1=first character in string.)

string1$ The string to search.

string2$ The string to find.

compare An integer expression for the method to use to compare the strings.
(0=case-sensitive, 1=case-insensitive.)

If *start%* is not specified, the search starts at the beginning of the string (equivalent to a *start%* of 1). *string1$* and *string2$* can be of any type. They will be converted to strings.

InStr returns a zero under the following conditions:

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

If either *string1$* or *string2$* is a null Variant , **Instr** returns a null Variant.

If *string2$* is a null string (""), **Instr** returns the value of *start%*.

If *compare* is 0, a case-sensitive comparison based on the ANSI character set sequence is performed.
If *compare* is 1, a case-insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system.
If *compare* is omitted, the module level default, as specified with **Option Compare**, is used.

IPmt Function

See Also **Example**

Returns the interest portion of a payment for a given period of an annuity.

Syntax **IPmt(** *rate , per , nper , pv , fv , due* **)**

where: is:

rate Interest rate per period.

per Particular payment period in the range 1 through *nper*.

nper Total number of payment periods.

pv Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).

fv Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).

due 0 if payments are due at the end of each payment period, and 1 if they are due at the beginning of the period.

The given interest rate is assumed to remain constant over the life of the annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%. (0.09 / 12)

IRR Function

See Also **Example**

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

Syntax **IRR(** *valuearray( ) , guess* **)**

where: is:

valuearray( ) An array containing cash flow values.

guess A ballpark estimate of the value returned by **IRR**.

valuearray() must have at least one positive value (representing a receipt) and one negative value (representing a payment).
All payments and receipts must be represented in the exact sequence.
The value returned by **IRR** will vary with the change in the sequence of cash flows.

In general, a *guess* value of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable estimate.

IRR is an iterative function. It improves a given guess over several iterations until the result is within 0.00001 percent.

If it does not converge to a result within 20 iterations, it signals failure.

IsMissing Function

See Also **Example**

Returns -1 (TRUE) if an optional parameter was not supplied by the user, 0 (FALSE) otherwise.

[In other words, TRUE indicates the optional parameter is missing, FALSE indicates that it is not missing (it was supplied).]

Syntax **IsMissing(** *argname* **)**

where:	is:
argname	An optional argument for a subprogram, function, VCBasic statement, or VCBasic function.

IsMissing is used in procedures that have optional arguments to find out whether the argument's value was supplied or not.

Kill Statement

See Also **Example**

Deletes files from a hard disk or floppy drive.

Syntax **Kill** *pathname$*

where:	is:
pathname$	A String expression that specifies a valid DOS file.

The *pathname$* specification can contain paths and wildcards.

Kill deletes files only, not directories. Use the **RmDir** function to delete directories.

LBound Function

See Also **Example**

Returns the lower bound of the subscript range for the specified array.

Syntax **LBound(** *arrayname* [, *dimension*] **)**

where:	is:
arrayname	The name of the array to use.
dimension	The dimension to use.

The dimensions of an array are numbered starting with 1.

If *dimension* is not specified, 1 is used as a default.

LBound can be used with **UBound** to determine the length of an array.

LCase Function

See Also **Example**

Returns a copy of a string, with all uppercase letters converted to lowercase.

Syntax **LCase**[**$**]**(** *string$* **)**

where:	is:
string$	A string, or an expression containing the string to use.

The translation is based on the country specified in the Windows Control Panel. **Lcase$** accepts expressions of type String. **LCase** accepts any type of argument, including numeric values, and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified the return type is String. If omitted the function will typically return a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

Left Function

See Also **Example**

Returns a string of a specified number of characters copied from the beginning of another string.

Syntax **Left**[**$**]**(** *string$*, *length%* **)**

where:	**is:**
string$	A string or an expression containing the string to copy.
length%	The number of characters to copy.

If *length%* is greater than the length of *string$,* **Left** returns the whole string.

Left$ accepts expressions of type String. **Left** accepts any type of *string$*, including numeric values, and will convert the input value to a string.

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

If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

To obtain a string of a specified number of bytes, copied from the beginning of another string, use the **LeftB** function.

Len Function

See Also **Example**

Returns the length of a string or variable.

Syntax A **Len(** *string$* **)**

Syntax B **Len(** *varname* **)**

where:	**is:**
string$	A string or an expression that evaluates to a string.
varname	A variable that contains a string.

If the argument is a string, the number of characters in the string is returned.

If the argument is a Variant variable, **Len** returns the number of bytes required to represent its value as a string.

If the argument is not a string or variant, the length of the built-in data type or user-defined type is returned.

If syntax B is used, and *varname* is a **Variant** containing a NULL, **Len** will return a Null Variant.

To return the number bytes in a string, use the **LenB** function.

Let (Assignment Statement)

See Also **Example**

Assigns an expression to a VCBasic variable.

Syntax **[Let]** *variable* **=** *expression*

where:	**is:**
variable	The name of a VC Basic variable to which the *expression* is assigned.
expression	The expression to assign to the variable.

The keyword **Let** is optional.

The **Let** statement can be used to assign a value or expression to a variable with a data type of **Numeric**, **String**, **Variant** or **Record** variable.

The **Let** statement can also assign a value to a record field or to an element of an array.

When assigning a value to a numeric or string variable, standard conversion rules apply.

Let differs from **Set** in that Set assigns a variable to an OLE object. For example,

Set o1 = o2 will set the object reference.

Let o1 = o2 will set the value of the default member.

Like Operator

See Also **Example**

Returns the value -1 (TRUE) if a string matches a pattern and the value 0 (FALSE) if the string doesn't match the pattern.

Syntax *string$* **LIKE** *pattern$*

where:	**is:**
string$	Any string expression.
pattern$	Any string expression to match to *string$*.

pattern$ can include the following special characters:

Character:	**Matches:**
?	A single character
*	A set of zero or more characters
#	A single digit character (0-9)

[*chars*] A single character in *chars*

[!*chars*] A single character not in *chars*

[*schar-echar*] A single character in range *schar* to *echar*

[!*schar-echar*] A single character not in range *schar* to *echar*

Both ranges and lists can appear within a single set of square brackets. Ranges are matched according to their ANSI values. In a range, *schar* must be less than *echar*.

If either *string$* or *pattern$* is NULL then the result value is NULL.

The **Like** operator respects the current setting of **Option Compare**.

For more information about operators, see **Expressions**.

Line Input Statement

See Also **Example**

Reads a line from a sequential file or keyboard into a string variable.

Syntax A **Line Input** [#] *filenumber% , varname$*

Syntax B **Line Input** [*prompt$,*] *varname$*

where:	**is:**
#	You may use this symbol or not. It has no effect.
filenumber%	An integer expression identifying the open file to read from.
prompt$	An optional string that can be used to prompt for keyboard input; it must be a literal string.
varname$	A string variable to contain the line read.

If specified, the *filenumber%* is the number used in the **Open** statement to open the file. If *filenumber%* is not provided, the line is read from the keyboard.

If *prompt$* is not provided, a prompt of "?" is used.

ListBox Statement

See Also **Example**

Defines a list box of choices for a dialog box.

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

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

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the list box.
text$	A string containing the selections for the list box.

stringarray$ An array of dynamic strings for the selections in the list box.

.field The name of the dialog-record field that will hold a number for the choice made in the list box.

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

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

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

A number representing the selection's position in the *text$* string is recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The numbers begin at 0. If no item is selected, it is -1. The *field* argument is also used by the dialog statements that act on this control.

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

Loc Function

See Also **Example**

Returns the current offset within an open file.

Syntax **Loc(** *filenumber%* **)**

where: **is:**

filenumber% An integer expression identifying the open file to query.

The *filenumber%* is the number used in the **Open** statement of the file.

For files opened in **Random** mode, **Loc** returns the number of the last record read or written.

For files opened in **Append**, **Input**, or **Output** mode, **Loc** returns the current byte offset divided by 128.

For files opened in **Binary** mode, **Loc** returns the offset of the last byte read or written.

Lock Statement

See Also **Example**

Controls access by other processes to some or all of an open file.

Syntax **Lock** [#]*filenumber%* [, [*start&*] [**To** *end&*]]

Unlock [#]*filenumber%* [, [*start&*] [**To** *end&*]]

where: **is:**

# You may use this symbol or not. It has no effect.

filenumber% An integer expression identifying the open file.

start& Number of the first record or byte offset to lock/unlock.

end& Number of the last record or byte offset to lock/unlock.

The *filenumber%* is the number used in the **Open** statement of the file.

For **Binary** mode, *start&*, and *end&* are byte offsets.

For **Random** mode, *start&*, and *end&* are record numbers.

If *start&* is specified without *end&*, then only the record or byte at *start&* is locked. If **To** *end&* is specified without *start&*, then all records or bytes from record number or offset 1 to *end&* are locked.

For **Input**, **Output** and **Append** modes, *start&*, and *end&* are ignored and the whole file is locked.

Lock and **Unlock** always occur in pairs with identical parameters. All locks on open files must be removed before closing the file, or unpredictable results will occur.

Lof Function

See Also **Example**

Returns the length in bytes of an open file specified by *filenumber%*.

.

Syntax **Lof(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file.

The *filenumber%* is the number used in the **Open** statement of the file.

Log Function

See Also **Example**

Returns the natural logarithm of a number.

Syntax **Log(** *number* **)**

where:	**is:**
number	Any valid numeric expression.

The return value is single-precision for an integer, currency or single-precision numeric expression.

The return value is double precision for a long, Variant or double-precision numeric expression.

Lset Statement

See Also **Example**

Copies one string to another, or assigns a user-defined type variable to another.

Syntax A **Lset** *string$* **=** *string-expression*

Syntax B **Lset** *variable1 = variable2*

where:	is:
string$	A string or string expression to contain the copied characters.
string-expression	An expression containing the string to copy.
variable1	A variable with a user-defined type to contain the copied variable.
variable2	A variable with a user-defined type to copy.

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$.*

If *string$* is longer than *string-expression*, all characters of *string-expression* are copied into *string$*, filling it from left to right. All leftover characters of *string$* are replaced with spaces.

In Syntax B, the number of characters copied is equal to the length of the shorter of *variable1* and *variable2.*

Lset cannot be used to assign variables of different user-defined types if either contains a **Variant** or a variable-length string.

LTrim Function

See Also **Example**

Returns a copy of a string with all leading space characters removed.

Syntax **LTrim**[$]**(** *string$* **)**

where:	is:
string$	A string or expression containing a string to copy.

Ltrim$ [with $] accepts expressions of type String. **Ltrim [without the $]** accepts any type of expression, including numeric values, and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

Me

Example **See Also**

Refers to the currently used OLE2 automation object.

Syntax **Me**

The alias **Me** refers to the current form, and is normally used in functions in place of the formname.

Some VCBasic modules are attached to application objects and VCBasic subroutines are invoked when that application object encounters events. A good example is a user visible button that triggers a Basic routine when the user clicks the mouse on the button.

Subroutines in such contexts can use the variable **Me** to refer to the object that triggered the event (i.e., which button was clicked). The programmer can use **Me** in all the same ways as any other object variable except that **Me** cannot be **Set**.

Mid Function

See Also **Example**

Returns a portion of a string, starting at a specified character position within the string.

Syntax **Mid**[**$**]**(** *string$***,** *start%*[**,** *length%*] **)**

where:	**is:**
string$	A string or expression that contains the string to change.
start%	The starting position in *string$* to begin replacing characters.
length%	The number of characters to replace.

Mid accepts any type of *string$*, including numeric values, and will convert the input value to a string. If the *length%* argument is omitted, or if *string$* is smaller than *length%*, then **Mid** returns all characters in *string$*. If *start%* is larger than *string$*, then **Mid** returns a null string ("").

The index of the first character in a string is 1.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string$* is Null, a Variant of vartype 1 (Null) is returned. **Mid$** requires the string argument to be of type string or variant. **Mid** allows the string argument to be of any datatype.

To return a specified number of bytes from a string, use the **MidB** function. With the **MidB** function, *start%* specifies a byte position and *length%* specifies a number of bytes.

To modify a portion of a string value, see **Mid Statement**.

Mid Statement

See Also **Example**

Replaces part (or all) of one string with another string, starting at a specified location.

Syntax **Mid (** *stringvar$***,** *start%*[**,** *length%*] **) =** *string$*

where:	**is:**
stringvar$	The string to change.
start%	An expression for the position to begin replacing characters.
length%	An expression for the number of characters to replace.
string$	The string to place into another string.

If the *length%* argument is omitted, or if there are fewer characters in *string$* than specified in *length%*, then **Mid** replaces all the characters from the *start%* to the end of the *string$*.
If *start%* is larger than the number of characters in the indicated *stringvar$*, then **Mid** appends *string%* to *stringvar$*.

If *length%* is greater than the length of *string$*, then *length%* is set to the length of *string$*. If *start%* is greater than the number of characters in *stringvar$*, an illegal function call error will occur at runtime. If *length%* plus *start%* is greater than the length of *stringvar$*, then only the characters up to the end of *stringvar$* are replaced.

Mid never changes the number of characters in *stringvar$*.

The index of the first character in a string is 1.

To replace a specified number of bytes in a string with those from another string, use the **MidB** statement. With the **MidB** statement, *start%* specifies a byte position and *length%* specifies a number of bytes.

Minute Function

See Also **Example**

Returns an integer indicating the minute component (0-59) of a date-time value.

Syntax **Minute(** *time* **)**

where:	is:
time	Any expression that can evaluate to a date-time value.

Minute accepts any type of *time*, including strings, and will attempt to convert the input value to a date value. If it cannot convert it, a run-time error occurs.

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

MkDir Statement

See Also **Example**

Creates a new directory.

Syntax **MkDir** *path$*

where:	is:
path$	A string expression identifying the new default directory to create.

The syntax for *path$* is:

[*drive*:] [\] *directory* [*directory*]

The *drive* argument is optional. If *drive* is omitted, **MkDir** makes a new directory on the current drive. The *directory* argument is any directory name.

Month Function

See Also **Example**

Returns an integer from 1 to 12 indicating the month component of a date-time value.

Syntax **Month(** *date* **)**

where:	is:
date	Any expression that evaluates to a date-time value.

It accepts any type of *date*, including strings, and will attempt to convert the input value to a date value. If it cannot convert it, a run-time error occurs.

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

Msgbox Function

See Also **Example**

Displays a message dialog box and returns a value (1-7) indicating which button the user selected.

Syntax **Msgbox(** *prompt$* ,[*buttons%*][, *title$*] **)**

where: is:

prompt$ The text to display in a dialog box.

buttons% An integer value for the buttons, the icon, and the default button choice to display in a dialog box.

title$ A string expression containing the title for the message box.

prompt$ must be no more than 1,024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.

buttons% is the sum of three values, one from each of the following groups:

	Value	**Description**
Group 1: Buttons	0	OK only
	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
Group 2: Icons	16	Critical Message (STOP)
	32	Warning Query (?)
	4	Warning Message (!)

	8	
	64	Information Message (i)
Group 3: Defaults	0	First button
	256	Second button
	512	Third button

If *buttons%* is omitted, Msgbox displays a single OK button.

After the user clicks a button, **Msgbox** returns a value indicating the user's choice. The return values for the Msgbox function are:

Value	**Button Pressed**
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Msgbox Statement

See Also **Example**

Displays a prompt in a message dialog box.

Syntax **MsgBox** *prompt$* , [*buttons%*][, *title$*]

where:	**is:**

prompt$ The text to display in a dialog box.

buttons% An integer value for the buttons, the icon, and the default button choice to display in a dialog box.

title$ A string expression containing the message box's title.

Prompt$* must be no more than 1,024 characters long, including spaces. If *prompt$ is greater than 255 characters without intervening spaces, it will be truncated after the 255th character.

If *title$* is omitted, nothing will be diplayed in the title bar.

If *buttons%* is omitted, **Msgbox** displays a single OK button in the message box.

To specify the button choices, the icons, and which button will be selected by default, you must include a value for *buttons%* . The value for *buttons%* will be the sum of three values, one from each of the following groups:

Groups	**Value**	**Description**
Buttons	0	OK only
	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
Icons	16	Critical Message (STOP)
	32	Warning Query (?)
	48	Warning Message (!)
	64	Information Message (i)
Default Selection	0	First button

Second button
2
5
6

Third button
5
1
2

Name Statement

See Also **Example**

Renames a file or moves a file from one directory to another.

Syntax **Name** *oldfilename$* **As** *newfilename$*

where:	**is:**
oldfilename$	A string expression containing the file to rename.
newfilename$	A string expression containing the new name for the file.

A path can be part of either filename argument. If the paths are different, the file is moved to the new directory.

A file must be closed in order to be renamed. If the file *oldfilename$* is open or if the file *newfilename$* already exists, VCBasic generates an error message.

New Operator

See Also **Example**

Allocates and initializes a new OLE2 object of the named class.

Syntax **Set** *objectVar* **= New** *className*

Dim *objectVar* **As New** *className*

where:	**is:**
objectVar	The OLE2 object to allocate and initialize.
className	The class to assign to the object.

In the **Dim** statement, **New** marks *objectVar* so that a new object will be allocated and initialized when *objectVar* is first used. If *objectVar* is not referenced, then no new object will be allocated.

Note: An object variable that was declared with **New** will allocate a second object if *objectVar* is **Set** to **Nothing** and referenced again.

Nothing Function

See Also **Example** **Overview**

Returns an object value that doesn't refer to an object.

Syntax **Set** *variableName* **= Nothing**

where:	**is:**
variableName	The name of the object variable to set to Nothing.

Nothing is the value object variables have when they do not refer to an object, either because the have not been initialized yet or because they were explicitly **Set** to **Nothing**. For example:

> **If Not** *objectVar* **Is Nothing then**
> *objectVar.Close*
> **Set** *objectVar* **= Nothing**
> **End If**

NPV Function

See Also **Example**

Returns the net present value of an investment based on a stream of periodic cash flows and a constant interest rate.

Syntax **NPV** *(rate , valuearray( ))*

where: **is:**

rate Discount rate per period. If the discount rate is 12% per period, *rate* is the decimal equivalent, i.e. 0.12.

valuearray( ) An array containing cash flow values.

Valuearray( ) must have at least one positive value (representing a receipt) and one negative value (representing a payment). All payments and receipts must be represented in the exact sequence. The value returned by **NPV** will vary with the change in the sequence of cash flows.

NPV uses future cash flows as the basis for the net present value calculation. If the first cash flow occurs at the beginning of the first period, its value should be added to the result returned by **NPV** and must not be included in *valuearray().*

Null Function

See Also **Example**

Returns a **Variant** value set to NULL.

Syntax **Null**

Null is used to set a Variant to the Null value explicitly, as follows:
variableName **= Null**

Note that Variants are initialized by VCBasic to the empty value, which is different from the null value.

Object Class

See Also Example

A class that provides access to OLE2 automation objects.

Syntax **Dim** *variableName* **As Object**

where:	is:
variableName	The name of the object variable to declare.

To create a new object, first dimension a variable, using the **Dim** statement, then **Set** the variable to the return value of **CreateObject** or **GetObject**, as follows:

Dim *OLE2* **As Object**

Set*OLE2* **= CreateObject(**"spoly.cpoly"**)**

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

OLE2.reset

Oct Function

See Also Example

Returns the octal representation of a number as a string.

Syntax **Oct**[$]**(** *number* **)**

where:	is:
number	A numeric expression for the number to convert to octal.

If the numeric expression has a data type of **Integer**, the returned string contains up to six octal digits.

If the numeric expression is not an Integer, it will be converted to a data type of **Long**, and the returned string can contain up to 11 octal digits.

To represent an octal number directly, precede the octal value with **&O** (this is the letter "O" and not a zero). For example, &O10 equals decimal 8 in octal notation.

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

OKButton Statement

See Also Example

Determines the position and size of an OK button in a dialog box.

Syntax **OKButton** *x* , *y* , *dx* , *dy* [, **.***id*]

where: is:

x , y The position of the OK button relative to the upper left corner of the dialog box.

dx , dy The width and height of the button.

.id An optional identifier for the button.

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

.id is an optional identifier used by the dialog statements that act on this control.

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

On...Goto Statement

See Also **Example**

Branch to a label in the current procedure based on the value of a numeric expression.

Syntax **ON** *numeric-expression* **GoTo** *label1* [,*label2 , ...*]

where: is:

numeric-expression Any numeric expression that evaluates to a positive number.

label1 , label2 A label in the current procedure to branch to if *numeric-expression* evaluates to 1, 2, etc.

The **On ... GoTo** statement branches the control to any one of the locations specified by *label1*, *label2*, *...labeln*, depending on the value of *numeric-expression*.

If *numeric expression* evaluates to 0 or to a number greater than the number of labels following **GoTo**, the program continues at 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 Error Statement

See Also **Example** **Overview**

Enables an error-handling routine by specifying the location of the desired routine within the current procedure.

On Error can also be used to disable an error-handling routine. Unless an **On Error** statement is used, any run-time error will be fatal, that is, VCBasic will terminate the execution of the program.

Syntax **ON [Local] Error** {**GoTo** *label* [**Resume Next**] **GoTo 0**}

where: is:

label A string used as a label in the current procedure to identify the lines of code that process errors.

An **On Error** statement is composed of the following parts:

Part **Definition**

Local Keyword allowed in error-handling routines at the procedure level. Used to ensure compatibility with other variants of Basic.

GoTo label Enables the error-handling routine that starts at *label* . If the designated label is not in the same procedure as the On Error statement, VCBasic generates an error message.

Resume Next Designates that error-handling code is handled by the statement that immediately follows the statement that caused an error. At this point, use the Err function to retrieve the error-code of the run-time error.

GoTo 0 Disables any error handler that has been enabled.

When it is referenced by an **On Error GoTo** *label* statement, an error-handler is enabled. Once this enabling occurs, a run-time error will result in program control switching to the error-handling routine and "activating" the error handler. The error handler remains active from the time the run-time error has been trapped until a **Resume** statement is executed in the error handler.

If another error occurs while the error handler is active, VCBasic will search for an error handler in the procedure that called the current procedure (if this fails, VCBasic will look for a handler belonging to the caller's caller, and so on). If a handler is found, the current procedure will terminate, and the error handler in the calling procedure will be activated.

It is an error (No Resume) to execute an **End Sub** or **End Function** statement while an error handler is active. The **Exit Sub** or **Exit Function** statement can be used to end the error condition and exit the current procedure.

Open Statement

See Also **Example**

Opens a file or device for input or output.

Syntax **Open** *filename$* [**For** *mode*] [**Access** *access*] [*lock*] **As [**#**]** *filenumber%* [**Len =** *reclen*]

where: **is:**

filename$ A string or string expression for the name of the file to open.

mode One of the following keywords:

Input Get data from the file sequentially.

Output Output data to the file sequentially.

Append Add data to the file sequentially.

Random Access a file with fixed-length records randomly.

Binary Access a file with arbitrary data randomly.

access One of the following keywords:

Read Read data from the file only.

Write Write data to the file only.

Read Write Read or write data to the file.

Lock One of the following keywords to designate access by other processes:

Shared Read or write available on the file.

Lock Read Read data only.

Lock Write Write data only.

Lock Read Write No read or write available.

filenumber% An integer or expression containing the integer to assign to the open file (between 1 and 255).

reclen The length of the records (for Random or Binary files only).

A file must be opened before any input/output operation can be performed on it.

If *filename$* does not exist, it is created when opened in **Append**, **Binary**, **Output** or **Random** modes.

If *mode* is not specified, it defaults to **Random**.

If *access* is not specified for **Random** or **Binary** modes, *access* is attempted in the following order: **Read Write**, **Write**, **Read**.

If *lock* is not specified, *filename$* can be opened by other processes that do not specify a *lock*, although that process cannot perform any file operations on the file while the original process still has the file open.

You may use the # symbol or not. It has no effect.

Use the **FreeFile** function to find the next available value for *filenumber%*.

Reclen is ignored for **Input**, **Output**, and **Append** *modes*.

OptionButton Statement

See Also **Example**

Defines the position and text associated with an option button in a dialog box.

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

where:	is:
x , *y*	The position of the button relative to the upper left corner of the dialog box.
dx , *dy*	The width and height of the button.
text$	A string to display next to the option button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

You must have at least two **OptionButton** statements in a dialog box. You use these statements in conjunction with the **OptionGroup** statement.

A *dy* value of 12 typically accommodates text in the system font.

To enable the user to select an option button by typing a character from the keyboard, precede the character in *text$* with an ampersand (&).

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

OptionGroup Statement

See Also **Example**

Groups a series of option buttons under one heading in a dialog box.

Syntax **OptionGroup** *.field*

where:	is:
.field	A value for the option button selected by the user: the value will be 0 when the first option button is selected, 1 when the for the second button is selected, and so on.

The **OptionGroup** statement is used in conjunction with **OptionButton** statements to set up a series of related options.

The **OptionGroup** Statement begins the definition of the option buttons and establishes the dialog-record field that will contain the option selection.

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

Details

Argument	Description
.field	The dialog-record field that will indicate the current option selection. It will contain a value of zero when the choice associated with the first **OptionButton** statement is selected, a value of 1 when the choice associated with the second **OptionButton** statement is chosen, and so on.

Option Base Statement

See Also **Example** **Overview**

Specifies the default lower bound to use for array subscripts.

Syntax **Option Base** *lowerBound%*

where:	is:
lowerBound	A number or an expression containing a number for the default lower bound: must be either 0 or 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.

Only one **Option Base** statement is allowed per module.

Option Compare Statement

See Also **Example**

Specifies the default method for string comparisons: either case-sensitive or case-insensitive.

Syntax **Option Compare** { **Binary** | **Text** }

where:	means:
Binary	Comparisons are case-sensitive (i.e., lowercase and uppercase letters are different).
Text	Comparisons are not case-sensitive.

Binary comparisons compare strings based upon the ANSI character set.

Text comparisons are based upon the relative order of characters as determined by the country code setting for your system.

Option Explicit Statement

See Also **Example**

Specifies that all variables in a module ***must*** be explicitly declared.

Syntax **Option Explicit**

By default, VCBasic automatically declares any variables that do not appear in a **Dim**, **Global**, **Redim**, or **Static** statement. **Option Explicit** causes such variables to produce a "Variable Not Declared" error.

PasswordBox Function

See Also **Example**

Returns a string entered by the user without echoing it to the screen.

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

where:	**is:**
prompt$	A string expression containing the text to show in the dialog box
title$	The caption for the dialog box's title bar
default$	The string expression shown in the edit box as the default response.
xpos% , *ypos%*	The position of the dialog box, relative to the upper left corner of the screen.

The **PasswordBox** function displays a dialog box containing a prompt. Once the user has entered text, or made the button choice being prompted for, the contents of the box are returned. The user's keystrokes will not be echoed in the input box.

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

If either *prompt$* or *default$* is omitted, nothing is displayed in the corresponding area.

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

Note: To specify the dialog box's position, you must enter both of these arguments. If you enter one without the other, the default positioning is used.

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

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

Picture Statement

See Also **Example**

Defines a picture control in a dialog box

Syntax **Picture** *x , y , dx , dy , filename$, type%* [, **.***id*]

where:	**is:**
x , y	The position of the picture relative to the upper left corner of the dialog box.
dx , dy	The width and height of the picture.
filename$	The name of the bitmap file (a file with .BMP extension) where the picture is located.
type	An integer for the location of the bitmap (0=*filename$*, 3=Windows Clipboard).
.id	An optional identifier used by the dialog statements that act on this control.

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

Note: The picture will be scaled equally in both directions and centered if the dimensions of the picture are not proportional to *dx* and *dy*.

If *type%* is 3, *filename$* is ignored.

If the picture is not available (the file *filename$* doesn't exist, doesn't contain a bitmap, or there is no bitmap on the Clipboard), the picture control will display the picture frame and the text "(missing picture)". This behavior can be changed by adding 16 to the value of *type%* to total 16 or 19. If *type%* is 16 (bitmap) or 19 (clipboard) and the picture is not available, then a runtime error occurs.

Pmt Function

See Also **Example**

Returns a constant periodic payment amount for an annuity or a loan.

Syntax **Pmt** *(rate , nper , pv , fv , due)*

where:	**is:**
rate	Interest rate per period.
nper	Total number of payment periods.
pv	Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
fv	Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Rate is assumed to be constant over the life of the loan or annuity.

If payments are on a monthly schedule, then, for example, *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%. [. 09 / 12]

PPmt Function

See Also **Example**

Returns the principal portion of the payment for a given period of an annuity.

Syntax **PPmt** *(rate , per , nper , pv , fv , due)*

where: is:

rate Interest rate per period.

per Particular payment period in the range 1 through *nper*.

nper Total number of payment periods.

pv Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).

fv Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).

due An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Rate is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Print Statement

See Also **Example**

Prints data to an open file or to the screen.

Syntax **Print** [# *filenumber%* ,] *expressionlist* [{ ; | , }]

where: is:

# You may use this symbol or not. It has no effect.

filenumber% An integer expression identifying the open file to use.

expressionlist A numeric, string, and Variant expression containing the list of values to print.

The **Print** statement outputs data to the specified *filenumber%*. *filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information.

If *filenumber%* is omitted, the **Print** statement outputs data to the screen.

If the *expressionlist* is omitted, a blank line is written to the file.

The values in *expressionlist* are separated by either a semi-colon (";") or a comma (",") . A semi-colon indicates that the next value should appear immediately after the preceding one without intervening white space. A comma indicates that the next value should be positioned at the next print zone. Print zones begin every 14 spaces.

The optional [{;|,}] argument at the end of the **Print** statement determines where output for the next **Print** statement to the same output file should begin. A semi-colon will place output immediately after the output from this **Print** statement on the current line; a comma will start output at the next print zone on the current line. If neither separator is specified, a CR-LF pair will be generated and the next **Print** statement will print to the next line.

Special functions **Spc** and **Tab** can be used inside **Print** statement to insert a given number of spaces and to move the print position to a desired column.

The **Print** statement supports only elementary VCBasic data types. See **Input** for more information on parsing this statement.

PushButton Statement

See Also **Example**

Defines a custom push button.

Syntax A **PushButton** *x* , *y* , *dx* , *dy* , *text$* [, **.***id*]

Syntax B **Button** *x, y, dx, dy, text$* [, **.***id*]

where:	is:
x , *y*	The position of the button relative to the upper left corner of the dialog box.
dx , *dy*	The width and height of the button.
text$	The name for the push button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

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

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

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

Put Statement

See Also **Example**

Writes a variable to a file opened in **Random** or **Binary** mode.

Syntax **Put** [#] *filenumber%* , [*recnumber&*], *varname*

where:	is:
#	You may use this symbol or not. It has no effect in VCBasic.
filenumber%	An integer expression identifying the open file to use.
recnumber&	A **Long** expression containing the record number or the byte offset at which to start writing.
varname	The name of the variable containing the data to write.

Filenumber% is the identifying number assigned to the file when it was opened. See the **Open** statement for more information.

Recnumber& is in the range 1 to 2,147,483,647. If *recnumber&* is omitted, the next record or byte is written.

Note: The commas before and after *recnumber%* are ***required***, even if no *recnumber&* is specified.

Varname can be any variable except **Object**, **Application Data Type** or **Array** variables (single array elements can be used).

For **Random** mode, the following apply:

- Blocks of data are written to the file in chunks whose size is equal to the size specified in the **Len** clause of the **Open** statement. If the size of *varname* is smaller than the record length, the record is padded to the correct record size. If the size of variable is larger than the record length, an error occurs.

- For variable length String variables, **Put** writes two bytes of data that indicate the length of the string, then writes the string data.

- For Variant variables, **Put** writes two bytes of data that indicate the type of the Variant, then it writes the body of the Variant into the variable. Note that Variants containing strings contain two bytes of type information, followed by two bytes of length, followed by the body of the string.

- User defined types are written as if each member were written separately, except no padding occurs between elements.

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

- **Put** writes variables to the disk without record padding.

- Variable length **Strings** that are not part of user defined types are not preceded by the two byte string length.

PV Function

See Also **Example**

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

Syntax **PV** *(rate , nper , pmt , fv , due)*

where:	**is:**
rate	Interest rate per period.
nper	Total number of payment periods.
pmt	Constant periodic payment per period.
fv	Future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Rate is assumed constant over the life of the annuity. If payments are on a monthly schedule, then *rate*, for example, will be 0.0075 if the annual percentage rate on the annuity or loan is 9%. [.09 / 12]

Randomize Statement

See Also **Example**

Seeds the random number generator.

Syntax **Randomize** [*number%*]

where: **is:**

number% An integer value between -32768 and 32767.

If no *number%* argument is given, VCBasic uses the **Timer** function to initialize the random number generator.

Rate Function

See Also **Example**

Returns the interest rate per period for an annuity or a loan.

Syntax **Rate** *(nper , pmt , pv , fv , due , guess)*

where: **is:**

nper Total number of payment periods.

pmt Constant periodic payment per period.

pv Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).

fv Future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).

due An integer value for when the payments are due (0=end of each period, 1= beginning of the period)

guess A ballpark estimate for the rate returned.

In general, a guess of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable value for *guess*.

Rate is an iterative function: it improves the given value of *guess* over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

ReDim Statement

See Also **Example** **Overview**

Changes the upper and lower bounds of a dynamic array's dimensions.

Syntax **ReDim** [**Preserve**] *variableName* **(** *subscriptRange* , ... **)** [**As** [**New**] *type*] , ...

where: **is:**

variableName The variable array name to redimension. It must begin with a letter and contain only letters, numbers, and underscores. Variable names may also be delimited by brackets. Except for other brackets, any character may be used inside the brackets.

subscriptRange The new upper and lower bounds for the array.

type	The type for the data elements in the array.

ReDim re-allocates memory for the dynamic array to support the specified dimensions, and can optionally re-initialize the array elements. **ReDim** cannot be used at the module level; it must be used inside of a procedure.

The **Preserve** option is used to change the last dimension in the array while maintaining its contents. If **Preserve** is not specified, the contents of the array are re-initialized. Numbers will be set to zero (0). Strings and Variants will be set to empty ("").

The *subscriptRange* is of the format:

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

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

A dynamic array is normally created by using **Dim** to declare an array without a specified *subscriptRange*. The maximum number of dimensions for a dynamic array created in this fashion is 8. If you need more than 8 dimensions, you can use the **ReDim** statement inside of a procedure to declare an array that has not previously been declared using **Dim** or **Global**. In this case, the maximum number of dimensions allowed is 60.

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

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

The **ReDim** statement cannot be used to change the number of dimensions of a dynamic array once the array has been given dimensions. It can only change the upper and lower bounds of the dimensions of the array. The **LBound** and **UBound** functions can be used to query the current bounds of an array variable's dimensions.

Care should be taken to avoid **ReDim**ing an array in a procedure that has received a reference to an element in the array in an argument; the result is unpredictable.

Rem Statement
Example

Identifies a line of code as a comment in a VCBasic program. (from REMark)

Syntax A: **Rem** *comment*

Syntax B: **'** *comment*

where:	**is:**
comment	The text of the comment.

Everything from **Rem** or the single quote (') to the end of the line is ignored by the program.

Metacommands (e.g., **$CSTRINGS**) **must** be preceded by the single quote comment form.

A comment is text that documents the program. Comments (except for metacommands) have no effect on the program. If the first character in a comment is a dollar sign ($), the comment will be interpreted as a metacommand

Reset Statement

See Also **Example**

Closes all open disk files and writes any data remaining in the operating system buffers to disk.

Syntax **Reset**

Resume Statement

See Also **Example** **Overview**

Halts an error-handling routine and resumes execution.

Syntax A **Resume Next**

Syntax B **Resume** *label*

Syntax C **Resume** [**0**]

where:	**is:**
label	The label that identifies the statement to go to after handling an error.

When the **Resume Next** statement is used, control is passed to the statement that immediately follows the statement in which the error occurred.

When the **Resume [0]** statement is used, control is passed to the statement in which the error occurred.

When the **Resume** *label* statement is used, control is passed to the statement that immediately follows the specified label

The location of the error handler that has caught the error determines where execution will resume. If an error is trapped in the same procedure as the error handler, program execution will resume with the statement that caused the error. If an error is located in a different procedure from the error handler, program control reverts to the statement that last called out the procedure containing the error handler.

Right Function

See Also **Example**

Returns a string of a specified number of characters copied from the end of another string.

Syntax **Right**[**$**]**(** *string$***,** *length%* **)**

where:	**is:**
string$	A string or expression containing the string to copy.
length%	The number of characters to copy.

If *length%* is greater than the length of *string$,* **Right** returns the whole string.

Right accepts any type of *string$,* including numeric values, and will convert the input value to a string.

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

If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

To obtain a string of a specified number of bytes, copied from the end of another string, use the **RightB** function.

RmDir Statement

See Also **Example**

Removes a directory.

Syntax **RmDir** *path$*

where:	**is:**
path$	A string expression identifying the directory to remove. The syntax for *path$* is: [*drive*:] [\] *directory* [*directory*]

The *drive* argument is optional. The *directory* argument is a directory name.

The directory to be removed must be empty, except for the working (.) and parent (..) directories.

Rnd Function

See Also **Example**

Returns a single precision random number between 0 and 1.

Syntax **Rnd** [**(** *number!* **)**]

where: **is:**

number! A numeric expression to specify how to generate the random numbers. (<0=use the number specified, >0=use the next number in the sequence, 0=use the number most recently generated.)

If *number!* is omitted, **Rnd** uses the next number in the sequence to generate a random number.

Rnd will generate the same sequence of random numbers each time it is executed unless the random number generator is re-initialized by the **Randomize** statement.

Rset Statement

See Also **Example**

Right aligns one string inside another string.

Syntax **Rset** *string$* = *string-expression*

where: **is:**

string$ The string to contain the right-aligned characters.

string-expression The string containing the characters to put into *string$*.

If *string$* is longer than *string-expression,* the leftmost characters of *string$* are replaced with spaces.

If *string$* is shorter than *string-expression*, only the leftmost characters of *string-expression* are copied.

Rset cannot be used to assign variables of different user-defined types.

RTrim Function

See Also **Example**

Copies a string and removes any trailing spaces.

Syntax **RTrim**[$]**(** *string$* **)**

where:	**is:**
string$	An expression that evaluates to a string.

RTrim accepts any type of *string* including numeric values and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If it is omitted the function will return a **Variant** of vartype 8 (string). If the value of *string* is NULL, a Variant of vartype 1 (Null) is returned.

Second Function

See Also **Example**

Returns an integer from 0 to 59 to indicate the second component of a date-time value.

Syntax **Second(** *time* **)**

where:	**is:**
time	An expression containing a date time value.

Second accepts any type of *time* including strings and will attempt to convert the input value to a date value. If it cannot convert it, a run-time error occurs.

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

Seek Function

See Also **Example**

Returns the current file position for an open file.

Syntax **Seek(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying an open file to query.

Filenumber% is the number assigned to the file when it was opened. See the **Open** statement for more information.

For files opened in **Random** mode, **Seek** returns the number of the next record to be read or written. For all other modes, **Seek** returns the file offset for the next operation. The first byte in the file is at offset 1, the second byte is at offset 2, etc.

The return value is a **Long**.

Seek Statement

See Also **Example**

Sets the position within an open file for the next read or write operation.

Syntax **Seek** [#] *filenumber% , position&*

where:	**is:**
filenumber%	An integer expression identifying an open file to query.

position& A numeric expression for the starting position of the next read or write operation (record number or byte offset).

If you write to a file after seeking beyond the end of the file, the file's length is extended. VCBasic will return an error message if a **Seek** operation is attempted that specifies a negative or zero position.

Filenumber% is an integer expression identifying the open file to **Seek** in. See the **Open** statement for more details.

For files opened in **Random** mode, *position&* is a record number; for all other modes, *position&* is a byte offset. *Position&* is in the range 1 to 2,147,483,647. The first byte or record in the file is at position 1, the second is at position 2, etc.

Select Case Statement

See Also **Example**

Executes one branch a series of statements, depending on the value of an expression.

Syntax **Select Case** *testexpression*

[**Case** *expressionlist*

[*statement_block*]]

[**Case** *expressionlist*

[*statement_block*]]

.

.

[**Case Else**

[*statement_block*]]

End Select

where:	**is:**
testexpression	Any expression containing a variable to test.
expressionlist	One or more expressions that contain a possible value for *testexpression*.
statement_block	The statements to execute if *testexpression* equals *expressionlist*.

When there is a match between *testexpression* and one of the values in *expressionlist*, the *statement_block* following the **Case** clause is executed. When the next **Case** clause is reached, execution control goes to the statement following the **End Select** statement.

The *expressionlist(s)* can be a comma-separated list of expressions of the following forms:

- *expression*
- *expression* **To** *expression*
- **Is** *comparison_operator expression*

The type of each *expression* must be compatible with the type of *testexpression*.

Note that when the **To** keyword is used to specify a range of values, the smaller value must appear first.

The *comparison_operator* used with the **Is** keyword is one of: <, >, =, <=, >=, <>.

Each *statement_block* can contain any number of statements on any number of lines.

SendKeys Statement

See Also **Example**

Send keystrokes to an active Windows application.

Syntax **SendKeys** *string$* [, *wait%*]

where:	**is:**
string$	An expression containing the characters to send.
wait%	A numeric expression to determine whether to wait until all keys are processed before continuing program execution (-1=wait, 0=don't wait).

The keystrokes are represented by characters of *string.*

The default value for *wait* is 0 (FALSE).

SendKeys on the IBM does not wait for the keyboard to unlock before processing the keys; this can cause characters to be dropped or lost.

SendKeys can send keystrokes only to the currently active application. Therefore, you have to use the **AppActivate** or AppClassActivate statement to activate an application before sending keys (unless it is already active).

To specify an ordinary character, enter this character in the *string*. For example, to send character 'a' use "a" as *string.* Several characters can be combined in one string: *string* "abc" means send 'a', 'b', and 'c'.

To specify that Shift, Alt, or Control keys should be pressed simultaneously with a character, prefix the character with

+	to specify Shift
%	to specify Alt
^	to specify Control.

Parentheses can be used to specify that the Shift, Alt, or Control key should be pressed with a group of characters. For example, "%(abc)" is equivalent to "%a%b%c".

Since '+', '%', '^' ,'(' and ')' characters have special meaning to **SendKeys**, they must be enclosed in braces if they need to be sent with **SendKeys**. For example *string* "{%}" specifies a percent character '%'.

Other characters that need to be enclosed in braces are '~' which stands for a newline or "Enter" if used by itself and braces themselves: use {{} to send '{' and {}} to send '}'. Brackets '[' and ']' do not have special meaning to **SendKeys** but might have special meaning in other applications, therefore, they need to be enclosed inside braces as well.

To specify that a key needs to be sent several times, enclose the character in braces and specify the number of keys sent after a space: for example, use {X 20} to send 20 'X' characters.

To send one of the non-printable keys use a special keyword inside braces:

Key to Send	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	The NumLock key and processing is not currently enabled within Windows operating systems.
Page Down	{PGDN}
Page Up	{PGUP}
Right Arrow	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Up Arrow	{UP}

To send one of the function keys F1-F15, simply enclose the name of the key inside braces. For example, to send F5 use "{F5}"

Note that special keywords can be used in combination with +, %, and ^. For example: %{TAB} means Alt-Tab. Also, you can send several special keys in the same way as you would send several normal keys: {UP 25} sends 25 Up arrows.

SendKeys cannot be used to send keys to an application that was not designed to run under Windows.

Set Statement

See Also **Example** **Overview**

Assigns a variable to an OLE2 object.

Syntax **Set** *variableName* **=** *expression*

where:	**is:**
variableName	An object variable or a Variant variable.
expression	An expression that evaluates to an object--typically a function, an object member, or **Nothing**.

The following example shows the syntax for the **Set** statement:

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

Note: If you omit the keyword **Set** when assigning an object variable, VCBasic will try to copy the default member of one object to the default member of another. This usually results in a runtime error:

' Incorrect code - tries to copy default member!
OLE2 **= GetObject(,**"spoly.cpoly"**)**

Set differs from **Let** in that Let assigns an expression to a VCBasic variable. For example,

Set o1 = o2	will set the object reference.
Let o1 = o2	will set the value of the default member.

SetAttr Statement

See Also **Example**

Sets the attributes for a file.

Syntax **SetAttr** *pathname$, attributes%*

where:	**is:**
pathname$	A string expression containing the filename to modify.
attributes %	An integer containing the new attributes for the file.

Wildcards are not allowed in *pathname$*.

If the file is open, you can modify its attributes, but only if it is opened for **Read** access.

These are the attributes that can be modified:

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

SetField Function [VCBasic Extension]*

See Also **Example**

Replaces a field within a string and returns the modified string.

Syntax **SetField**[$]**(** *string$***,** *field_number%***,** *field$***,** *separator_chars$* **)**

where:	is:
string$	A string consisting of a series of fields, separated by *separator_char$*.
field_number%	An integer for the field to replace within *string$*.
field$	An expression containing the new value for the field.
separator_char$	A string containing the character(s) used to separate the fields in *string$*.

separator_char$* can contain multiple separator characters, . If more than one separator character was specified (in *separator_chars$), the first one will be used as the separator character.

The *field_number%* starts with 1. If *field_number%* is greater than the number of fields in the string, the returned string will be extended with separator characters to produce a string containing the specified number of fields.

It is legal for the new *field$* value to be a different size than the old value.

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

Sgn Function

See Also **Example**

Returns a value indicating the sign of a number — positive, negative, or zero.

Syntax **Sgn(** *number* **)**

where:	**is:**
number	An expression for the number to evaluate

The value that the **Sgn** function returns depends on the sign of *number.*

For *numbers* > 0, **Sgn** (*number*) returns 1.

For *numbers* = 0, **Sgn** (*number*) returns 0.

For *numbers* < 0, **Sgn** (*number*) returns -1.

Shell Function

See Also **Example**

Starts an executable program and returns its task ID.

Syntax **Shell(** *pathname$* , [*windowstyle%*] **)**

where:	**is:**
pathname$	The name of the program to execute
windowstyle%	An integer value for the style of the program's window (1-7).

Shell returns the task ID for the program, which is a unique number that identifies the running program.

Pathname$* can be the name of any valid .COM, .EXE., .BAT, or .PIF file. Arguments or command line switches can be included. If *pathname$ is not a valid executable file name, or if **Shell** cannot start the program, an error message occurs.

Windowstyle% is one of the following values:

Value	**Window Style**
1	Normal window with focus
2	Minimized with focus
3	Maximized with focus
4	Normal window without focus
7	Minimized without focus

If *windowstyle%* is not specified, the default of *windowstyle%* = 1 is assumed (normal window with focus).

Sin Function

See Also **Example**

Returns the sine of an angle specified in radians.

Syntax **Sin**(*number*)

where:	**is:**
number	An expression containing the angle in radians.

The return value will be between -1 and 1.

The return value is single-precision if the angle is an integer, currency or single-precision value.

The return value is double precision for a long, Variant or double-precision value.

The angle must be specified in radians, and can be either positive or negative.

To convert degrees to radians, multiply by (PI/180). The value of PI is 3.14159.

Space Function

See Also **Example**

Returns a string of spaces.

Syntax **Space**[**$**]**(** *number* **)**

where:	**is:**
number	A numeric expression for the number of spaces to return.

number can be any numeric data type, but will be rounded to an integer.

number must be between 0 and 32,767.

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

Spc Function

See Also **Example**

Prints a number of spaces to file or to the screen.

Syntax **Spc (** *n* **)**

where:	**is:**
n	An integer for the number of spaces to output.

The **Spc** function can only be used inside a **Print** statement.

When the **Print** statement is used, the **Spc** function will use the following rules for determining the number of spaces to output:

- If *n* is less than the total line width, **Spc** outputs *n* spaces.
- If *n* is greater than the total line width, **Spc** outputs *n* **Mod** *width* spaces.
- If the difference between the current print position and the output line width (call this difference x) is less than *n* or *n* **Mod** *width*, then **Spc** skips to the next line and outputs *n* - x spaces.

To set the width of a print line, use the **Width** statement.

SQLClose Function

See Also **Example**

Disconnects from an ODBC data source connection that was established by **SQLOpen**.

Syntax **SQLClose (** *connection&* **)**

where:	**is:**
connection&	A named argument that must be a long integer, returned by **SQLOpen**.

The return is a variant. Success returns 0 and the connection is subsequently invalid. If the connection is not valid, -1 is returned.

SQLError Function

See Also **Example**

Can be used to retrieve more detailed information about errors that might have occurred when making an ODBC function call.

Returns errors for the last ODBC function and the last connection.

Syntax **SQLError (** *destination()* **)**

where: **is:**

destination A two dimensional array in which each row contains one error. A named argument that is required, must be an array of variants.

There is no return value.

The fields are: 1) character string indicating the ODBC error class/subclass,

2) numeric value indicating the data source native error code,

3) text message describing the error.

If there are no errors from a previous ODBC function call, then a 0 is returned in the caller's array at (1,1).

If the array is not two dimensional or does not provide for the return of the three fields above, then an error message is returned in the caller's array at (1,1).

SQLExecQuery Function

See Also **Example**

Executes an SQL statement on a connection established by **SQLOpen**.

Syntax **SQLExecQuery (** *connection& , query$* **)**

where: **is:**

connection& A named argument, required. A long integer, returned by **SQLOpen**.

query$ A string containing a valid SQL statement. The return is a variant.

SQLExecQuery returns the number of columns in the result set for SQL SELECT statements

For UPDATE, INSERT, or DELETE it returns the number of rows affected by the statement.

For any other SQL statement, it returns 0.

If the function is unable to execute the query on the specified data source, or if the connection is invalid, a negative error code is returned.

If **SQLExecQuery** is called and there are any pending results on that connection, the pending results are replaced by the new results.

SQLGetSchema Function

See Also **Example**

Returns a variety of information, including information on the data sources available, current user ID, names of tables, names and types of table columns, and other data source/database related information.

Syntax **SQLGetSchema (***connection& , action% , qualifier$, ref()* **)**

where:	is:
connection	A long integer returned by **SQLOpen.**
action%	Required.
qualifier$	Required.
ref()	A variant array for the results appropriate to the action requested, must be an array even if only one dimension with one element. The return is a variant.

A negative return value indicates an error. A -1 is returned if the requested information cannot be found or if the connection is not valid. The destination array must be properly dimensioned to support the action or an error will be returned. Actions 2 and 3 are not currently supported. Action 4 returns all tables and does not support the use of the *qualifier*. Not all database products and ODBC drivers support all actions.

Action	Meaning
1	List of available datasources (dimension of *ref()* is one)
2	List of databases on the current connection (not supported)
3	List of owners in a database on the current connection (not supported)
4	List of tables on the specified connection
5	List of columns in a the table specified by *qualifier. (ref()* must be two dimensions). Returns column name and SQL data type.
6	The user ID of the current connection user.
7	The name of the current database.

8 The name of the data source for the current connection.

9 The name of the DBMS the data source users (e.g., Oracle).

10 The server name for the data source.

11 The terminology used by the data source to refer to owners.

12 The terminology used by the data source to refer to a table.

13 The terminology used by the data source to refer to a qualifier.

14 The terminology used by the data source to refer to a procedure.

SQLOpen Function

See Also **Example**

Establishes a connection to an ODBC data source specified in *connectStr*.

Returns a connection ID in the return value and the completed connection string in *outputStr*.

If the connection cannot be established, then a negative number ODBC error is returned.

Syntax **SQLOpen (** *connectStr$, outputStr$, prompt%* **)**

where:	**is:**
connectStr	A named argument, a required parameter.
outputStr	A variable to contain the returned connection string. Optional
prompt	Specifies when the driver dialog box is displayed. Optional.

The content of *connectStr* is described in the Microsoft Programmer's Reference Guide for ODBC. An example string might be "DSN=datasourcename; UID=myid; PWD=mypassword". The return must be a long.

If *prompt* is omitted, **SQLOpen** uses 2 as the default.

Prompt Value	**Meaning**
1	Driver dialog is always displayed.
2	Driver dialog is displayed only when the specification is not sufficient to make the connection.
3	The same as 2, except that dialogs that are not required are grayed and cannot be modified.
4	Driver dialog is not displayed. If the connection is not successful, an error is returned.

SQLRequest Function

See Also **Example**

Establishes a connection to the data source specified in *connectionStr,* executes the SQL statement contained in *query*, returns the results of the request in the *ref()* array, and closes the connection.

Syntax **SQLRequest(** *connectionStr$, query$, outputStr$, prompt% , columnNames% , ref()* **)**

where: is:

connectionStr$ A required argument.

query$ A required argument.

outputStr$ A variable to contain the completed connection string.

prompt% An integer that specifies when driver dialog boxes are displayed (see **SQLOpen**).

columnNames% An integer with a value of 0 or nonzero. When *columnNames* is nonzero, column names are returned as the first row of the *ref()* array. If *columnNames* is omitted, the default is 0.

ref() A required argument that is a two dimensional variant array.

In the event that the connection cannot be made, the query is invalid, or other error condition, a negative number error is returned.

In the event the request is successful, the positive number of results returned or rows affected is returned. Other SQL statements return 0.

The arguments are named arguments. The return is a variant.

SQLRetrieve Function

See Also **Example**

Fetches the results of a pending query on the connection specified by *connection* and returns the results in the *destination()* array.

Syntax **SQLRetrieve(** *connection& , destination() , [maxColumns% , maxRows% , columnNames% , rowNumbers% , fetchFirst%*] **)**

where: is:

connection& A long.

destination() A two dimensional variant array.

maxColumns% An integer and an optional parameter used to specify the number of columns to be retrieved in the request.

maxRows% An integer and an optional parameter used to specify the number of rows to be retrieved in the request.

columnNames% An integer and an optional parameter; defaults to 0.

rowNumbers% An integer and an optional parameter; defaults to 0.

fetchFirst% An integer and an optional parameter, defaults to 0.

The return value is the number of rows in the result set or the *maxRows* requested.

If the function is unable to retrieve the results on the specified connection, or if there are not results pending, -1 is returned. If no data is found, the function returns 0.

The arguments are named arguments. The return is a variant.

If *maxColumns* or *maxRows* are omitted, the array size is used to determine the maximum number of columns and rows retrieved, and an attempt is made to return the entire result set. Extra rows can be retrieved by using **SQLRetrieve** again and by setting *fetchFirst* to 0. If *maxColumns* specifies fewer columns than are available in the result, **SQLRetrieve** discards the rightmost result columns until the results fit the specified size.

When *columnNames* is nonzero, the first row of the array will be set to the column names as specified by the database schema. When r*owNumbers* is nonzero, row numbers are returned in the first column of *destination()*. **SQLRetrieve** will clear the user's array prior to fetching the results.

When *fetchFirst* is nonzero, it causes the result set to be repositioned to the first row if the database supports the function. If the database does not support repositioning, the result set -1 error will be returned.

If there are more rows in the result set than can be contained in the *destination()* array or than have been requested using *maxRows*, the user can make repeated calls to **SQLRetrieve** until the return value is 0.

SQLRetrieveToFile Function

See Also Example

Fetches the results of a pending query on the connection specified by *connection&* and stores them in the file specified by *destination$*.

Syntax SQLRetrieveToFile(*connection& , destination$* [, *columnNames% , columnDelimiter$*] **)**

where:	**is:**
connection&	A required argument. A long integer
destination$	A required argument. A string containing the full path and filename to be used for storing the results.
columnNames%	An integer; when nonzero, the first row of the file will be set to the column names as specified by the database schema. If zero, column names are not returned. If omitted, the default is 0.
columnDelimiter$	Specifies the string to be used to delimit the fields within each row. If *columnDelimiter* is omitted, a horizontal tab is used to delimit fields.

Upon successful completion of the operation, the return value is the number of rows in the result set. If the function is unable to retrieve the results on the specified connection, or if there are not results pending, --1 is returned.

The arguments are named arguments. The return is a variant.

Sqr Function

See Also Example

Returns the square root of a number.

Syntax Sqr(*number* **)**

where:	**is:**
number	An expression containing the number to use.

The return value is single-precision for an integer, currency or single-precision numeric expression, double precision for a long, Variant or double-precision numeric expression.

Static Statement

See Also **Example**

Declares variables and allocate storage space within procedures.

Syntax **Static** *variableName* [**As** *type*] [,*variableName* [**As** *type*]] **...**

where: is:

variableName The name of the variable to declare.

type The data type of the variable.

Variables declared with the **Static** statement retain their value as long as the program is running. The syntax of **Static** is exactly the same as the syntax of the **Dim** statement.

All variables of a procedure can be made static by using the **Static** keyword in a definition of that procedure See **Function** or **Sub** for more information.

StaticComboBox Statement

See Also **Example**

Creates a combination of a list of choices and a text box.

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

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

where: is:

x , y The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.

dx , dy The width and height of the combo box in which the user enters or selects text.

text$ A string containing the selections for the combo box.

stringarray$ An array of dynamic strings for the selections in the combo box.

.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 **StaticComboBox** statement is equivalent to the **ComboBox** or **DropComboBox** statement, but the list box of **StaticComboBox** always stays visible. All dialog functions and statements that apply to the **ComboBox** apply to the **StaticComboBox** as well.

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

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

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

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

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

Stop Statement

See Also **Example**

Halts program execution.

Syntax **Stop**

Stop statements can be placed anywhere in a program to suspend its execution. Although the **Stop** statement halts program execution, it does **not** close files or clear variables.

Str Function

See Also **Example**

Returns a string representation of a number.

Syntax **Str**[$]**(** *number* **)**

where:	**is:**
number	The number to be represented as a string.

The precision in the returned string is single-precision for an integer or single-precision numeric expression, double precision for a long or double-precision numeric expression, and currency precision for currency. Variants return the precision of their underlying vartype.

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

StrComp Function

See Also **Example**

Compares two strings and returns an integer specifying the result of the comparison.

Syntax **StrComp(** *string1$* , *string2$* [, *compare%*] **)**

where:	**is:**
string1$	Any expression containing the first string to compare.
string2$	The second string to compare.
compare%	An integer for the method of comparison (0=case-sensitive, 1=case-insensitive).

StrComp returns one of the following values:

Value	**Meaning**
-1	*string1$* < *string2$*
0	*string1$* = *string2$*
1	*string1$* > *string2$*

Null	Either *string1$* or *string2$* or both = Null

If *compare%* is 0, a case sensitive comparison based on the ANSI character set sequence is performed.

If *compare%* is 1, a case insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system.

If *compare%* is omitted, the module level default as specified with **Option Compare** is used.

The *string1* and *string2* arguments are both passed as Variants. Therefore, any type of expression is supported. Numbers will be automatically converted to strings.

String Function

See Also **Example**

Returns a string consisting of a repeated character.

Syntax A **String**[**$**]**(** *number , Character%* **)**

Syntax B **String**[**$**] **(** *number , string-expression$* **)**

where:	is:
number	Specifies the length of the string to be returned.
Character%	A numeric expression that contains an integer for the decimal ANSI code of the character to use.
string-expression$	A string argument, the first character of which becomes the repeated character.

number must be between 0 and 32,767.

Character% is a numeric expression that VCBasic will evaluate as an integer from 0 to 255.

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

Sub ... End Sub Statement

See Also **Example**

Defines a subprogram procedure.

Syntax **[Static] [Private] Sub** *name* **[(** [**Optional**] *parameter* **[As** *type*] **, ...)]**

End Sub

where:	is:
name	The name of the subprogram.
parameter	A comma-separated list of parameter names.
Type	A data type for *parameter*

A call to a subprogram stands alone as a separate statement. (See the **Call** statement). Recursion is supported.

The data type of a parameter can be specified by using a type character or by using the **As** clause. Record parameters are declared by using an **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the **Sub** statement. All references to an array within the body of the subprogram must have a consistent number of dimensions.

If a *parameter* is declared as **Optional**, its value can be omitted when the function is called. Only Variant parameters can be declared as optional, and all optional parameters must appear after all required parameters in the **Sub** statement. The function **IsMissing** must be used to check whether an optional parameter was omitted by the user or not. See the **Call** statement for more information on using named parameters.

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

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

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

VCBasic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it will modify the variable passed by the caller.

The MAIN subprogram has a special meaning. In many implementations of Basic, MAIN will be called when the module is "run". The MAIN subprogram is not allowed to take arguments.

Use **Function** to define a procedure that has a return value.

Tab Function

See Also **Example**

Moves the current print position to the column specified.

Syntax **Tab (** *n* **)**

where:	is:
n	The new print position to use.

The **Tab** function can be only be used inside the **Print** statement.

The leftmost print position is position number 1.

When the **Print** statement is used, the **Tab** function will use the following rules for determining the next print position:

1. If *n* is less than the total line width, the new print position is *n*.

2. If *n* is greater than the total line width, the new print position is *n* **Mod** *width* .

3. If the current print position is greater than *n* or *n* **Mod** *width*, **Tab** skips to the next line and sets the print position to *n* or *n* **Mod** *width*.

To set the width of a print line, use the **Width** statement.

Tan Function

See Also **Example**

Returns the tangent of an angle in radians.

Syntax **Tan**(*number*)

where:	is:
number	An expression containing the angle in radians.

number is specified in radians, and can be either positive or negative.

The return value is single-precision if the angle is an integer, currency or single-precision value.

The return value is double precision for a long, Variant or double-precision value.

To convert degrees to radians, multiply by PI/180. [The value of PI is approximately 3.14159, so for a quick conversion, multiply the number of degrees by 0.0174532.]

Text Statement

See Also **Example**

Places line(s) of text in a dialog box.

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

where:	is:
x , y	The upper left corner coordinates of the text area, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the text area.
text$	A string containing the text to appear in the text area defined by *x , y*.
.id	An optional identifier used by the dialog statements that act on this control.

If the width of *text$* is greater than *dx*, the spillover characters wrap to the next line. This will continue as long as the height of the text area established by *dy* is not exceeded. Excess characters are truncated.

By preceding an underlined character in *text$* with an ampersand (&), you enable a user to press the underlined character on the keyboard and position the cursor in the combo or text box defined in the statement immediately following the **Text** statement.

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

TextBox Statement

See Also **Example**

Creates a text box in a dialog box.

Syntax **TextBox [NoEcho]** *x , y , dx , dy , .field*

where: is:

NoEcho Often used for passwords, this keyword displays asterisks (*) instead of thel characters entered.

x , y The upper left corner coordinates of the text box, relative to the upper left corner of the dialog box.

dx , dy The width and height of the text box area.

.field The name of the dialog record field to hold the text string.

A *dy* value of 12 will usually accommodate text in the system font.

When the user selects the OK button, or any pushbutton other than cancel, the text string entered in the text box will be recorded in *.field*.

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

Time Function

See Also **Example**

Returns a string representing the current time.

Syntax **Time**[**$**]

The **Time** function returns an eight character string. The format of the string is "***hh:mm:ss***" where ***hh*** is the hour, ***mm*** is the minutes and ***ss*** is the seconds. The hour is specified in military style [a 24-hour clock], and ranges from 0 (midnight) to 23 (11 pm).

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

Time Statement

See Also **Example**

Sets the current system time.

Syntax Time[**$**] = *expression*

where: is:

expression An expression that evaluates to a valid time.

When **Time** (with the dollar sign "$") is used, the *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 [military time]. Thus, 6:00 P.M. must be entered as 18:00:00.

Time (without the **$**) accepts both 12 and 24 hour clocks.

If the dollar sign '$' is omitted, *expression* can be a string containing a valid date, a **Variant** of vartype 7 (date) or 8 (string).

If *expression* is not already a Variant of vartype 7 (date), **Time** attempts to convert it to a valid time.

Time recognizes time separator characters as defined in the International section of the Windows Control Panel.

Timer Function

See Also **Example**

Returns the number of seconds that have elapsed since midnight.

Syntax **Timer**

The Timer function can be used in conjunction with the **Randomize** statement to seed the random number generator.

TimeSerial Function

See Also **Example**

Returns a time as a a **variant** of vartype 7 (date/time) for a specific hour, minute, and second.

Syntax **TimeSerial(** *hour%*, *minute%*, *second%* **)**

where:	is:
hour%	A numeric expression for an hour (0-23).
minute%	A numeric expression for a minute (0-59).
second%	A numeric expression for a second (0-59).

You also can specify relative times for each argument by using a numeric expression representing the number of hours, minutes, or seconds before or after a certain time.
For example: **(5 - 2, 20 + 10, 0)** represents 3:30 am.

TimeValue Function

See AlsoExample

Returns a time value for a specified string.

Syntax **TimeValue(** *time$* **)**

where:	is:
time$	A string representing a valid date time value.

The **TimeValue** function returns a **Variant** of vartype 7 (date/time) that represents a time between 0:00:00 and 23:59:59, or 12:00:00 A.M. and 11:59:59 P.M., inclusive.

Trim Function

See Also **Example**

Returns a copy of a string after removing all leading and trailing spaces.

Syntax **Trim**[$]**(** *string* **)**

where:	**is:**
string	An expression containing the string to trim.

Trim$ accepts expressions of type String. **Trim** [without the $] accepts any type of *string* including numeric values and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is String. If omitted, the function typically returns a **Variant** of vartype 8 (String). If the value of *string* is NULL, a Variant of vartype 1 (Null) is returned.

Type Statement

See Also **Example**

Declares a user-defined type. [A user-defined type is sometimes referred to as a *record type* or a *structure type*.]

Syntax **Type** *userType*
field1 **As** *type1*
field2 **As** *type2*
...
End Type

where:	**is:**
userType	A user-defined type.
field1 , *field2*	The name of a field in the user-defined type.
type1 , *type2*	A data type: Integer, Long, Single, Double, Currency, String, String**length*, Variant, or another user-defined type.

The user-defined type declared by **Type** can then be used in the **Dim** statement to declare a record variable.

field cannot be an array. However, arrays of records are allowed.

The **Type** statement is not valid inside of a procedure definition. To access the fields of a record, use notation of the form:

recordName.fieldName

To access the fields of an array of records, use notation of the form:

arrayName(index).fieldName

Typeof Function

See Also   Example

Returns a value indicating whether an object is of a given class (-1=TRUE, 0=FALSE).

Syntax **If Typeof** *objectVariable* **Is** *className* **then. . .**

where:	is:
objectVariable	The object to test.
className	The class to compare the object to.

Typeof can only be used in an **If** statement and cannot be combined with other boolean operators. That is, **Typeof** can only be used exactly as shown in the syntax above.

To test if an object does *not* belong to a class, use the following code structure:

```
If Typeof objectVariable Is className Then
Else
        Rem Perform some action.
End If
```

UBound Function

See Also   Example

Returns the upper bound of the subscript range for the specified array.

Syntax **UBound(** *arrayname* [, *dimension*] **)**

where:	is:
arrayname	The name of the array to use.
dimension	The dimension to use.

The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is used as a default.

LBound can be used with **UBound** to determine the length of an array.

UCase Function

See Also   Example

Returns a copy of a string after converting all lower case letters to upper case.

Syntax **UCase**[$]**(** *string* **)**

where:	is:

string An expression that evaluates to a string.

The translation is based on the country specified in the Windows Control Panel.

Ucase$ accepts expressions of type string. **UCase** accepts any type of argument and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (String). If the value of *string* is Null, a Variant of vartype 1 (Null) is returned.

Unlock Statement

See Also **Example**

Controls access to an open file.

Syntax **Unlock** [#]*filenumber%* [, { *record&* | [*start&*] **To** *end&* }]

where: is:

filenumber% An integer expression identifying the open file.

record& Number of the starting record to unlock.

start& Number of the first record or byte offset to lock/unlock.

end& Number of the last record or byte offset to lock/unlock.

The *filenumber%* is the number used in the **Open** statement of the file.

For **Binary** mode, *start&*, and *end&* are byte offsets. For **Random** mode, *start&*, and *end&* are record numbers. If *start&* is specified without *end&*, then only the record or byte at *start&* is locked. If **To** *end&* is specified without *start&*, then all records or bytes from record number or offset 1 to *end&* are locked.

For **Input**, **Output** and **Append** modes, *start&*, and *end&* are ignored and the whole file is locked.

Lock and **Unlock** always occur in pairs with identical parameters. All locks on open files must be removed before closing the file, or unpredictable results will occur.

Val Function

See Also **Example**

Returns the numeric value of the first number found in the specified string.

Syntax **Val(** *string$* **)**

where: is:

string$ A string expression containing a number.

Spaces in the source string are ignored. If the first number found is zero, or if no number is found, **Val** retuns zero.

VarType Function

See Also **Example** **Overview**

Returns a value (0-9) that specifies the Variant type of the Variant variable.

Syntax **VarType(** *varname* **)**

where:	**is:**
varname	The **Variant** variable to use.

The value returned by **VarType** is one of the following:

Ordinal	Representation
0	(Empty)
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	String
9	Object

Weekday Function

See Also **Example**

Returns the day of the week for the specified date-time value.

Syntax **Weekday(** *date* **)**

where:	**is:**
date	An expression containing a date time value.

The **Weekday** function returns an integer between 1 and 7, inclusive (1=Sunday, 7=Saturday).

Weekday accepts any expression, including strings, and attempts to convert the input value to a date value.
If it cannot convert it, a run-time error occurs.

The return value is a **Variant** of vartype 2 (Integer). If the value of *date* is NULL, a Variant of vartype 1 (Null) is returned.

While ... Wend

See Also **Example**

Controls a repetitive action. The *condition* is tested; if it is non-zero (TRUE), the *statementblock* is executed. This process is repeated until *condition* becomes zero (FALSE).

Syntax **While** *condition*

statementblock

Wend

where: is:

condition An expression that evaluates to TRUE (non-zero) or FALSE (zero).

statementblock A series of statements to execute if *condition* is TRUE.

The *statementblock* statements are until *condition* becomes 0 (FALSE).

The **While** statement is included in VCBasic for compatibility with older versions of Basic. The **Do** statement is a more general and powerful flow control statement.

Width Statement

See Also **Example**

Sets the output line width for an open file.

Syntax **Width** [#]*filenumber% , width%*

where: is:

# You may use this symbol or not. It has no effect.

filenumber% An integer expression for the open file to use.

width% An integer expression for the width of the line (0 to 255). [This is the number of characters to be on a line before a new line is started.]

Filenumber% is the number assigned to the file when it is opened. See the **Open** statement for more information.

A value of zero (0) for *width%* indicates there is no line length limit.

The default *width%* for a file is zero (0).

With Statement

See Also **Example**

Executes a series of statements on a specified variable.

Syntax **With** *variable*
statement_block
End With

where: is:

variable The variable to be changed by the statements in *statement_block*.

statement_block The statements to execute.

Variable can be an object or a user-defined type.

With statements can be nested.

Write Statement

See Also **Example**

Writes data to an open sequential file.

Syntax **Write** #*filenumber%* [, *expressionlist*]

where:	is:
filenumber%	An integer expression for the open file to use.
expressionlist	One or more values to write to the file.

The file must be opened in **Output** or **Append** mode. *Filenumber%* is the number assigned to the file when it is opened. (See the **Open** statement for more information.)

If *expressionlist* is omitted, the **Write** statement writes a blank line to the file. (See **Input** for more information.)

Year Function

See Also **Example**

Returns the year component of a date-time value.

Syntax **Year(** *date* **)**

where:	is:
date	An expression that can evaluate to a date time value.

The **Year** function returns an integer between 100 and 9999, inclusive.

Year accepts any type of *date*, including strings, and will attempt to convert the input value to a date value. If **Year** cannot convert it, a run-time error occurs.

The return value is a **Variant** of vartype 2 (Integer). If the value of *date* is NULL, a Variant of vartype 1 (Null) is returned.

Data Types

See Also

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. Data types can also be specified by using a type character, which is used as a suffix to the name of a function or variable.

The characters are:

$	Dynamic String
%	Integer
&	Long integer
!	Single precision floating point
#	Double precision floating point
@	Currency exact fixed point

In any case, the variable can only contain data of the declared type. Variables of user-defined type must be explicitly declared. VCBasic supports standard Basic numeric, string, record and array data. VCBasic also supports Dialog Box Records and Objects (which are defined by the application).

Arrays

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

Numbers

These are the five numeric types and their ranges; negative numbers are in red.

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	-1.797693134862315d+308	-4.94065645841247d-308
	0.0	
	2.2250738585072014d-308	1.797693134862315d+308
Currency	-922,337,203,685,477.5808	922,337,203,685,477.5807

Numeric values are always signed.

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

Integer constants can be expressed in decimal, octal, or hexadecimal notation. Decimal constants are expressed by simply using the decimal representation. To represent an octal value, precede the constant with "&O" or "&o" (e.g., &o177). Note that that is the letter "o" and not a zero. To represent a hexadecimal value, precede the constant with "&H" or "&h" (e.g., &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, e.g., *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. See the **Begin Dialog** statement for more information.

Strings

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

Step 1: Define a dialog box

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

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

Step 2: Write a dialog box function

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

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

Step 3: Display the dialog box

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

Step 1: Create an object variable to access the application

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

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

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

appvariable.object.property

appvariable.object.method

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

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

Option 1: Trap error within body of code

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

Option 2: Trap error using error handler

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

Derived Trigonometric Functions

A number of trigonometric functions can 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
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

Assert Statement [VCBasic Extension]

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

Syntax **Assert** *condition*

where:	**is:**
condition	A numeric or string expression that can evaluate to TRUE or FALSE.

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

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

CCur Function

See Also **Example**

Converts an expression to the data type **Currency**.

Syntax **CCur(** *expression* **)**

where:	**is:**
expression	Any expression that evaluates to a number.

CCur accepts any type of *expression.* Numbers that do not fit in the Currency data type result in an "Overflow" error. Strings that cannot be converted result in a "Type Mismatch" error.

Variants containing null result in an "Illegal Use of Null" error.

CDbl Function

See Also **Example**

Converts an expression to the data type **Double**.

Syntax **CDbl(** *expression* **)**

where:	**is:**
expression	Any expression that evaluates to a number.

CDbl accepts any type of *expression.*

Strings that cannot be converted to a double-precision floating point result in a "Type Mismatch" error.

Variants containing null result in an "Illegal Use of Null" error.

Chr Function

See Also **Example**

Returns the one-character string corresponding to a character code.

Syntax **Chr**[$]**(** *charcode* **)**

where:	**is:**
charcode	An integer representing the character to be returned.

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

To obtain a byte representing a given character, use **ChrB**.

CInt Function

See Also **Example**

Converts an expression to the data type **Integer** by rounding.

Syntax **CInt(** *expression* **)**

where: is:

expression Any expression that can evaluate to a number.

After rounding, the resulting number must be within the range of -32767 to 32767, or an error occurs.

Strings that cannot be converted to an integer result in a "Type Mismatch" error.

Variants containing null result in an "Illegal Use of Null" error.

ComboBox Statement

See Also **Example**

Creates a combination text box and list box in a dialog box.

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

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

where: is:

x , y The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.

dx , dy The width and height of the combo box in which the user enters or selects text.

text$ A string containing the selections for the combo box.

stringarray$ An array of dynamic strings for the selections in the combo box.

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

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

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

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

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

Command Function

See Also **Example**

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

Syntax **Command**[**$**]

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

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

Const Statement

See Also **Example**

Use the **Const** statement to declare symbolic constants for use in a VCBasic program. VCBasic is a strongly typed language. The available data types for constants are numbers and strings.

Syntax **[Global] Const** *constantName* [**As** *type*]= *expression* [,*constantName* [**As** *type*]= *expression*] **...**

where:	**is:**
constantName	The variable name to contain a constant value.
type	The data type of the constant (**Number** or **String)**
expression	Any expression that evaluates to a constant number.

Instead of using the **As** clause, the type of the constant can be specified by using a type character as a suffix (# for numbers, $ for strings) to the *constantName*. If no type character is specified, the type of the *constantName* is derived from the type of the expression.

If **Global** is specified, the constant is validated at module load time. If the constant has already been added to the run-time global area, the constant's type and value are compared to the previous definition, and the load fails if a mismatch is found. This is useful as a mechanism for detecting version mismatches between modules.

CreateObject Function

See Also **Example** **Overview**

Creates a new OLE2 automation object.

Syntax **CreateObject(** *class* **)**

where:	**is:**
class	The name of the application, a period, and the name of the object to be used.

To create an object, you first must declare an object variable, using **Dim**, and then **Set** the variable equal to the new object, as follows:

Dim *OLE2* **As Object**

Set *OLE2* **= CreateObject(**"spoly.cpoly"**)**

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

OLE2.reset

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

CStrings Metacommand [VCBasic Extension]

See Also **Example**

Tells the compiler to treat a backslash character inside a string (\) as an escape character.

Syntax **'$CStrings** [**Save** | **Restore**]

where:	**is:**
Save	Saves the current $Cstrings setting.
Restore	Restores a previously saved $CStrings setting.

This treatment of a backslash in a string is based on the 'C' language.

Save and **Restore** operate as a stack and allow the user to change the setting for a range of the program without impacting the rest of the program.

The supported special characters are:

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

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

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

Octal Code	\ddd
Hexadecimal Code	\xddd

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

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

CVar Function

See Also **Example**

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

Syntax **CVar(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number. **CVar** accepts any type of *expression.*

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

CVDate Function

See Also **Example**

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

Syntax **CVDate(** *expression* **)**

where: **is:**

expression Any expression that can evaluate to a number.

CVDate accepts both string and numeric values.

The **CVDate** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999. A value of zero represents December 30, 1899. Times are represented as fractional days.

Date Statement

See Also **Example**

Sets the system date.

Syntax **Date**[**$**] **=** *expression*

where: **is:**

expression A string in one of the following forms:

mm-dd-yy

mm-dd-yyyy

mm/dd/yy

mm/dd/yyyy

where *mm* denotes a month (01-12), *dd* denotes a day (01-31), and *yy* or *yyyy* denotes 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).

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 in the International section of Windows Control Panel to recognize day, month, and year if a string contains three numbers delimited by valid date separators. In addition, **Date** recognizes month names in either full or abbreviated form.

DDEInitiate Function

See Also **Example**

Opens a dynamic-data exchange (DDE) channel and returns the DDE channel number (1,2, etc.).

Syntax **DDEInitiate(** *appname$* , *topic$* **)**

where: **is:**

appname$ A string or expression for the name of the DDE application to talk to.

topic$* A string or expression for the name of a topic recognized by *appname$.

If **DDEInitiate** is unable to open a channel, it returns zero (0).

Appname$ is usually the name of the application's .EXE file without the .EXE filename extension. If the application is not running, **DDEInitiate** cannot open a channel and returns an error. Use **Shell** to start an application.

Topic$* is usually an open filename. If *appname$ doesn't recognize *topic$*, **DDEInitiate** generates an error. Many applications that support DDE recognize a topic named **System**, which is always available and can be used to find out which other topics are available. For more information on the **System** topic, see **DDERequest**.

The maximum number of channels that can be open simultaneously is determined by the operating system and your system's memory and resources. If you aren't using an open channel, you should conserve resources by closing it using **DDETerminate**.

DDEPoke Statement

See Also **Example**

Sends data to an application on an open dynamic-data exchange (DDE) channel.

Syntax **DDEPoke** *channel%*, *item$*, *data$*

where: is:

channel% An integer or expression for the open DDE channel number.

item$ A string or expression for the name of an item in the currently opened topic.

data$ A string or expression for the information to send to the topic.

If *channel%* doesn't correspond to an open channel, an error occurs.

When you open a channel to an application using **DDEInitiate**, you also specify a topic, such as a filename, to communicate with. The *item$* is the part of the topic you want to send data to. **DDEPoke** sends data as a text string; you cannot send text in any other format, nor can you send graphics.

If the server application doesn't recognize *item$*, an error occurs.

DDETerminate Statement

See Also **Example**

Closes the specified dynamic data exchange (DDE) channel.

Syntax **DDETerminate** *channel%*

where: is:

channel% An integer or expression for the open DDE channel number.

To free system resources, you should close channels you aren't using. If *channel%* doesn't correspond to an open channel, an error occurs.

Dir Function

See Also **Example**

Returns a filename that matches the specified pattern.

Syntax **Dir**[**$**] [(*pathname$* [,*attributes%*)]

where: is:

pathname$ A string expression identifying a path or filename.

attributes% An integer expression specifying the file attributes to select.

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

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

Here are the possible values for *attributes%*:

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

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

If *attributes%* is set to 8, the **Dir** function returns the volume label of the drive specified in the *pathname$*, or of the current drive if drive is not explicitly specified. If volume label attribute is set, all other attributes are ignored.

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

Int Function

See Also **Example**

Returns the integer part of a *number*.

Syntax **Int(** *number* **)**

where:	**is:**
number	Any numeric expression.

For positive *numbers*, **Int** removes the fractional part of the expression and returns the integer part only. For negative *numbers*, **Int** returns the largest integer less than or equal to the expression. For example, **Int** (6.2) returns 6; **Int**(-6.2) returns -7.

The return type matches the type of the numeric expression. 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).

$NoCStrings Metacommand [VCBasic Extension]*

See Also **Example**

Tells the compiler to treat a backslash (\) inside a string as a normal character.

Syntax **'$NoCStrings** [**Save**]

where:	**means:**
Save	Saves the current **'$CStrings** setting before restoring the treatment of the backslash (\) to a normal character.

Use the **'$CStings Restore** command to restore a previously saved setting. Save and Restore operate as a stack and allow the user to change the **'$CStrings** setting for a range of the program without impacting the rest of the program.

Use the **'$CStrings** metacommand to tell the compiler to treat a backslash (\) inside of a string as an Escape character.

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

Now Function

See Also **Example**

Returns the current date and time.

Syntax **Now( )**

The **Now** function returns a **Variant** of vartype 7 (date) that represents the current date and time according to the setting of the computer's system date and time.

Help Typographic Conventions

VCBasic Help uses the following typographic conventions:

To represent:	**Help syntax is:**
Statements and functions	Boldface; initial letter uppercase: **Abs** **Len**(*variable*)
Arguments to statements or functions	All lowercase, italicized 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	A list inside braces, with OR operator (|) separating choices: {Goto *label* | Resume Next | Goto 0}

Other Ways to Halt Programs

For ending a program, see Unload Form Method.

If you are debugging a program, see Setting Breakpoints .

IsDate Function

See Also **Example**

Returns -1 (TRUE) if an expression is a valid date, 0 (FALSE) if it is not.

Syntax IsDate**(** *expression* **)**

Where:	**Is:**
expression	The expression to be evaluated.

IsDate returns -1 (TRUE) if the expression is of vartype 7 (date) or a string that can be interpreted as a date.

IsEmpty Function

See Also **Example**

Returns -1 (TRUE) if a Variant has been initialized. 0 (FALSE) otherwise.

Syntax IsEmpty**(** *expression* **)**

Where:	**Is:**
expression	Any expression with a data type of **Variant**.

IsEmpty returns -1 (TRUE) if the Variant is of vartype 0 (empty).

Any newly-defined Variant defaults to being of Empty type to signify that it contains no initialized data.

IsNull Function

See Also **Example**

Returns a value that signifies whether or not an expression has resulted in a null value.

Syntax **IsNull(** *expression***)**

where:	**is:**
expression	Any expression with a data type of **Variant**.

IsNull returns -1 (TRUE) if a Variant expression contains the Null value, 0 (FALSE) if it does not.

Null Variants have no associated data and serve only to represent invalid or ambiguous results.

Null is **not** the same as Empty; Empty indicates that a Variant has not yet been initialized.

IsNumeric Function

See Also **Example**

Returns -1 (TRUE) if an expression has a data type of **Numeric**, 0 (FALSE) otherwise.

Syntax **IsNumeric(** *expression* **)**

where:	**is:**
expression	Any valid expression.

IsNumeric returns -1 (TRUE) if the expression is of vartypes 2-6 (numeric) or a string that can be interpreted as a number; otherwise, it returns 0 (FALSE).

Is Operator

See Also **Example** **Overview**

Compares two object expressions and returns -1 (TRUE) if they refer to the same object, and 0 (FALSE) if they do not..

Syntax *objectExpression* **Is** *objectExpression2*

Where:	**is:**
objectexpression	Any valid object expression.
objectexpression2	Any other valid object expression.

Is can also be used to test if an object variable has been Set to Nothing.

AppClassActivate Statement

See Also **Example**

Activates an application window.

Syntax **AppClassActivate** class [, title]

where:	**is:**
class	A string expression for the class name of the application window to activate.
title	An optional string expression for the title-bar name of the application window to activate.

Class must match the class of the window character for character, but comparison is not case-sensitive, e.g., "File Manager" is the same as "file manager" or "FILE MANAGER". If no title is specified and there is more than one window with a name matching class, a window is chosen at random unless a title is specified. If there is more than one window with a name and title matching those supplied, a window is chosen at random.

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

Visual CommBasic offers a number of functions that are not included in or recognized by Microsoft's Visual Basic. These functions and statements are used specifically for terminal emulation manipulation.

At this time, OutsideView macros created prior to version 6.0 are not compatible with version 6.0 or later. All future versions of OutsideView, however, will maintain Visual CommBasic compatilibility to the extent that advances in operating systems allow.

' Abs Function Example

```
'This example finds the difference between two variables, oldacct and newacct.

Sub main

Dim oldacct, newacct, count

   oldacct=InputBox("Enter the oldacct number")

   newacct=InputBox("Enter the newacct number")

   count=Abs(oldacct-newacct)

   MsgBox "The absolute value is: " &count

End Sub
```

' AppActivate Statement Example

```
'This example opens the Windows bitmap file SETUP.BMP in Paint. (Paint must already be open
before running this example. It must also not be minimized.)

Sub main

   MsgBox "Opening C:\WINDOWS\SETUP.BMP in Paint."

   AppActivate "untitled - Paint"

   DoEvents

   SendKeys "%FOC:\WINDOWS\SETUP.BMP{Enter}",1

   MsgBox "File opened."

End Sub
```

' 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
```

' Atn Function Example

'This example finds the roof angle necessary for a house with an attic ceiling of 8 feet (at the roof peak) and a 16 foot span from the outside wall to the center of the house. The Atn function returns the angle in radians; it is multiplied by 180/PI to convert it to degrees.

```
Sub main

   Dim height, span, angle, PI

   PI=3.14159

   height=8

   span=16

   angle=Atn(height/span)*(180/PI)

   MsgBox "The angle is " & Format(angle, "##.##") & " degrees"

End Sub
```

' Beep Statement Example

'This example beeps and displays a message in a box if the variable *balance* is less than 0. (If you have a set of speakers hooked up to your computer, you might need to turn them on to hear the beep.)

```
Sub main

   Dim expenses, balance, msgtext

   balance=InputBox("Enter your account balance")

   expenses=1000

   balance=balance-expenses

   If balance<0 then

      Beep

      Msgbox "I'm sorry, your account is overdrawn."

   Else

      Msgbox "Your balance minus expenses is: " &balance

   End If
End Sub
```

' 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, etc.

Sub main
  Dim ComboBox1() as String
  Dim ListBox1() as String
  Dim DropListBox1() as String
  ReDim ListBox1(0)
  ReDim ComboBox1(0)
  ReDim DropListBox1(3)
  ListBox1(0)="C:\"
  ComboBox1(0)=Dir("C:\*.*")
  For x=0 to 2
   DropListBox1(x)=Chr(65+x) & ":"
  Next x
  Begin Dialog UserDialog 274, 171, "VCBasic 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 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, "VCBasic 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 Example

```
'This example defines a dialog box with a group of three buttons.
Sub main
  Begin Dialog UserDialog 34,0,231,140, "VCBasic 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 Example

```
'This example calls a subprogram named CREATEFILE to open a file, write the numbers 1 to 10 in it and leave it open. The calling procedure then checks the file's mode. If the mode is 1 (open for Input) or 2 (open for Output), the procedure closes the file.
Declare Sub createfile()
Sub main
  Dim filemode as Integer
  Dim attrib as Integer
  Call createfile
  attrib=1
  filemode=FileAttr(1,attrib)
  If filemode=1 or 2 then
    MsgBox "File was left open. Closing now."
    Close #1
  End If
  Kill "C:\TEMP001"
End Sub


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

' 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, "VCBasic 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 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 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 Example

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

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

' ChDir Statement Example

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

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

' ChDrive Statement Example

```
'This example changes the default drive to A:\.
Sub main
  Dim newdrive as String
  newdrive="A:"
  If Left(CurDir,2) <> newdrive then
    ChDrive newdrive
  End If
  MsgBox "The default drive is now " & newdrive
End Sub
```

' 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, "VCBasic 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 Example

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

Sub main

  Dim numb as Integer

  Dim msgtext

  Dim out

  out=0

  Do Until out

    numb=InputBox("Type a number between 65 and 122:")

    If Chr$(numb)>="A" AND Chr$(numb)<="Z" OR Chr$(numb)>="a" AND _

      Chr$(numb)<="z" then

      msgtext="The letter for the number " & numb  &" is: " & Chr$(numb)

      out=1

    ElseIf numb=0 then

      Exit Sub

    Else

      Beep

      msgtext="Does not convert to a character; try again."

    End If

    MsgBox msgtext

  Loop

End Sub
```

' CInt Function Example

```
'This example calculates the average of ten golf scores.

Sub main

  Dim score As Integer

  Dim x, sum

  Dim msgtext

  Let sum=0

  For x=1 to 10

    score=InputBox("Enter golf score #"&x &":")

    sum=sum+score

  Next x

  msgtext="Your average is: " & Format(CInt(sum/(x-1)),"General Number")

  MsgBox msgtext

End Sub
```

' Clipboard Example

```
'This example places the text string "Hello, world." on the Clipboard.

Sub main

  Dim mytext as String

  mytext="Hello, world."

  Clipboard.Settext mytext

  MsgBox "The text: '" & mytext & "' added to the Clipboard."

End Sub
```

' CLng Function Example

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

Sub main

  Dim debt As Single

  Dim msgtext

  Const Populace = 250000000

  debt=InputBox("Enter the current US national debt:")
  msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")

  MsgBox msgtext

End Sub
```

' Close Statement Example

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

```
Declare Sub createfile()

Sub main

  Dim acctno as String*3

  Dim recno as Long

  Dim msgtext as String

  Call createfile

  recno=1

  newline=Chr(10)

  Open "C:\TEMP001" For Random As #1 Len=3

  msgtext="The account numbers are:" & newline & newline

  Do Until recno=11

      Get #1,recno,acctno

      msgtext=msgtext & acctno

      recno=recno+1

  Loop

  MsgBox msgtext

  Close #1

  Kill "C:\TEMP001"

End Sub


Sub createfile()

  Rem Put the numbers 1-10 into a file

  Dim x as Integer

  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
  Close #1
End Sub
```

' 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, "VCBasic 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 Example

```
'This example opens the file entered by the user on the command line.
Sub main
  Dim filename as String
  Dim cmdline as String
  Dim cmdlength as Integer

  Dim position as Integer

  cmdline=Command

  If cmdline="" then

    MsgBox "No command line information."

    Exit Sub

  End If

  cmdlength=Len(cmdline)

  position=InStr(cmdline,Chr(32))

  filename=Mid(cmdline,position+1,cmdlength-position)

  On Error Resume Next

  Open filename for Input as #1

  If Err<>0 then

    MsgBox "Error loading file."

    Exit Sub

  End If

  MsgBox "File " & filename & " opened."

  Close #1

  MsgBox "File " & filename & " closed."

End Sub
```

' Const Statement Example

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

```
Sub main

         Dim debt As Single

         Dim msgtext

         Const Populace=250000000

         debt=InputBox("Enter the current US national debt:")

         msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")

         MsgBox msgtext
```

```
End Sub
```

' Cos Function Example

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

```
Sub main

   Dim bwidth, roof,pitch

   Dim msgtext

   Const PI=3.14159

   Const conversion=PI/180

   pitch=InputBox("Enter roof pitch in degrees")

   pitch=Cos(pitch*conversion)

   bwidth=InputBox("Enter 1/2 of house width in feet")

   roof=bwidth/pitch

   msgtext="The length of the roof is " & Format(roof, "##.##") & " feet."

   MsgBox msgtext

End Sub
```

' CreateObject Function Example

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

```
Sub main

   Dim visio as Object

   Dim doc as Object

   Dim i as Integer, doccount as Integer



'Initialize Visio

   on error resume next

   Set visio = GetObject(,"visio.application") ' find Visio

   If (visio Is Nothing) then

      Set visio = CreateObject("visio.application")        ' find Visio

      If (visio Is Nothing) then

         Msgbox "Couldn't find Visio!"
      Exit Sub

    End If

  End If

  MsgBox "Visio is open."

End Sub
```

' CSng Function Example

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

```
Sub main

  Dim number as Integer

  Dim factorial as Double

  Dim msgtext

  number=InputBox("Enter an integer between 1 and 170:")

  If number<=0 then

    Exit Sub

  End If

  factorial=1

  For x=number to 2 step -1

    factorial=factorial*x

  Next x

Rem If number =<35, then its factorial is small enough to be stored

Rem 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
```

' 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 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 Statement Example

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

```
Sub main
   Dim newdir as String
   newdir="c:\windows"
   If CurDir <> newdir then
```

```
    ChDir newdir
  End If
  MsgBox "The default directory is now: " & newdir
End Sub
```

' 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 with a type of: " & VarType(answer)
End Sub
```

' 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 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 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 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 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 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
```

' DDEAppReturnCode Function Example

(None)

' DDEExecute Statement Example

```
'This example opens Microsoft Write, uses DDEPoke to write the text "Hello, world" to the open document (Untitled) and uses DDEExecute to save the text to the file TEMP001.

Sub main

  Dim channel as Integer

  Dim appname as String

  Dim topic as String

  Dim testtext as String

  Dim item as String

  Dim pcommand as String

  Dim msgtext as String

  Dim x as Integer

  Dim path as String

  appname="WinWord"

  path="c:\msoffice\winword\"
  topic="Document1"
  item="Page1"
  testtext="Hello, world."
  On Error Goto Errhandler
  x=Shell(path & appname & ".EXE")
  channel = DDEInitiate(appname, topic)
  If channel=0 then
    MsgBox "Unable to open Write."
    Exit Sub
  End If
  DDEPoke channel, item, testtext
  pcommand="[FileSaveAs .Name = " & Chr$(34) & "C:\TEMP001" & Chr$(34) & "]"
  DDEExecute channel, pcommand
  pcommand="[FileClose]"
  DDEExecute channel, pcommand
  msgtext="The text: " & testtext & " saved to C:\TEMP001." & Chr$(13)
  msgtext=msgtext & Chr$(13) & "Delete? (Y/N)"
  answer=InputBox(msgtext)
  If answer="Y" or answer="y" then
    Kill "C:\TEMP001.doc"
  End If
  DDETerminate channel
  Exit Sub
Errhandler:
  If Err<>0 then
   MsgBox "DDE Access failed."
  End If
End Sub
```

' DDEInitiate Function Example

```
'This example uses DDEInitiate to open a channel to Microsoft Word. It uses DDERequest to obtain a list of available topics (using the System topic).

Sub main

  Dim channel as Integer

  Dim appname as String

  Dim topic as String

  Dim item as String

  Dim msgtext as String

  Dim path as string

  appname="winword"

  topic="System"

  item="Topics"

  path="c:\msoffice\winword\"

  channel = -1

  x=Shell(path & appname & ".EXE")

  channel = DDEInitiate(appname, topic)

  If channel= -1 then

    msgtext="M/S Word not found -- please place on your path."

  Else

    On Error Resume Next

    msgtext="The Word topics available are:" & Chr$(13)

    msgtext=msgtext & Chr$(13) & DDERequest(channel,item)

    DDETerminate channel

    If Err<>0 then

      msgtext="DDE Access failed."

    End If

  End If

  MsgBox msgtext

End Sub
```

' DDEPoke Statement Example

'This example opens Microsoft Write, uses DDEPoke to write the text "Hello, world" to the open document (Untitled) and uses DDEExecute to save the text to the file TEMP001.

```
Sub main

  Dim channel as Integer

  Dim appname as String

  Dim topic as String

  Dim testtext as String

  Dim item as String

  Dim pcommand as String

  Dim msgtext as String

  Dim x as Integer

  Dim path as String

  appname="WinWord"

  path="c:\msoffice\winword\"

  topic="Document1"

  item="Page1"

  testtext="Hello, world."

  On Error Goto Errhandler

  x=Shell(path & appname & ".EXE")

  channel = DDEInitiate(appname, topic)

  If channel=0 then

    MsgBox "Unable to open Write."

    Exit Sub

  End If

  DDEPoke channel, item, testtext

  pcommand="[FileSaveAs .Name = " & Chr$(34) & "C:\TEMP001" & Chr$(34) & "]"

  DDEExecute channel, pcommand

  pcommand="[FileClose]"

  DDEExecute channel, pcommand

  msgtext="The text: " & testtext & " saved to C:\TEMP001." & Chr$(13)
  msgtext=msgtext & Chr$(13) & "Delete? (Y/N)"
  answer=InputBox(msgtext)
  If answer="Y" or answer="y" then
    Kill "C:\TEMP001.doc"
  End If
  DDETerminate channel
  Exit Sub
Errhandler:
  If Err<>0 then
    MsgBox "DDE Access failed."
  End If
End Sub
```

' DDERequest Function Example

'This example uses DDEInitiate to open a channel to Microsoft Word. It uses DDERequest to obtain a list of available topics (using the System topic).

```
Sub main
  Dim channel as Integer
  Dim appname as String
  Dim topic as String
  Dim item as String
  Dim msgtext as String
  Dim path as string
  appname="winword"
  topic="System"
  item="Topics"
  path="c:\msoffice\winword\"
  channel = -1
  x=Shell(path & appname & ".EXE")
  channel = DDEInitiate(appname, topic)
  If channel= -1 then
```

```
    msgtext="M/S Word not found -- please place on your path."
  Else
    On Error Resume Next
    msgtext="The Word topics available are:" & Chr$(13)
    msgtext=msgtext & Chr$(13) & DDERequest(channel,item)
    DDETerminate channel
    If Err<>0 then
      msgtext="DDE Access failed."
    End If
  End If
  MsgBox msgtext
End Sub
```

' DDETerminate Statement Example

'This example uses DDEInitiate to open a channel to Microsoft Word. It uses DDERequest to obtain a list of available topics (using the System topic) and then terminates the channel using DDETerminate.

```
Sub main
  Dim channel as Integer
  Dim appname as String
  Dim topic as String
  Dim item as String
  Dim msgtext as String
  Dim path as string
  appname="winword"
  topic="System"
  item="Topics"
  path="c:\msoffice\winword\"
  channel = -1
  x=Shell(path & appname & ".EXE")
  channel = DDEInitiate(appname, topic)
  If channel= -1 then
```

```
    msgtext="M/S Word not found -- please place on your path."
  Else
    On Error Resume Next
    msgtext="The Word topics available are:" & Chr$(13)
    msgtext=msgtext & Chr$(13) & DDERequest(channel,item)
    DDETerminate channel
    If Err<>0 then
      msgtext="DDE Access failed."
    End If
  End If
  MsgBox msgtext
End Sub
```

' Declare Statement Example

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

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


Function VCBasic_exfunction()
  VCBasic_exfunction=1
End Function
```

' Def*type* Statement Example

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

```
DefInt c,s,t
DefSng a
```

```
Sub main
  Dim count
  Dim total
  Dim score
  Dim average
  Dim msgtext
  For count=0 to 4
    score=InputBox("Enter bowling score #" & count+1 &":")
    total=total+score
  Next count
  average=total/count
  msgtext="Your average is: " &average
  MsgBox msgtext
End Sub
```

' 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, "VCBasic 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 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, "VCBasic 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 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 Example

```
'This example lists the contents of the diskette in drive A.

Sub main

  Dim msgret

  Dim directory, count

  Dim x, msgtext

  Dim A()

  msgret=MsgBox("Insert a disk in drive A.")

  count=1

  ReDim A(100)

  directory=Dir ("A:\*.*")

  Do While directory<>""
```

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

' 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 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 grayed. Likewise, if Or is selected, Either is grayed. 'This example uses the DlgEnable statement to toggle the state of the buttons.

```
Declare Function FileDlgFunction(identifier$, action, 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
```

' 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
```

' DlgEnd Statement Example

'This example displays a dialog box with the message "You have 30 seconds to cancel." The dialog box counts down from 30 seconds to 0. If the user clicks OK or Cancel during the countdown, the dialog box closes. If the countdown reaches 0, however, the DlgEnd statement closes the dialog box.

```
Function timeout(id$,action%,suppvalue&)
  Static timeoutStart as Long
  Static currentSecs as Long
  Dim thisSecs as Long
  Select Case action%
    Case 1
      ' initialize the dialog box. Set the ticker value to 30
      ' and remember when we put up the dialog box
      DlgText "ticker", "30"
      timeoutStart = timer
      currentSecs = 30
    Case 5
      ' this is an idle message - set thisSecs to the number of
      ' seconds left until timeout
      thisSecs = timer
      If thisSecs < timeoutStart Then thisSecs = thisSecs + 24*60*60
      thisSecs = 30 - (thisSecs - timeoutStart)
      ' if there are negative seconds left, timeout!
      If thisSecs < 0 Then DlgEnd -1
      ' If the seconds left has changed since last time,
      ' update the dialog box
      If thisSecs <> currentSecs Then
```

```
         DlgText "ticker", trim$(str$(thisSecs))
         currentSecs = thisSecs
      End If
      ' make sure to return non-zero so we keep getting idle messages
      timeout = 1
   End Select
End Function


Sub main
  Begin Dialog newdlg 167, 78, "Do You Want to Continue?", .timeout
    '$CStrings Save
    OKButton  27, 49, 50, 14
    CancelButton  91, 49, 50, 14
    Text  24, 14, 119, 8, "This is your last chance to bail out."
    Text  27, 30, 35, 8, "You have"
    Text  62, 30, 13, 8, "30", .ticker
    Text  74, 30, 66, 8, "seconds to cancel."
    '$CStrings Restore
  End Dialog
  Dim dlgVar As newdlg
  If dialog(dlgvar) = 0 Then
     Exit Sub                         ' abort
  End If
  ' do whatever it is we want to do
End Sub
```

' 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 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 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 If
    End Select
End Function
```

' 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 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\CIRCLES.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\TILES.BMP",0
        Else
          DlgSetPicture 2, "C:\WINDOWS\CIRCLES.BMP",0
        End If
      End If
  End Select
End Function
```

' DlgText Function Example

'This example displays a dialog box similar to File Open. It uses DlgText to determine what group of files to display.

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


Sub main
  Dim identifier$
  Dim action as Integer
  Dim suppvalue as Integer
  Dim filetypes as String
  Dim exestr$()
  Dim button as Integer
  Dim x as Integer
  Dim directory as String
  filetypes="Program files (*.exe)"+Chr$(9)+"All Files (*.*)"
  Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
    '$CStrings Save
    Text  8, 6, 60, 11, "&Filename:"
    TextBox  8, 17, 76, 13, .TextBox1
    ListBox  9, 36, 75, 61, exestr$(), .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 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 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\CIRCLES.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\TILES.BMP",0

        Else

          DlgSetPicture 2, "C:\WINDOWS\CIRCLES.BMP",0

        End If

      End If

  End Select

End Function
```

' 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 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 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
```

' Do...Loop Statement Example

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

```
Sub main
Dim msgret
  Dim directory, count
  Dim x, msgtext
  Dim A()
  msgret=MsgBox("Insert a disk in drive A.")
  count=1
  ReDim A(100)
  directory=Dir ("A:\*.*")
  Do While directory<>""
    A(count)=directory
    count=count+1
    directory=Dir
  Loop
  msgtext="Directory of drive A:\ is:" & Chr(10)
  For x=1 to count
    msgtext=msgtext & A(x) & Chr(10)
  Next x
  MsgBox msgtext
End Sub
```

' DoEvents Statement Example

'This example activates the Windows 95 Phone Dialer 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("Dialer.exe",1)
  For i = 1 to 5
    DoEvents
  Next i
  AppActivate "Phone Dialer"
  SendKeys phonenumber & "{Enter}",1
  msgtext="Dialing..."
  MsgBox msgtext
  DoEvents
End Sub
```

' 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, "VCBasic 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 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, "VCBasic 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 Statement Example

```
'This example lists all the strings from the operating system environment table.
Sub main
  Dim str1(100)
  Dim msgtext
  Dim count, x
  Dim newline
  newline=Chr(10)
  x=1
  str1(x)= Environ(x)
  Do While Environ(x)<>""
    str1(x)= Environ(x)
    x=x+1
    str1(x)=Environ(x)
  Loop
  msgtext="The Environment Strings are:" & newline & newline
  count=x
  For x=1 to count
    msgtext=msgtext & str1(x) & newline
  Next x
  MsgBox msgtext
End Sub
```

' Eof Function Example

'This example uses the Eof function to read records from a Random file, using a Get statement. The Eof function keeps the Get statement from attempting to read beyond the end of the file. The subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

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


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

' 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 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 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 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 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
```

' 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.

```
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
```

' 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 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 Example

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

```
Declare Sub createfile()
Sub main
  Dim filemode as Integer
  Dim attrib as Integer
  Call createfile
  attrib=1
  filemode=FileAttr(1,attrib)
  If filemode=1 or 2 then
    MsgBox "File was left open. Closing now."
    Close #1
  Else
    For x=11 to 15
      Write #1, x
    Next x
    Close #1
  End If
  Kill "C:\TEMP001"
End Sub
Sub createfile()
  Rem Put the numbers 1-10 into a file
  Dim x as Integer
  Open "C:\TEMP001" for Output as #1
  For x=1 to 10
    Write #1, x
  Next x
End Sub
```

' 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 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 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 Example

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

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

' 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.

```
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 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 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 Example

```
'This example declares a function that is later called by the main subprogram. The function does
nothing but set its return value to 1.
Declare Function VCBasic_exfunction()
Sub main
  Dim y as Integer
  Call VCBasic_exfunction
  y=VCBasic_exfunction
  MsgBox "The value returned by the function is: " & y
End Sub
```

```
Function VCBasic_exfunction()
  VCBasic_exfunction=1
End Function
```

' 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 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 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
```

' 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
```

' GetObject Function Example

```
'This example displays a list of open files in the software application, VISIO. It uses the GetObject
function to access VISIO. To see how this example works, you need to start VISIO and open one or
more documents.
Sub main
  Dim visio as Object
  Dim doc as Object
  Dim msgtext as String
  Dim i as Integer, doccount as Integer

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

' 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 Example

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

```
Sub main
  Dim str1 as String
  Dim nextweek
  Dim msgtext
i: str1=InputBox$("Enter a date:")
```

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

' 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 Example

```
'This example returns the hex value for a number entered by the user.

Sub main

  Dim usernum as Integer

  Dim hexvalue

  usernum=InputBox("Enter a number to convert to hexidecimal:")

  hexvalue=Hex(usernum)

  Msgbox "The HEX value is: " & hexvalue

End Sub
```

' 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 Function Example

```
'This example checks the time and the day of the week, and returns an appropriate message.
Sub main
  Dim h, m, m2, w
  h = hour(now)
  If h > 18 then
    m= "Good evening, "
  Elseif h >12 then
    m= "Good afternoon, "
  Else
    m= "Good morning, "
  End If
    w = weekday(now)
  If w = 1 or w = 7 then m2 = "the office is closed." else m2 = "please hold for company operator."
  Msgbox m & m2
End Sub
```

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 VCBasic 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 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 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 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 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 timer() + x*255
    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 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 timer() + x*255

        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 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 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 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 Example

'This example accepts a string from the user and checks to see if it is a valid date

```
Sub main
  Dim theDate
  theDate = InputBox( "Enter a date:" )
  If IsDate(theDate)=-1 then
    MsgBox "The new date is: " & Format(CVDate(theDate), "dddddd")
  Else
    MsgBox "The date is not valid."
  End If
End Sub
```

' IsEmpty Function Example

'This example prompts for a series of test scores and uses IsEmpty to determine whether the maximum allowable limit has been hit. (IsEmpty determines when to exit the Do...Loop.)

```
Sub main
  Dim arrayvar(10)
  Dim x as Integer
  Dim tscore as Single
  Dim total as Integer
  x=1
  Do
    tscore=InputBox("Enter test score #" & x & ":")
    arrayvar(x)=tscore
    x=x+1
  Loop Until IsEmpty(arrayvar(10))<>-1
  total=x-1
  msgtext="You entered: " & Chr(10)
  For x=1 to total
     msgtext=msgtext & Chr(10) & arrayvar(x)
  Next x
  MsgBox msgtext
End Sub
```

' IsMissing Function Example

'This example prints a list of letters. The number printed is determined by the user. If the user wants to print all letters, the Function myfunc is called without any argument. The function uses IsMissing to determine whether to print all the letters or just the number specified by the user.

```
Sub myfunc(Optional arg1)
  If IsMissing(arg1)=-1 then
    arg1=26
  End If
  msgtext="The letters are: " & Chr$(10)
  For x= 1 to arg1
    msgtext=msgtext & Chr$(x+64) & Chr$(10)
  Next x
  MsgBox msgtext
End sub


Sub main
  Dim arg1
  arg1=InputBox("How many letters do you want to print? (0 for all)")
  If arg1=0 then
    myfunc
  Else
    myfunc arg1
  End If
End Sub
```

' 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 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 Function Example

'This example prompts a user for an account number, opens a file, searches for the account number and displays the matching letter for that number. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram. After processing is complete, the first subroutine uses Kill to delete the file.

```
Declare Sub createfile()
Global x as Integer
Global y(100) as String


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

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

```
End Sub
```

' LBound Function Example

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1

Sub main

  Dim arrayvar() as Integer

  Dim count as Integer

  Dim answer as String

  Dim x, y as Integer

  Dim total

  total=0

  x=1

  count=InputBox("How many test scores do you have?")

  ReDim arrayvar(count)

start:

  Do until x=count+1

    arrayvar(x)=InputBox("Enter test score #" &x & ":")

    x=x+1

  Loop

  answer=InputBox$("Do you have more scores? (Y/N)")

  If answer="Y" or answer="y" then

    count=InputBox("How many more do you have?")

    If count<>0 then

      count=count+(x-1)

      ReDim Preserve arrayvar(count)

      Goto start

    End If

  End If

  x=LBound(arrayvar,1)
  count=UBound(arrayvar,1)

  For y=x to count

      total=total+arrayvar(y)

  Next y

  MsgBox "The average of " & count & " scores is: " & Int(total/count)

End Sub
```

' LCase Function Example

```
'This example converts a string entered by the user to lowercase.

Sub main

Dim userstr as String

  userstr=InputBox$("Enter a string in upper and lowercase letters")

  userstr=LCase$(userstr)

  Msgbox "The string now is: " & userstr

End Sub
```

' Left Function Example

```
'This example extracts a user's first name from the entire name entered.

Sub main

          Dim username as String

          Dim count as Integer

          Dim firstname as String

          Dim charspace

          charspace=Chr(32)

          username=InputBox("Enter your first and last name")

          count=InStr(username,charspace)

          firstname=Left(username,count)

          Msgbox "Your first name is: " &firstname

End Sub
```

' Len Function Example

```
'This example returns the length of a name entered by the user (including spaces).
```

```
Sub main
        Dim username as String
        username=InputBox("Enter your name")
        count=Len(username)
        Msgbox "The length of your name is: " &count
End Sub
```

' Let (Assignment Statement) Example

'This example uses the Let statement for the variable sum. The subroutine finds an average of 10 golf scores.

```
Sub main
        Dim score As Integer
        Dim x, sum
        Dim msgtext
        Let sum=0
        For x=1 to 10
                score=InputBox("Enter your last ten golf scores #" & x & ":")
                sum=sum+score
        Next x
        msgtext="Your average is: " & CInt(sum/(x-1))
        MsgBox msgtext
End Sub
```

' 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 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 contents of c:\temp001 is: " & newline

  Do Until x=Lof(1)

    Line Input #1, testscore

    x=x+1

    y=Seek(1)

    If y>Lof(1) then

      x=Lof(1)

    Else

      Seek 1,y
    End If
    msgtext=msgtext & testscore & newline
  Loop
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub

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

' 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, "VCBasic 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 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 Function 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 Example

```
'This example opens a file and prints its contents to the screen.
Sub main
  Dim fname
  Dim fchar()
  Dim x as Integer
  Dim msgtext
  Dim newline
  newline=Chr(10)
  fname=InputBox("Enter a filename to print:")
  On Error Resume Next
  Open fname for Input as #1
  If Err<>0 then
    MsgBox "Error loading file. Re-run program."
    Exit Sub
  End If
  msgtext="The contents of " & fname & " is: " & newline &newline
  Redim fchar(Lof(1))

   For x=1 to Lof(1)

      fchar(x)=Input(1,#1)

      msgtext=msgtext & fchar(x)

  Next x

  MsgBox msgtext

  Close #1

End Sub
```

' 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 cannot 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 Example

'This example puts a user's last name into the variable LASTNAME. If the name is longer than the size of LASTNAME, then the user's name is truncated. If you have a long last name and you get lots of junk mail, you've probably seen how this works already.

```
Sub main

        Dim lastname as String

        Dim strlast as String*8
        lastname=InputBox("Enter your last name")

        Lset strlast=lastname

        msgtext="Your last name is: " &strlast

        MsgBox msgtext

End Sub
```

' LTrim Function Example

'This example trims the leading spaces from a string padded with spaces on the left.

```
Sub main

  Dim userinput as String

  Dim numsize

  Dim str1 as String*50

  Dim strsize

  strsize=50

  userinput=InputBox("Enter a string of characters:")

  numsize=Len(userinput)

  str1=Space(strsize-numsize) & userinput

' Str1 has a variable number of leading spaces.

  MsgBox "The string is: " &str1

  str1=LTrim$(str1)

' Str1 now has no leading spaces.

  MsgBox "The string now has no leading spaces: " & str1

End Sub
```

' Mid Statement Example

'This example uses the Mid statement to replace the last name in a user-entered string to asterisks (*).

```
Sub main

  Dim username as String

  Dim position as Integer

  Dim count as Integer

  Dim uname as String
  Dim replacement as String
  username=InputBox("Enter your full name:")
  uname=username
  replacement="*"
  Do
    position=InStr(username," ")
    If position=0 then
      Exit Do
    End If
    username=Mid(username,position+1)
    count=count+position
  Loop
  For x=1 to Len(username)
    count=count+1
    Mid(uname,count)=replacement
  Next x
  MsgBox "Your name now is: " & uname
End Sub
```

' Mid Function Example

```
'This example uses the Mid function to find the last name in a string entered by the user.
Sub main
  Dim username as String
  Dim position as Integer
  username=InputBox("Enter your full name:")
  Do
   position=InStr(username," ")
   If position=0 then
    Exit Do
   End If
   position=position+1
```

```
    username=Mid(username,position)
  Loop
  MsgBox "Your last name is: " & username
End Sub
```

' 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 Example

```
'This example makes a new temporary directory in C:\ and then deletes it.

Sub main

    Dim path as String

    On Error Resume Next

    path=CurDir(C)

    If path<>"C:\" then

        ChDir "C:\"

    End If

    MkDir "C:\TEMP01"

    If Err=75 then

        MsgBox "Directory already exists"

    Else

        MsgBox "Directory C:\TEMP01 created"

        MsgBox "Now removing directory"

        RmDir "C:\TEMP01"

    End If

End Sub
```

' 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 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 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 Example

'This example creates a temporary file, C:\TEMP001, renames the file to C:\TEMP002, then deletes them both. It calls the subprogram, CREATEFILE, to create the C:\TEMP001 file.

```
Declare Sub createfile()

Sub main
```

```
    Call createfile

    On Error Resume Next

    Name "C:\TEMP001" As "C:\TEMP002"

    MsgBox "The file has been renamed"

    MsgBox "Now deleting both files"

    Kill "TEMP001"

    Kill "TEMP002"

End Sub


Sub createfile()

    Rem Put the numbers 1-10 into a file

    Dim x as Integer

    Dim y()

    Dim startletter

    Open "C:\TEMP001" for Output as #1

    For x=1 to 10

        Write #1, x

    Next x

    Close #1

End Sub
```

' New Operator Example

(None)

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 Function 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 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 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 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 Class 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 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 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, "VCBasic 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 ..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
```

' On Error Statement Example

'This example prompts the user for a drive and directory name and uses On Error to trap invalid entries.

```
Sub main

    Dim userdrive, userdir, msgtext
```

```
in1:  userdrive=InputBox("Enter drive:",,"C:")
    On Error Resume Next
    ChDrive userdrive
    If Err=68 then
      MsgBox "Invalid Drive. Try again."
      Goto in1
    End If
in2:  On Error Goto Errhdlr1
    userdir=InputBox("Enter directory path:")
    ChDir userdrive & userdir
    Msgbox "New default directory is: " & userdrive & userdir
    Exit Sub
Errhdlr1:
    Select Case Err
      Case 75
        msgtext="Path is invalid."
      Case 76
        msgtext="Path not found."
      Case 70
        msgtext="Permission denied."
      Case Else
        msgtext="Error " & Err & ": " & Error$ & "occurred."
    End Select
    MsgBox msgtext & " Try again."
    Resume in2
End Sub
```

' Open Statement Example

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

```
Declare Sub createfile()
```

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

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

' 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, "VCBasic 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
```

' OptionGroup 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 192, 71, "VCBasic Dialog Box"

    GroupBox  7, 6, 97, 57, "File Range"

    OptionGroup .OptionGroup2

      OptionButton  18, 14, 46, 12, "All pages", .OptionButton3
      OptionButton  18, 30, 67, 8, "Range of pages", .OptionButton4
    Text  24, 41, 20, 10, "From:", .Text6
    Text  62, 41, 14, 9, "To:", .Text7
    TextBox  78, 41, 13, 12, .TextBox4
    TextBox  46, 41, 12, 11, .TextBox5
    OKButton  126, 6, 54, 14
    CancelButton  126, 26, 54, 14
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

' Option Base Statement Example

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
  Dim arrayvar() as Integer
  Dim count as Integer
  Dim answer as String
  Dim x, y as Integer
  Dim total
  total=0
  x=1
  count=InputBox("How many test scores do you have?")
  ReDim arrayvar(count)
start:
  Do until x=count+1
    arrayvar(x)=InputBox("Enter test score #" &x & ":")
    x=x+1
  Loop
  answer=InputBox$("Do you have more scores? (Y/N)")
  If answer="Y" or answer="y" then
    count=InputBox("How many more do you have?")
    If count<>0 then
      count=count+(x-1)
      ReDim Preserve arrayvar(count)
      Goto start
    End If
  End If
  x=LBound(arrayvar,1)
  count=UBound(arrayvar,1)
  For y=x to count
       total=total+arrayvar(y)
  Next y
  MsgBox "The average of " & count & " scores is: " & Int(total/count)
End Sub
```

' Option Compare Statement Example

'This example compares two strings: "JANE SMITH" and "jane smith". When Option Compare is Text, the strings are considered the same. If Option Compare is Binary, they will not be the same. Binary is the default. To see the difference, run the example once, then run it again, commenting out the Option Compare statement.

```
Option Compare Text
Sub main
  Dim strg1 as String
  Dim strg2 as String
  Dim retvalue as Integer
  strg1="JANE SMITH"
  strg2="jane 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 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
```

' 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 Example

```
'This example defines a dialog box with a picture, an OK button, and a Cancel button.
Sub main
  Begin Dialog UserDialog 148, 73, "VCBasic Dialog Box"
    Picture  8, 7, 46, 46, "C:\WINDOWS\CIRCLES.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 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 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 Example

```
'This example prints the octal values for the numbers from 1 to 25.

Sub main

        Dim x as Integer

        Dim y

        For x=1 to 25

                y=Oct$(x)

                Print x Tab(10) y

        Next x

End Sub
```

' PushButton 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, "VCBasic 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
```

' 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 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 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 timer() + x*255
    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 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 Example

'This example finds the net present value for a series of cash flows. The array variable that holds the cash flow amounts is initially a dynamic array that is redimensioned after the user enters the number of cash flow periods they have.

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

' Rem Statement Example

'This example defines a dialog box with a combination list box and two buttons. The Rem statements describe each block of definition code.

```
Sub main
  Dim fchoices as String
  fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
  Begin Dialog UserDialog 185, 94, "VCBasic Dialog Box"
Rem The next two lines create the combo box
    Text  9, 5, 69, 10, "Filename:", .Text1
    DropComboBox  9, 17, 88, 71, fchoices, .ComboBox1
Rem The next two lines create the command buttons
    OKButton  113, 14, 54, 13
    CancelButton  113, 33, 54, 13
  End Dialog
  Dim mydialog as UserDialog
  On Error Resume Next
  Dialog mydialog
  If Err=102 then
    MsgBox "Dialog box canceled."
  End If
End Sub
```

' Reset Statement 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 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 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
    Shell "PBrush"
    For I = 1 to 10
      DoEvents
    Next i
    AppActivate "untitled - Paint"
    DoEvents
    Sendkeys "%FO" & filename & "{Enter}", 1
  Else
    MsgBox "File not found or extension not .BMP."
  End If
End Sub
```

' RmDir Statement Example

```
'This example makes a new temporary directory in C:\ and then deletes it.
Sub main
  Dim path as String
  On Error Resume Next
  path=CurDir(C)
  If path<>"C:\" then
    ChDir "C:\"
  End If
  MkDir "C:\TEMP01"
  If Err=75 then
    MsgBox "Directory already exists"
  Else
    MsgBox "Directory C:\TEMP01 created"
    MsgBox "Now removing directory"
    RmDir "C:\TEMP01"
  End If
End Sub
```

' Rnd Function Example

'This example generates a random string of characters within a range. The Rnd function is used to set the range between lowercase "a" and "z". The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub main
  Dim x as Integer
  Dim y
  Dim str1 as String
  Dim str2 as String
  Dim letter as String
  Dim randomvalue
  Dim upper, lower
  Dim msgtext
  upper=Asc("z")
  lower=Asc("a")
  newline=Chr(10)
  For x=1 to 26
    Randomize timer() + x*255
    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
```

' 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 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 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 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
```

' 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 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
```

' SendKeys Statement Example

'This example activates the Windows 95 Phone Dialer 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("Dialer.exe",1)

  For i = 1 to 5

    DoEvents

  Next i

  AppActivate "Phone Dialer"

  SendKeys phonenumber & "{Enter}",1

  msgtext="Dialing..."

  MsgBox msgtext

  DoEvents

End Sub
```

' Set Statement Example

'This example displays a list of open files in the software application, VISIO. It uses the Set statement to assign VISIO and its document files to object variables. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main

  Dim visio as Object

  Dim doc as Object

  Dim msgtext as String

  Dim i as Integer, doccount as Integer



'Initialize Visio
```

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

' 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 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 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 Example

'This example activates the Windows 95 Phone Dialer 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("Dialer.exe",1)
  For i = 1 to 5
    DoEvents
  Next i
  AppActivate "Phone Dialer"
  SendKeys phonenumber & "{Enter}",1
  msgtext="Dialing..."
  MsgBox msgtext
  DoEvents
End Sub
```

' Sin Function Example

```
'This example finds the height of the building, given the length of a roof and the roof pitch.
Sub main
  Dim height, rooflength
  Dim pitch
  Dim msgtext
  Const PI=3.14159
  Const conversion= PI/180
  pitch=InputBox("Enter the roof pitch in degrees:")
  pitch=pitch*conversion
  rooflength=InputBox("Enter the length of the roof in feet:")
  height=Sin(pitch)*rooflength
  msgtext="The height of the building is "
  msgtext=msgtext & Format(height, "##.##") & " feet."
  MsgBox msgtext
End Sub
```

' Space Function Example

```
'This example prints the octal numbers from 1 to 15 as a two-column list and uses Space to separate the columns.

Sub main

  Dim x,y

  Dim msgtext

  Dim 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
```

' 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
```

' SQLClose Function Example

```
'This example opens the data source named "VCBasicTest," gets the names in the ODBC data sources, and closes the connection.
Sub main
'   Declarations
'
    Dim outputStr As String
    Dim connection As Long
    Dim prompt As Integer
    Dim datasources(1 To 50) As Variant
    Dim retcode As Variant

    prompt = 5
'   Open the datasource "VCBasicTest"
    connection = SQLOpen("DSN=VCBasicTest", outputStr, prompt:=5)

    action1 = 1  ' Get the names of the ODBC datasources
    retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier, ref:=datasources())

'   Close the datasource connection
    retcode = SQLClose(connection)

End Sub
```

' SQLError Function Example

```
'This example forces an error to test SQLError function.
sub main
```

```
'  Declarations
   Dim connection As long
   Dim prompt as integer
   Dim retcode as long
   Dim errors(1 To 3, 1 To 10) as Variant

'  Open the datasource
   connection = SQLOpen("DSN=VCBasicTESTW;UID=DBA;PWD=SQL",outputStr,prompt:=3)

'  force an error to test SQLError select a nonexistent table
   retcode = SQLExecQuery(connection:=connection,query:="select * from notable ")

'  Retrieve the detailed error message information into the errors array
   SQLError destination:=errors
   retcode = SQLClose(connection)
end sub
```

' SQLExecQuery Function Example

```
'This example performs a query on the data source.
Sub main
'   Declarations
'
    Dim connection As Long
    Dim destination(1 To 50, 1 To 125)  As Variant
    Dim retcode As long

'   open the connection
    connection = SQLOpen("DSN=VCBasicTest",outputStr,prompt:=3)
'
'   Execute the query
    query = "select * from customer"
    retcode = SQLExecQuery(connection,query)
'
'   retrieve the first 50 rows with the first 6 columns of each row into
'   the array destination, omit row numbers and put column names in the
'   first row of the array
'
    retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6,fetchFirst:=0)


'   Get the next 50 rows of from the result set
    retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6)


'   Close the connection
    retcode = SQLClose(connection)


End Sub
```

' SQLGetSchema Function Example

'This example opens the data source named "VCBasicTest," gets the names in the ODBC data sources, and closes the connection.

```
Sub main
'   Declarations
'
    Dim outputStr As String
    Dim connection As Long
    Dim prompt As Integer
    Dim datasources(1 To 50) As Variant
    Dim retcode As Variant


    prompt = 5
'   Open the datasource "VCBasicTest"
```

```
    connection = SQLOpen("DSN=VCBasicTest", outputStr, prompt:=5)

    action1 = 1  ' Get the names of the ODBC datasources
    retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier, ref:=datasources())

'   Close the datasource connection
    retcode = SQLClose(connection)

End Sub
```

' SQLOpen Function Example

'This example opens the data source named "VCBasicTest," gets the names in the ODBC data sources, and closes the connection.

```
Sub main
'   Declarations
'

    Dim outputStr As String
    Dim connection As Long
    Dim prompt As Integer
    Dim datasources(1 To 50) As Variant
    Dim retcode As Variant

    prompt = 5
'   Open the datasource "VCBasicTest"
    connection = SQLOpen("DSN=VCBasicTest", outputStr, prompt:=5)

    action1 = 1  ' Get the names of the ODBC datasources
    retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier, ref:=datasources())

'   Close the datasource connection
```

```
    retcode = SQLClose(connection)


End Sub
```

' SQLRequest Function Example

'This example will open the datasource VCBasicTESTW and execute the query specified by query and return the results in destination

```
Sub main

'  Declarations

'

   Dim destination(1 To 50, 1 To 125)  As Variant

   Dim prompt As integer


'  The following will open the datasource VCBasicTESTW and execute the query

'  specified by query and return the results in destination

'

   query = "select * from class"

   retcode = SQLRequest("DSN=VCBasicTESTW;UID=DBA;PWD=SQL",query,outputStr,prompt,0,destination())


End Sub
```

' SQLRetrieve Function Example

'This example retrieves information from a data source.

```
Sub main

'   Declarations

'

    Dim connection As Long

    Dim destination(1 To 50, 1 To 125)  As Variant

    Dim retcode As long


'   open the connection
```

```
    connection = SQLOpen("DSN=VCBasicTest",outputStr,prompt:=3)
 '
 '  Execute the query
    query = "select * from customer"
    retcode = SQLExecQuery(connection,query)

 '  retrieve the first 50 rows with the first 6 columns of each row into
 '  the array destination, omit row numbers and put column names in the
 '  first row of the array

    retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6,fetchFirst:=0)

'   Get the next 50 rows of from the result set
    retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6)

'   Close the connection
    retcode = SQLClose(connection)
End Sub
```

' SQLRetrieveToFile Function Example

```
'This example opens a connection to a data source and retrieves information to a file.
Sub main
'   Declarations
'
    Dim connection As Long
    Dim destination(1 To 50, 1 To 125)  As Variant
    Dim retcode As long

'   open the connection
```

```
   connection = SQLOpen("DSN=VCBasicTest",outputStr,prompt:=3)
 '
 '  Execute the query
 '
   query = "select * from customer"
   retcode = SQLExecQuery(connection,query)


 '  Place the results of the previous query in the file named by
 '  filename and put the column names in the file as the first row.
 '  The field delimiter is %
 '
   filename = "c:\myfile.txt"
   columnDelimiter = "%"
   retcode = SQLRetrieveToFile(connection:=connection,destination:=filename,
columnNames:=1,columnDelimiter:=columnDelimiter)


   retcode = SQLClose(connection)


End Sub
```

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

' Static Statement Example

```
'This example puts account numbers to a file using the record variable GRECORD and then prints them again.
Type acctrecord
  acctno as Integer
End Type

Sub main
  Static grecord as acctrecord
  Dim x
  Dim total
  x=1
  grecord.acctno=1
  On Error Resume Next
  Open "C:\TEMP001" For Output as #1
  Do While grecord.acctno<>0
i:    grecord.acctno=InputBox("Enter 0 or new account #" & x & ":")
    If Err<>0 then
      MsgBox "Error occurred.  Try again."
      Err=0
      Goto i
    End If
    If grecord.acctno<>0 then
      Print #1, grecord.acctno
      x=x+1
    End If
  Loop
  Close #1
  total=x-1
  msgtext="The account numbers are: " & Chr(10)
  Open "C:\TEMP001" For Input as #1
  For x=1 to total
    Input #1, grecord.acctno
    msgtext=msgtext & Chr(10) & grecord.acctno
  Next x
  MsgBox msgtext
  Close #1
  Kill "C:\TEMP001"
End Sub
```

' 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, "VCBasic 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 Example

'This example stops program execution at the user's request.

```
Sub main
   Dim str1
   str1=InputBox("Stop program execution? (Y/N):")
   If str1="Y" or str1="y" then
      Stop
   End If
   MsgBox "Program complete."
End Sub
```

' Str Function Example

```
'This example prompts for two numbers, adds them, then shows them as a concatenated string.
Sub main
   Dim x as Integer
   Dim y as Integer
   Dim str1 as String
   Dim value1 as Integer
   x=InputBox("Enter a value for x: ")
   y=InputBox("Enter a value for y: ")
   MsgBox "The sum of these numbers is: " & x+y
   str1=Str(x) & Str(y)
   MsgBox "The concatenated string for these numbers is: " & str1
End Sub
```

' 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 Example

```
'This example places asterisks (*) in front of a string that is printed as a payment amount.
Sub main
  Dim str1 as String
  Dim size as Integer
i: str1=InputBox("Enter an amount up to 999,999.99: ")
  If Instr(str1,".")=0 then
    str1=str1+".00"
  End If
  If Len(str1)>10 then
    MsgBox "Amount too large.  Try again."
    Goto i
  End If
  size=10-Len(str1)
'Print amount in a space on a check allotted for 10 characters
  str1=String(size,Asc("*")) & str1
  Msgbox "The amount is: $" & str1
End Sub
```

' Sub...End Sub Function Example

```
'This example is a subroutine that uses the Sub...End Sub function.
Sub main
          MsgBox "Hello, World."
End Sub
```

' Tab Function Statement Example

'This example prints the octal values for the numbers from 1 to 25. It uses Tab to put five character spaces between the values.

```
Sub main
    Dim x as Integer
    Dim y
    For x=1 to 25
        y=Oct$(x)
        Print x Tab(10) y
    Next x
End Sub
```

' Tan Function Example

'This example finds the height of the exterior wall of a building, given its roof pitch and the length of the building.

```
Sub main
  Dim bldglen, wallht
  Dim pitch
  Dim msgtext
  Const PI=3.14159
  Const conversion= PI/180
  On Error Resume Next
  pitch=InputBox("Enter the roof pitch in degrees:")
  pitch=pitch*conversion
  bldglen=InputBox("Enter the length of the building in feet:")
  wallht=Tan(pitch)*(bldglen/2)
  msgtext="The height of the building is: " & Format(wallht, "##.00")
  MsgBox msgtext
End Sub
```

' 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, "VCBasic 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 Example

```
'This example creates a dialog box with a group box, and two buttons.
Sub main
  Begin Dialog UserDialog 194, 76, "VCBasic 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 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
```

' Time Statement Example

```
'This example changes the time on the system clock.
Sub main
  Dim newtime as String
  Dim answer as String
  On Error Resume Next
i: newtime=InputBox("What time is it?")
  answer=InputBox("Is this AM or PM?")
  If answer="PM" or answer="pm" then
```

```
    newtime=newtime &"PM"
  End If
  Time=newtime
  If Err<>0 then
    MsgBox "Invalid time.  Try again."
    Err=0
    Goto i
  End If
End Sub
```

' Timer Function Example

```
'This example uses Timer Function to find a Megabucks number.
Sub main
  Dim msgtext
  Dim value(9)
  Dim nextvalue
  Dim x
  Dim y
  msgtext="Your Megabucks numbers are: "
  For x=1 to 8
    Do
      value(x)=Timer
      value(x)=value(x)*100
      value(x)=Str(value(x))
      value(x)=Val(Right(value(x),2))
    Loop Until value(x)>1 and value(x)<36
    For y=1 to 1500
    Next y
  Next x
  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 Example

'This example displays the current time using Time Serial.

```
Sub main
  Dim y
  Dim msgtext
  Dim nowhr
  Dim nowmin
  Dim nowsec
  nowhr=Hour(Now)
  nowmin=Minute(Now)
  nowsec=Second(Now)
  y=TimeSerial(nowhr,nowmin,nowsec)
  msgtext="The time is: " & y
  MsgBox msgtext
End Sub
```

' 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 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 it."
End Sub
```

' 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 &" correct? (Y/N)")
  If answer="Y" or answer="y" then
    MsgBox "Thank you."
  Else
    MsgBox "Try again."
    Goto i
  End If
End Sub
```

' Typeof Statement Example

(None)

' UBound Function Example

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
  Dim arrayvar() as Integer
  Dim count as Integer
  Dim answer as String
  Dim x, y as Integer
  Dim total
  total=0
  x=1
  count=InputBox("How many test scores do you have?")
  ReDim arrayvar(count)
start:
  Do until x=count+1
```

```
    arrayvar(x)=InputBox("Enter test score #" &x & ":")
    x=x+1
  Loop
  answer=InputBox$("Do you have more scores? (Y/N)")
  If answer="Y" or answer="y" then
    count=InputBox("How many more do you have?")
    If count<>0 then
      count=count+(x-1)
      ReDim Preserve arrayvar(count)
      Goto start
    End If
  End If
  x=LBound(arrayvar,1)
  count=UBound(arrayvar,1)
  For y=x to count
       total=total+arrayvar(y)
  Next y
  MsgBox "The average of the " & count & " scores is: " & Int(total/count)
End Sub
```

' UCase Function Example

```
'This example converts a filename entered by a user to all uppercase letters.
Option Base 1
Sub main
  Dim filename as String
  filename=InputBox("Enter a filename: ")
  filename=UCase(filename)
  MsgBox "The filename in uppercase is: " & filename
End Sub
```

' Unlock Function 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
```

' Val 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
```

' 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 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 Structure Example

This example opens a series of customer files and checks for the string "*Overdue*" in each file. It uses While...Wend to loop through the C:\TEMP00? files. These files are created by the subroutine CREATEFILES.

```
Declare Sub createfiles

Sub main

  Dim custfile as String

  Dim aline as String

  Dim pattern as String

  Dim count as Integer

  Call createfiles

  Chdir "C:\"

  custfile=Dir$("TEMP00?")

  pattern="*" + "Overdue" + "*"

  While custfile <> ""

    Open custfile for input as #1

    On Error goto atEOF

    Do

      Line Input #1, aline

      If aline Like pattern Then

        count=count+1

      End If

    Loop

nxtfile:

    On Error GoTo 0

    Close #1

    custfile = Dir$

  Wend

  If count<>0 then

    Msgbox "Number of overdue accounts: " & count
```

```
   Else
      Msgbox "No accounts overdue"
   End If
   Kill "C:\TEMP001"
   Kill "C:\TEMP002"
   Exit Sub
atEOF:
   Resume nxtfile
End Sub

Sub createfiles()
  Dim odue as String
  Dim ontime as String
  Dim x
  Open "C:\TEMP001" for OUTPUT as #1
  odue="*" + "Overdue" + "*"
  ontime="*" + "On-Time" + "*"
  For x=1 to 3
     Write #1, odue
  Next x
  For x=4 to 6
     Write #1, ontime
  Next x
  Close #1
  Open "C:\TEMP002" for Output as #1
  Write #1, odue
  Close #1
End Sub
```

' 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 Example

This example creates a user-defined record type, custrecord and uses the With statement to fill in values for the record fields, for the record called "John".

```
Type custrecord
   name as String
   ss as String
   salary as Single
   dob as Variant
   street as String
   apt as Variant
   city as String
   state as String
End Type
Sub main
```

```
  Dim John as custrecord

  Dim msgtext

  John.name="John"

  With John

    .ss="037-67-2947"

    .salary=60000

    .dob=#10-09-65#

    .street="15 Chester St."

    .apt=28

    .city="Cambridge"

    .state="MA"

  End With

  msgtext=Chr(10) & "Name:" & Space(5) & John.name & Chr(10)

  msgtext=msgtext & "SS#: " & Space(6) & john.ss & chr(10)

  msgtext=msgtext & "D.O.B:" & Space(4) & john.dob

  Msgbox "Done with: " & Chr(10) & msgtext

End Sub
```

Write Statement Example

This example writes a variable to a disk file based on a comparison of its last saved time and the current time.

```
Sub main

  Dim tempfile

  Dim 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
      If x=1 then Exit Sub
      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-1
         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
```

' Year Function Example

```
'This example returns the year for today.

Sub main

    Dim nowyear

    nowyear=Year(Now)

    MsgBox "The current year is: " &nowyear

End Sub
```

CrtAttr Example

This example tests if the protected field attribute of row 15, column 20 is set

```
Sub main

Dim Row as Integer

Dim Col as Integer

Row = 15

Col = 20

'Test if protected field

If CRTAttr(Row, Col) And 64 Then

    MsgBox("Cursor position 15,20 is a protected data field.")

Else

    MsgBox("Cursor position 15,20 is an unprotected data field.")

End If

End Sub
```

CrtCopy Example

This procedure copies screen data to the printer and clipboard

```
Sub Main

Dim intStart as Integer

Dim intEnd as Integer

Dim intRet as Integer
```

```
'Copy the screen contents to a diskfile
intStart = 0
intEnd = CInt(CrtQuery("HEIGHT")) * CInt(CrtQuery("WIDTH")) - 1
intRet = CrtCopy(intStart, intEnd, 0, "screen.txt")
'Copy the screen contents to the clipboard
intRet = CrtCopy(intStart, intEnd, 0, "C")
'Copy the screen contents to the printer
intRet = CrtCopy(intStart, intEnd, 0, "P")
'Copy a column of information to the clip board
intStart = CrtPosition(0, 0)
intEnd = CrtPosition(20, 8)
intRet = CrtCopy(intStart, intEnd, 1, "C")
End Sub
```

CrtEmit Example

Print the current time in the upper right corner of the screen

```
Sub Main
Dim LineNum as Integer
Dim ColNum as Integer
Dim intRet as Integer
Dim WritePos as Integer
LineNum = CrtRow(-1)
ColNum = CrtCol(-1)
WritePos = CInt(CRTQUERY("WIDTH")) - 9
reslt% = CrtSetCursor(0, WritePos)
CrtEmit Time$
reslt% = CrtSetCursor(LineNum, ColNum)
End Sub
```

CrtFieldSearch_Example

This procedure will find all unprotected fields on the display and copy the data to Notepad.

```
Sub Main
```

```
Dim FieldType as Integer
Dim DataType as Integer
Dim SearchField as Integer
Dim StartPos as Integer
Dim EndPos as Integer
Dim Message as String
' Find all defined fields on the screen
FieldType = 1
DataType = 0
Do while CrtFieldSearch(SearchField, 0, FieldType, DataType) <> -1
        StartPos = CrtFieldSearch(SearchField, 0, FieldType, DataType)
        EndPos = CrtFieldSearch(SearchField, -1, FieldType, DataType)
        Message = Message & "Field " & CStr(SearchField) & " Start: "
Message = Message & cStr(StartPos) & " End: " & CStr(EndPos)
Message = Message & Chr$(13) & Chr$(10)
        SearchField = SearchField + 1
Loop
if Message = "" Then
        MsgBox "No fields of specified type found", 64
Else
        'Copy to clipboard
        clipboard.SetText(Message)
        'Start notepad
        shell ("Notepad.exe")
        'Paste into notepad (by sending Ctrl-v)
        sendKeys "^v",True
end if
End Sub
```

CrtQuery Example

This procedure copies screen data to the printer and clipboard

```
Sub Main

Dim intStart as Integer

Dim intEnd as Integer

Dim intRet as Integer

'Copy the screen contents to a diskfile

intStart = 0

intEnd = CInt(CrtQuery("HEIGHT")) * CInt(CrtQuery("WIDTH")) - 1

intRet = CrtCopy(intStart, intEnd, 0, "screen.txt")

'Copy the screen contents to the clipboard

intRet = CrtCopy(intStart, intEnd, 0, "C")

'Copy the screen contents to the printer

intRet = CrtCopy(intStart, intEnd, 0, "P")

'Copy a column of information to the clip board

intStart = CrtPosition(0, 0)

intEnd = CrtPosition(20, 8)

intRet = CrtCopy(intStart, intEnd, 1, "C")

End Sub
```

CrtRow Example

This procedure will wait for a TACL prompt then Emit the username and password.

```
Sub main

Dim pass as String

Dim LoginID as String

Dim NowTime as Long

Dim intRet as Integer

LoginID = "machine.user"

Pass = "NewPass"

'In case macro is the startup macro For the session, wait For TACL

'prompt

NowTime = Timer

Do While CRTGet$(CRTRow(-1), CRTCol(-1) - 2, 1) <> ">"
```

```
    'Allow 10s For session to start
    if Timer > NowTime + 10 then
        msgBox "Did not detect TACL Prompt", 48
        exit sub
    end if
    DoEvents
Loop
Emit "Logon " & LoginID
intRet = WaitStr(5,"Password:")
Emit Pass
End Sub
```

CrtSearch Example

This procedure will find the beginning of the "Password" entry field.

```
Sub main
Dim intRow as Integer
Dim intCol as Integer
Dim strText as String
Dim intPos as Integer
Dim RowNum as Integer
Dim ColNum as Integer
Dim Reply as String
' Find password field
intRow = 0
intCol = 0
strText = "PASSWORD:"
intPos = CrtSearch(intRow, intCol, strText, "N")
If intPos <>  -1 Then
    intPos = intPos + Len(strText) + 3 'Move to first field position
    Colnum = CrtCol(intPos)
    Rownum = CrtRow(intPos)
```

```
        Reply = "Password field is located at position " & Str$(Rownum)
        Reply = Reply & " Column " & Str$(Colnum)
        MsgBox Reply
Else
        MsgBox "Unable to locate the password field", 64
End If
End Sub
```

CrtSetCursor Example

This procedure will set the cursor to the second unprotected field.

```
Sub main
Dim intRow as Integer
Dim intCol as Integer
Dim intRet as Integer
Dim intPos as Integer
Dim FieldType as Integer
FieldType = 1

' Find second unprotected field
intPos = CrtFieldSearch(1, 0, FieldType)
intRow = CrtRow(intPos)
intCol = CrtCol(intPos)
intRet = CrtSetCursor(intRow, intCol)
End Sub
```

CrtTrigger Example

This procedure will trigger a special key in the emulation

```
Sub Main
Dim strRslt as String
'Secondary argument For "FuncKey" must be a match (case insensitive) to
'the function name as shown in the Mapped Keys dialog of the Key
'Mapper.
```

```
strRslt = CrtTrigger$("FUNCKEY", "Page Up (Conv) / Function key (Blk)")
If strRslt = "OK" Then
        MsgBox "Page Up request executed"
Else
        MsgBox "Page Up key not supported by this emulation.", 64
End If
End Sub
```

CrtTypeSet Example

This procedure will report current emulation, then change to VT220.

```
Sub main
Dim strEmu as String
'Determine current emulation type
strEmu = "Current crt emulation is: " & CrtTypeSet$("")
MsgBox strEmu
'Change to VT 220
strEmu = CrtTypeSet$("DEC VT220")
If Len(strEmu) > 0 Then
        MsgBox "New emulation is: " + strEmu
Else
        MsgBox "Unable to locate the VT220 emulation DLL"
End If
End Sub
```

Emit Example

This procedure will wait for a TACL prompt, then Emit the username and password.

```
Sub main
Dim pass as String
Dim LoginID as String
Dim NowTime as Long
Dim intRet as Integer
LoginID = "machine.user"
```

```
Pass = "NewPass"
'In case macro is the startup macro For the session, wait For TACL
'prompt
NowTime = Timer
Do While CRTGet$(CRTRow(-1), CRTCol(-1) - 2, 1) <> ">"
        'Allow 10s For session to start
        if Timer > NowTime + 10 then
                msgBox "Did not detect TACL Prompt", 48
                exit sub
        end if
        DoEvents
Loop
Emit "Logon " & LoginID
intRet = WaitStr(5,"Password:")
Emit Pass
End Sub
```

FtQuery Example

This procedure will create an FTP session and log on.

```
Sub Main
Dim sResult as String
Dim HostName as String
Dim UserName as String
Dim Password as String

UserName = "myname"
HostName = "myhost"
Password = "mypass"
sResult = FtTypeSet$("FTP")
sResult = FtTrigger$("OPEN", HostName)o
Do
```

```
    Waitsilent(1)
    sResult = FtQuery$("STATUS", "")
    Select Case uCase$(sResult)
        Case "INPUTUSERID"
            sResult = FtTrigger$("INPUT", UserName)
        Case "INPUTPASSWORD"
            sResult = FtTrigger$("INPUT", Password)
            exit Do
        Case "NOHOSTCIRCUIT"
            sResult = FtTrigger$("BYE","")
            exit sub
    End Select
    DoEvents
Loop
End Sub
```

FtSet Example

This procedure will send a file to the host using IXF

```
Sub Main
Dim strRet as String
strRet = FTTypeSet$("IXF")
strRet = FTSet$("BINARY", "OFF")
strRet = FTSet$("DELETETABS", "ON")
strRet = FTSet$("OVERWRITE", "ON")
strRet = FTSet$("HOSTNAME", "LINETEST")
strRet = FTTrigger$("SEND", "c:\temp\linetest.txt")
strRet = FTQuery$("STATUS")
Do While FTQuery$("STATUS") = "TRANSFERRING"
Loop
End Sub
```

CrtGet Example

This procedure will wait for a TACL prompt then Emit the username and password.

```
Sub main
Dim pass as String
Dim LoginID as String
Dim NowTime as Long
Dim intRet as Integer
LoginID = "machine.user"
Pass = "NewPass"
'In case macro is the startup macro For the session, wait For TACL
'prompt
NowTime = Timer
Do While CRTGet$(CRTRow(-1), CRTCol(-1) - 2, 1) <> ">"
    'Allow 10s For session to start
    if Timer > NowTime + 10 then
        msgBox "Did not detect TACL Prompt", 48
        exit sub
    end if
    DoEvents
Loop
Emit "Logon " & LoginID
intRet = WaitStr(5,"Password:")
Emit Pass
End Sub
```

CrtPosition Example

This procedure copies screen data to the printer and clipboard

```
Sub Main
Dim intStart as Integer
Dim intEnd as Integer
Dim intRet as Integer
```

```
'Copy the screen contents to a diskfile
intStart = 0
intEnd = CInt(CrtQuery("HEIGHT")) * CInt(CrtQuery("WIDTH")) - 1
intRet = CrtCopy(intStart, intEnd, 0, "screen.txt")
'Copy the screen contents to the clipboard
intRet = CrtCopy(intStart, intEnd, 0, "C")
'Copy the screen contents to the printer
intRet = CrtCopy(intStart, intEnd, 0, "P")
'Copy a column of information to the clip board
intStart = CrtPosition(0, 0)
intEnd = CrtPosition(20, 8)
intRet = CrtCopy(intStart, intEnd, 1, "C")
End Sub
```

FtTrigger Example

This procedure will create an FTP session and log on

```
Sub Main
Dim sResult as String
Dim HostName as String
Dim UserName as String
Dim Password as String

UserName = "myname"
HostName = "myhost"
Password = "mypass"
sResult = FtTypeSet$("FTP")
sResult = FtTrigger$("OPEN", HostName)
Do
        Waitsilent(1)
        sResult = FtQuery$("STATUS", "")
        Select Case uCase$(sResult)
```

```
                Case "INPUTUSERID"

                        sResult = FtTrigger$("INPUT", UserName)

                Case "INPUTPASSWORD"

                        sResult = FtTrigger$("INPUT", Password)

                        exit Do

                Case "NOHOSTCIRCUIT"

                        sResult = FtTrigger$("BYE","")

                        exit sub

        End Select

        DoEvents

Loop

End Sub
```

FtTypeSet Example

This procedure will create an FTP session and log on.

```
Sub Main

Dim sResult as String

Dim HostName as String

Dim UserName as String

Dim Password as String


UserName = "myname"

HostName = "myhost"

Password = "mypass"

sResult = FtTypeSet$("FTP")

sResult = FtTrigger$("OPEN", HostName)

Do

        Waitsilent(1)

        sResult = FtQuery$("STATUS", "")

        Select Case uCase$(sResult)

                Case "INPUTUSERID"
```

```
            sResult = FtTrigger$("INPUT", UserName)
        Case "INPUTPASSWORD"
            sResult = FtTrigger$("INPUT", Password)
            exit Do
        Case "NOHOSTCIRCUIT"
            sResult = FtTrigger$("BYE","")
            exit sub
    End Select
    DoEvents
Loop
End Sub
```

IoInput Example

This function will return the entire response to a TACL command even if the response is more than one screen. The screen data is returned as carriage-return/line-feed delimited lines.

```
function GetTacl(taclCmd as String)


Dim retStr as String, tmpStr as String
Dim charStr as String, charPos as Integer


'Take control of I/O stack
retStr = IoInput (1,0,0)
Emit taclCmd
' Collect response including the ">" prompt.
' NOTE: Processing will stop when ANY character in the Terminate$ parameter
' is detected.
' The maximum String length in VCB is 32767.
retStr = IoInput$(15,32767,4, ">")
' Release the I/O stack
tmpStr = IoInput$(0,0,0)
Emit ""
```

```
'Since many functions Do not handle embedded nulls (e.g. message boxes),
'replace all nulls with spaces.
charStr = chr$(0)
charPos = instr(retStr, charStr)
Do while charPos <> 0
        mid$(retStr, charPos) = " "
        charPos = instr(charPos + 1, retStr, charStr)
Loop
' Replace all EOT's with spaces
charStr = chr$(4)
charPos = instr(retStr, charStr)
Do while charPos <> 0
        mid$(retStr, charPos) = " "
        charPos = instr(charPos + 1, retStr, charStr)
Loop
GetTacl = retStr
end function
```

IoQuery Example

This procedure will set up a direct async connection. Note that a preferable method is to define the session in a new file.

```
Sub Main
Dim strRet as String
strRet = IoTypeSet$("Asynchronous")
strRet = IoSet$("COMPORT", "COM2")
strRet = IoSet$("BAUD", "9600")
strRet = IoSet$("CHARSIZE", "8")
strRet = IoSet$("STOPBITS", "1")
strRet = IoSet$("PARITY", "N")
strRet = IoSet$("COMTARGET", "HOST")
strRet = IoSet$("FLOW", "RTS/CTS")
```

```
MsgBox IoQuery$("*")
strRet = IoTrigger$("CONNECT", "")
End Sub
```

IoSet Example

This procedure will set up a direct async connection. Note that a preferable method is to define the session in a new parameter file.

```
Sub Main
Dim strRet as String
strRet = IoTypeSet$("Asynchronous")
strRet = IoSet$("COMPORT", "COM2")
strRet = IoSet$("BAUD", "9600")
strRet = IoSet$("CHARSIZE", "8")
strRet = IoSet$("STOPBITS", "1")
strRet = IoSet$("PARITY", "N")
strRet = IoSet$("COMTARGET", "HOST")
strRet = IoSet$("FLOW", "RTS/CTS")
MsgBox IoQuery$("*")
strRet = IoTrigger$("CONNECT", "")
End Sub
```

IoTrigger Example

This procedure will set up a direct async connection. Note that a preferable method is to define the session in a new parameter file.

```
Sub Main
Dim strRet as String
strRet = IoTypeSet$("Asynchronous")
strRet = IoSet$("COMPORT", "COM2")
strRet = IoSet$("BAUD", "9600")
strRet = IoSet$("CHARSIZE", "8")
strRet = IoSet$("STOPBITS", "1")
strRet = IoSet$("PARITY", "N")
strRet = IoSet$("COMTARGET", "HOST")
strRet = IoSet$("FLOW", "RTS/CTS")
MsgBox IoQuery$("*")
strRet = IoTrigger$("CONNECT", "")
End Sub
```

IoTypeSet Example

This procedure will set up a direct async connection. Note that a preferable method is to define the session in a new parameter file.

```
Sub Main
Dim strRet as String
strRet = IoTypeSet$("Asynchronous")
strRet = IoSet$("COMPORT", "COM2")
strRet = IoSet$("BAUD", "9600")
strRet = IoSet$("CHARSIZE", "8")
strRet = IoSet$("STOPBITS", "1")
strRet = IoSet$("PARITY", "N")
strRet = IoSet$("COMTARGET", "HOST")
strRet = IoSet$("FLOW", "RTS/CTS")
MsgBox IoQuery$("*")
strRet = IoTrigger$("CONNECT", "")
End Sub
```

WaitCrtCursor Example

This procedure will log a user onto the Tandem m6530 application.

```
Sub Main
dim strRet as string
dim intRet as integer
dim MyUserName as string
Dim MyPass as String
MyUserName = "MyName"
MyPass = "LetMeIn"
'Start mail application
```

```
Emit "m6530"
'Wait for cursor to arrive at "Correspondent Name" field
intRet = WaitCrtCursor(9, 25, 15)
'Send user name
Emit MyUserName
'Wait for cursor to arrive at password field
intRet = WaitCrtCursor(12, 25, 5)
Emit MyPass
intRet = WaitCrtCursor(23, 1, 5)
strRet = CrtTrigger$("FUNCKEY", "Tandem F16")
'Wait for mail screen to open
intRet = WaitCrtUnlock(15)
End Sub
```

WaitCrtUnlock Example

This procedure will log a user onto the Tandem m6530 application.

```
Sub Main
dim strRet as string
dim intRet as integer
dim MyUserName as string
Dim MyPass as String
MyUserName = "MyName"
MyPass = "LetMeIn"
'Start mail application
Emit "m6530"
'Wait for cursor to arrive at "Correspondent Name" field
intRet = WaitCrtCursor(9, 25, 15)
'Send user name
Emit MyUserName
'Wait for cursor to arrive at password field
intRet = WaitCrtCursor(12, 25, 5)
```

```
Emit MyPass

intRet = WaitCrtCursor(23, 1, 5)

strRet = CrtTrigger$("FUNCKEY", "Tandem F16")

'Wait for mail screen to open

intRet = WaitCrtUnlock(15)

End Sub
```

WaitDCD Example

This example will dialup a host and log on.

```
Sub main

Dim strRet as String

Dim PhoneNumber as String

Dim intRet as Integer

' Dial up and login to a host while hiding the process from the user

strRet = CrtTrigger$("SCREEN","OFF")

PhoneNumber = "555-1234"

Emit "ATDT"; PhoneNumber

intRet = WaitDCD(45)

If intRet = 0 Then

        strRet = CrtTrigger$("SCREEN", "ON")

        MsgBox("Call attempt failed")

        Exit Sub

End If

'Login to remote system

Emit ""

If WaitStr(5, ">") = 0 then

        strRet = CrtTrigger$("SCREEN", "ON")

        MsgBox("Never Received TACL Prompt")

        Exit Sub

Emit "logon SUPER"

intRet = WaitStr(5, "Password:")
```

```
Emit "OPENUP"

CrtCls

strRet = CrtTrigger$("SCREEN", "ON")

End Sub
```

WaitKeystrokes Example

This procedure will wait for the user to fill an 8 character field.

```
Sub main

Dim intRet as Integer

Dim strRet as String

intRet = WaitKeyStrokes(0, 8)

'Send F16 to the host

strRet = CrtTrigger$("FUNCKEY", "Tandem F16")

End Sub
```

WaitSilent Example

This procedure will create an FTP session and log on

```
Sub Main

Dim sResult as String

Dim HostName as String

Dim UserName as String

Dim Password as String


UserName = "myname"

HostName = "myhost"

Password = "mypass"

sResult = FtTypeSet$("FTP")

sResult = FtTrigger$("OPEN", HostName)

Do

        Waitsilent(1)

        sResult = FtQuery$("STATUS", "")

        Select Case uCase$(sResult)
        Case "INPUTUSERID"
            sResult = FtTrigger$("INPUT", UserName)
        Case "INPUTPASSWORD"
            sResult = FtTrigger$("INPUT", Password)
            exit Do
        Case "NOHOSTCIRCUIT"
            sResult = FtTrigger$("BYE","")
            exit sub
    End Select
    DoEvents
Loop
End Sub
```

WaitStr Example

This procedure will wait for a TACL prompt, then Emit the username and password.

```
Sub main
Dim pass as String
Dim LoginID as String
Dim NowTime as Long
Dim intRet as Integer
LoginID = "machine.user"
Pass = "NewPass"
'In case macro is the startup macro For the session, wait For TACL
'prompt
NowTime = Timer
Do While CRTGet$(CRTRow(-1), CRTCol(-1) - 2, 1) <> ">"
    'Allow 10s For session to start
    if Timer > NowTime + 10 then
        msgBox "Did not detect TACL Prompt", 48
        exit sub
        end if
        DoEvents
Loop
Emit "Logon " & LoginID
intRet = WaitStr(5,"Password:")
Emit Pass
End Sub
```

WaitTime Example

This procedure will send keystrokes to OutsideView to reconnect a session (Alt+sr).

```
Sub main
AppClassActivate "OutsideView" 'No session may be maximized!
'Wait for application switch
waittime(20)
'Send Alt+sr to reconnect session
SendKeys "%sr",TRUE
End Sub
```

' DDE Example

```
'This procedure will establish a DDE conversation with an Excel spreadsheet.
'It is assumed that Excel is running and that C:\temp\Ov_DDE.xls exists but is not opened.
Sub Main
Dim intRet as Integer
Dim strRet as String
Dim DDEChan as Integer
Dim SprdSht as String

SprdSht = "c:\temp\ov_DDE.xls"
On Error Resume Next
DDEChan = DDEInitiate("Excel", "System")
if Err <> 0 then
        MsgBox "Could not establish DDE conversation with Excel", 48
        exit sub
end if
DDEExecute DDEChan, "[OPEN(""" & SprdSht & """, 0, FALSE)]"
DDETerminate DDEChan
DDEChan = DDEInitiate("Excel", SprdSht)
if Err <> 0 then
        MsgBox "Could not establish DDE conversation with " & SprdSht, 48
        exit sub
end if
strRet = DDERequest(DDEChan, "R1C1")
MsgBox "Cell R1C1 = " & strRet, 64, "R1C1 Contents"
DDEPoke DDEChan,"R2C1","Hello from Outside View"
DDETerminate(ChanNum)
End Sub
```

AppClassActivate Example

This example opens the Windows bitmap file SETUP.BMP in Paint.
Paint must already be open and not minimized before running this example.

```
Sub main
  MsgBox "Opening C:\WINDOWS\SETUP.BMP in Paint."
  AppClassActivate "MSPaintApp"
  DoEvents
  SendKeys "%FOC:\WINDOWS\SETUP.BMP{Enter}",1
  MsgBox "File opened."
End Sub
```

Me Example

An example use of **Me** would be in the script statement

```
        UnloadForm me
```

CrtCol_Example

This procedure will wait for a TACL prompt, then Emit the username and password.

```
Sub main
```

```
Dim pass as String
Dim LoginID as String
Dim NowTime as Long
Dim intRet as Integer
LoginID = "machine.user"
Pass = "NewPass"
'In case macro is the startup macro For the session, wait For TACL
'prompt
NowTime = Timer
Do While CRTGet$(CRTRow(-1), CRTCol(-1) - 2, 1) <> ">"
        'Allow 10s For session to start
        if Timer > NowTime + 10 then
                msgBox "Did not detect TACL Prompt", 48
                exit sub
        end if
        DoEvents
Loop
Emit "Logon " & LoginID
intRet = WaitStr(5,"Password:")
Emit Pass
End Sub
```

QuickSort Program Example

```
Const max% = 5000          ' Maximum length of data to be sorted.
Const ButtonPush   = 2     ' Used to determine why a
Const TextBoxEnter = 3     ' dialog box function was called.
Const IdleLoop     = 5
Dim a(MAX) as Double
Dim count%, StarField%, Flag%, R%, Graphics%
'
' Display stars indicating recursion depth.
```

```
'
Sub Display
   For i%=1 To 1000 : Next i          ' Delay loop
   DlgText StarField, String$(R,"*")
End Sub

'  Sort the array of numbers.  Note that VCBasic allows recursion.
'
Sub QuickSort(LeftSide%, RightSide%)
   Dim v#, t as Double
   Dim i as integer, j%

   If Graphics Then
      R = R+1 : Call Display    ' display recursion level
   End If
   If (RightSide>LeftSide) Then
      v=a(RightSide) : i=LeftSide-1 : j=RightSide : a(0) = v
      Do
         Do : i=i+1 :   Loop Until a(i)>=v
         Do : j=j-1 :   Loop Until a(j)<=v
         t=a(i) : a(i)=a(j) : a(j)=t
      Loop Until (j<=i)
      a(j)=a(i) : a(i)= a(RightSide) : a(RightSide)=t
      Call QuickSort(LeftSide,i-1)
      Call QuickSort(i+1,RightSide)
   End If
   If Graphics Then
      R = R-1 : Call Display
   End If
End Sub
```

```
'  Dialog Box Function for star display dialog box.
'  Every dialog box can have its own dialog box function.
'
Function DlgFunc%(Control$, action%, values&)
   ' a sneaky way to make a dialog box with no button:
   ' create a button but make it invisible.
   If action = 1 Then DlgVisible DlgControlID("Stop"), 0
   If action = IdleLoop Then
      DlgFunc = 1
      If Flag = 0 Then
         Flag = 1
         ' get the ID of the field which will contain stars
         StarField = DlgControlID("Stars")
         Call QuickSort(1, count)
         ' when sorting is done, close the dialog box.
         SendKeys "{enter}"
         Exit Function
      End If
   End If
End Function


'  Verify array size.
'
Function InputFunc%(Control$, action%, values&)
   If (action = ButtonPush) And (Control = "OkBut") Then
     If (Val(DlgText("Data")) <= 0) Or (Val(DlgText("Data")) > 1000) Then
        MsgBox "Invalid list size"
        DlgFocus DlgControlID("Data")
        InputFunc = 1
        End If
    End If
End Function

Sub Main
    Begin Dialog StarBoxType  106, 20, "Recursion Level", .DlgFunc
      Text    5, 8, 101, 10, "Text", .Stars
      PushButton 1, 3,   1,  1, "Stop", .Stop
    End Dialog
    Begin Dialog DataBoxType 20, 30, 186, 47, "Quicksort Parameters", .InputFunc
      TextBox        83,  9, 25, 11, .Data
      OKButton      130,  6, 50, 14, .OkBut
      CancelButton  130, 23, 50, 14
      Text            4, 10, 75, 10, "Size of list (0 - 1000)"
      CheckBox        6, 25, 98,  8, "Animation", .Graphics
    End Dialog

    On Error Goto Done
    Randomize
    Dim DataBox as DataBoxType
    DataBox.Data = "500"            ' Default array size
    Dialog DataBox
    count = Val(DataBox.Data)       ' Actual array size
    t0    = timer
    For i=1 To count : a(i) = Rnd(0.5) : Next i ' make random data
    Dim StarBox as StarBoxType
    Graphics = DataBox.Graphics
    If Graphics Then
        Dialog StarBox
    Else
```

```
        Call QuickSort(1, count)
    End If
    t1 = timer
    Msgbox "elapsed time = "+str(t1-t0), ,"Quicksort Finished"
Done:
    Exit Sub
    Resume Next
End Sub
```

Bitmap Viewer Program Example

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal buf$, ByVal buflen%)
Dim fname$, WinDir$
Const IdleLoop = 5

'Dialog Box Function.  Find and display next bitmap.
Function DlgFunc% (id$, action%, svalue&)
    If action = IdleLoop And (svalue Mod 800 = 799) Then
        fname = dir$
        If fname = ""  Then
            SendKeys "{enter}"
            Exit Function
        End If
        ' load next picture
        DlgSetPicture "p1", WinDir & fname, 0
        DlgText DlgControlID("FileName"), fname
    End If
    If action = IdleLoop Then DlgFunc = 1
End Function

Sub Main
```

```
   Dim WinDirBuf as String * 150
'Find Windows bitmap files
   Call GetWindowsDirectory  (WinDirBuf, Len(WinDirBuf) )
   WinDir = Left(WinDirBuf, InStr(WinDirBuf, Chr$(0))-1) & "\"
   fname  = Dir$(WinDir & "*.bmp")
   If (fname = "") Then Exit Sub

   Begin Dialog PictureBoxType 25, 25, 210, 240, "Picture" , .DlgFunc
      Picture        5,   5, 200, 200, WinDir & fname, 0, .p1
      Text          15, 225,  70,  15, fname, .FileName
      PushButton 145, 220,  45,  15, "Stop"
   End Dialog
   Dim PictureBox as PictureBoxType
   Dialog PictureBox
End Sub
```

Find Files Program Example

```
Option compare binary
Dim count                  '  Number of files searched.
Const DialogInit   = 1       '  Used to determine why the
Const ButtonPush   = 2       '  dialog box function was called.
Const TextBoxEnter = 3
Const IdleLoop     = 5
'
' Function searchFiles finds the files and does the comparison.
' According to user defined flags, it will either use string
' comparison (the InStr function) or regular expressions
' comparison (the Like operator).  The user also chooses
' whether the comparison will be case sensitive or insensitive.
'
```

```
Function searchFiles$(fileSpec$, subPattern$, caseSensitive%, regexp%)
   Dim aLine$
   retVal   = ""
   thisFile = dir$(fileSpec)
   pattern  = subPattern
   While thisFile <> ""
      count = count+1
      Open thisFile for input as #1
      Do While Not Eof(1)
         Line Input #1, aLine
         If regexp Then
            On Error Goto badRegexp
            If Left$(pattern,1) <> "*" Then pattern = "*"+pattern+"*"
            If Not caseSensitive Then     ' convert to upper case
               pattern = UCase(pattern)
               aLine = UCase$(aLine)
            End If
            If aLine Like pattern Then
               retVal = retVal + thisFile + chr$(13)
               Exit Do
            End If
         ElseIf InStr(1, aLine, pattern, 1 - caseSensitive) Then
            retVal = retVal + thisFile + chr$(13)
            Exit Do
         End If
      Loop
      Close #1
      thisFile = dir$
   Wend
   searchFiles = retVal
```

```
    Exit Function
badRegexp:
    MsgBox "Error: Bad regular expression"
End Function
'
' Dialog Box Procedure
'
Function DlgProc%(Control$, action%, values&)
   CR = Chr(13) : TabC = Chr(9)
   HelpText = "Regular expression pattern matching rules:" & CR & CR & _
        "?"  & TabC & _
        "match any single character" & CR & _
        "*"  & TabC & _
        "match any set of zero or more characters" & CR & _
        "#"  & TabC & _
        "match any single digit character (0-9)" & CR & _
        "[chars]" & TabC & _
        "match any single character in chars" & CR & _
        "[!chars]" & TabC & _
        "match any single character not in chars" & CR & CR & _
        "Note: rules are per Visual Basic"
   DlgProc = 0
   Select Case action
      ' disable find button until a search string is entered.
      Case DialogInit : DlgEnable 7, 0
      Case ButtonPush And (values=18)
         ' display help message
         MsgBox HelpText, 0, "Help"
         DlgProc = 1
      Case TextBoxEnter And (Contol="searchPattern")
      ' search string entered, enable find button.
      DlgEnable 7, 1
    Case IdleLoop
      ' whenever the searchpattern is empty, disable the find button
      ' whenever it becomes nonempty, enable the find button
      patternID = DlgControlID("searchPattern")
      If DlgText(patternID) <> "" Then DlgEnable 7,1 Else DlgEnable 7,0
      DlgProc = 1
  End Select
End Function
'
' Prompt user for keyword and filespec.
'
Sub main
   Begin dialog listboxd 30, 50, 165, 110, "Document Search", .DlgProc
      text         10, 10, 60, 15, "&Files to Search:"
      textbox       70,  7, 75, 15, .files
      text         10, 27, 60, 15, "&Search Pattern:"
      textbox       70, 24, 75, 15, .searchPattern
      checkbox      25, 75, 85, 15, "Match Case", .xcase
      checkbox      25, 90, 85, 15, "Use Pattern Matching", .regexp
      buttongroup .but
      button        25, 55, 60, 15, "Find"
      button       110, 90, 40, 15, "Help"
      cancelbutton  90, 55, 60, 15
   End dialog
   On Error Goto Cancelled
   Dim SearchBox as listboxd
   SearchBox.files = "*.VCBasic"
   SearchBox.xcase = 0
   Dialog SearchBox
   fileList = searchFiles(SearchBox.files, SearchBox.searchPattern, _
          SearchBox.xcase, SearchBox.regexp)
   If fileList = "" Then
      MsgBox "Pattern " & """" & SearchBox.searchPattern & """" _
      & " not found in " & count & " file(s)"
   Else
      MsgBox fileList
   End If
Cancelled:
   Exit Sub
   Resume
End Sub
```

Greatest Common Factor Program Example

```
Dim msg$          ' Module-level variable, visible to all functions below
Const ButtonPush   = 2     ' Dialog box actions
Const TextBoxEnter = 3


'
' In this function, the greatest common factor is computed.
'
Function gcf% ( u%, v% )
   dim t%


   If ( u < v ) Then t=u  Else t=v
   While ( (u mod t) <> 0) OR ( (v mod t) <> 0)
     t=t-1
   Wend
   gcf = t
End Function
```

```
'
' CheckNumbers verifies both numbers are positive.
'
Function CheckNumbers% (Control$, action%, values&)

  If action = TextBoxEnter Then
    If Val(DlgText$(Control)) < 1 Then
      RetVal = 1
      DlgText "errmsg", "Bad number, please reenter"
      DlgFocus Control
    End If
  ElseIf action = ButtonPush and values = 16 Then
    a = Val(DlgText$("num1"))
    b = Val(DlgText$("num2"))
    If a < 1 Or a <> Int(a) Then
      RetVal = 1
      MsgBox "Bad number, please reenter"
      DlgFocus "num1"
    ElseIf (b < 1 Or b <> Int (b)) And RetVal = 0 Then
      RetVal = 1
      MsgBox "Bad number, please reenter"
      DlgFocus "num2"
    Else                  ' no error found, ok to print out answer
      DlgText "errmsg", "The answer is " & gcf(a,b)
      RetVal = 1
    End If
  End If
  CheckNumbers = RetVal      ' if RetVal = 0, dialog box will be exited
End Function
```

```
'
' Showdlg creates and displays the dialog box, prompting the
' user to input the two numbers.
'
Sub Showdlg

  Begin dialog enter2num 60,60,150,50, " ** G C F **",.CheckNumbers
    text     3,  4,  40, 10,   "first number"
    textbox 60,  2,  25, 12,                   .num1$
    text     3, 18,  70, 10,   "second number"
    textbox 60, 18,  25, 12,                   .num2$
    text     5, 35, 130, 10,    msg$,          .errmsg

    OptionGroup  .but
    PushButton 100,  1,  40, 15,   "OK",           .okBut
    PushButton 100, 18,  40, 15,   "Cancel",        .cancelBut
  End dialog

  Dim InputDlg as enter2num
  InputDlg.num1$="0"
  InputDlg.num2$="0"
  Dialog InputDlg
End Sub

Sub Main
  Call Showdlg
End Sub
```

Hello World Program Example

Demonstrates calls to subroutines and functions

```
'  MessageBox and GetCurrentTime are calls to functions defined in
'  user.dll.
Declare Sub MessageBox LIB "user.dll" (BYVAL h%, BYVAL t$, BYVAL c$, BYVAL u%)
Declare Function GetCurrentTime& LIB "user.dll" ()

'
' Function CAT$ concatenates two strings with a space between them
'
Function Cat$(a$, b$)
   Cat = a & " " & b
End Function

'
' Subprogram Say computes the time and display a message box.
'
Sub Say(what$)
   Dim min, sec, hrs

   sec = GetCurrentTime () /1000
   min = sec / 60 : sec = sec mod 60
   hrs = min / 60 : min = min mod 60

   Dim eTime as variant                    ' DIM can now be anywhere
   eTime = Format$(hrs,"00") & ":" & Format$(min,"00") & ":" & Format$(sec,"00")
   MessageBox 0, what, "Elapsed Time is " & eTime, 64
End Sub

Sub Main
   Dim msg$
```

```
    If (Command$ = "") Then msg$ = "world" Else msg$ = Command$

    Say Cat("Hello", msg$)

End Sub
```

A cell is a particular character position on the screen or in the CRT image.

A metacommand is a command that gives the compiler instructions on how to build the program.

In VCBasic, metacommands are specified in comments that begin with a dollar sign ("$").

A script is a set of instructions, written in VCBasic, which execute for a specific control or form at run time.

Control Flow and Assignment

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

Clipboard

The **Selection** pointer allows you to select an object or form. Selected objects can be moved, resized, grouped, etc., and have their properties and tasks defined.

Numeric Operators

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

String Operators

&	String concatenation
+	String concatenation

Comparison Operators (Numeric and String)

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

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

Logical Operators

Not	Unary Not	operand can be **Integer** or **Long**. The operation is performed bitwise (one's complement).
And	And	operands can be **Integer** or **Long**. The operation is performed bitwise.
Or	Inclusive Or	operands can be **Integer** or **Long**. The operation is performed bitwise.
Xo	Exclusive Or	operands can be **Integer** or **Long**. The operation is performed bitwise.

r		
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).

call by reference

Arguments passed by reference to a procedure can be modified by the procedure. Procedures written in Basic are defined to receive their arguments by reference. If you call such a procedure and pass it a variable, and if the procedure modifies its corresponding formal parameter, it will modify the variable. Passing an expression by reference is legal in Basic; if the called procedure modifies its corresponding parameter, a temporary value will be modified, with no apparent effect on the caller.

control ID

This can be either a text string, in which case it is the name of the control, or it can be a numeric ID. Note that control IDs are case-sensitive and do not include the dot that appears before the ID. Numeric IDs depend on the order in which dialog controls are defined. You can find the numeric ID using the **DlgControlID** function.

dialog control

An item in a dialog box, such as a list box, combo box, or command button.

function

A procedure that returns a value. In VCBasic, the return value is specified by assigning a value to the name of the function as if the function were a variable..

label

A label identifies a position in the program at which to continue execution, usually as a result of executing a **GoTo** statement. To be recognized as a label, a name must begin in the first column, and must be immediately followed by a colon (":"). Reserved words are not valid labels.

metacommand

A metacommand is a command that gives the compiler instructions on how to build the program. In VCBasic, metacommands are specified in comments that begin with a dollar sign ($).

name

A VCBasic name must start with a letter (A through Z). The remaining part of a name can also contain numeric digits (0 through 9) or an underscore character (_). A name cannot be more than 40 characters in length. Type characters are not considered part of a name.

precedence order

The system VCBasic uses to determine which operators in an expression to evaluate first, second, and so on. Operators with a higher precedence are evaluated before those with lower precedence.

Operators with equal precedence are evaluated from left to right. The default precedence order (from high to low) is: numeric, string, comparison, logical.

subprogram

A procedure that does not return a value.

type character

A special character used as a suffix to a name of a function, variable, or constant. The character defines the data type of the variable or function. The characters are:

Dynamic String		$
Integer		%
Long integer		&
Single	single precision floating point	!
Double	double precision floating point	#
Currency exact fixed point		@

vartype

The internal tag used to identify the type of value currently assigned to a variant. One of the following:

Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date	7
String	8
Object	9

See Also

AppActivate

SendKeys

Shell

See Also

Input Function

Input Statement

IOInput Function

Include

Arrays

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

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

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

where *subscriptRange* is of the format:

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

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

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

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

Numbers

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

Objects

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

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

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

Dim *variableName* **As New** *className*
variableName.methodName

Records

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

Dim *variableName* **As** *typeName*

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

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

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

Strings

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

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

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

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

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

Variants

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

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

Variant variables are initialized to vartype **Empty**.

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

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

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

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

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

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

button	

SHOWSTATUSDIALOG

If set to OFF, the IXF transfer status dialog will NOT display while transferring files; this allows for "silent" transfers.

Also, if set to OFF, the STATUSPAUSE setting will be ignored, since STATUSPAUSE waits for the user to click OK on the status dialog before terminating the transfer.

If set to ON, the IXF transfer status dialog will appear (default setting) and the STATUSPAUSE setting will apply.

STATUSPAUSE

When the STATUSPAUSE setting is ON, the file transfer status dialog remains displayed after the transfer is complete, allowing for review of the transfer information. The user must click OK.

If STATUSPAUSE is set to OFF, the file transfer status dialog disappears immediately after the transfer is complete.

If SHOWSTATUSDIALOG is set to OFF, the setting of STATUSPAUSE is ignored, since users will not be able to click on a dialog that is never shown.

Index