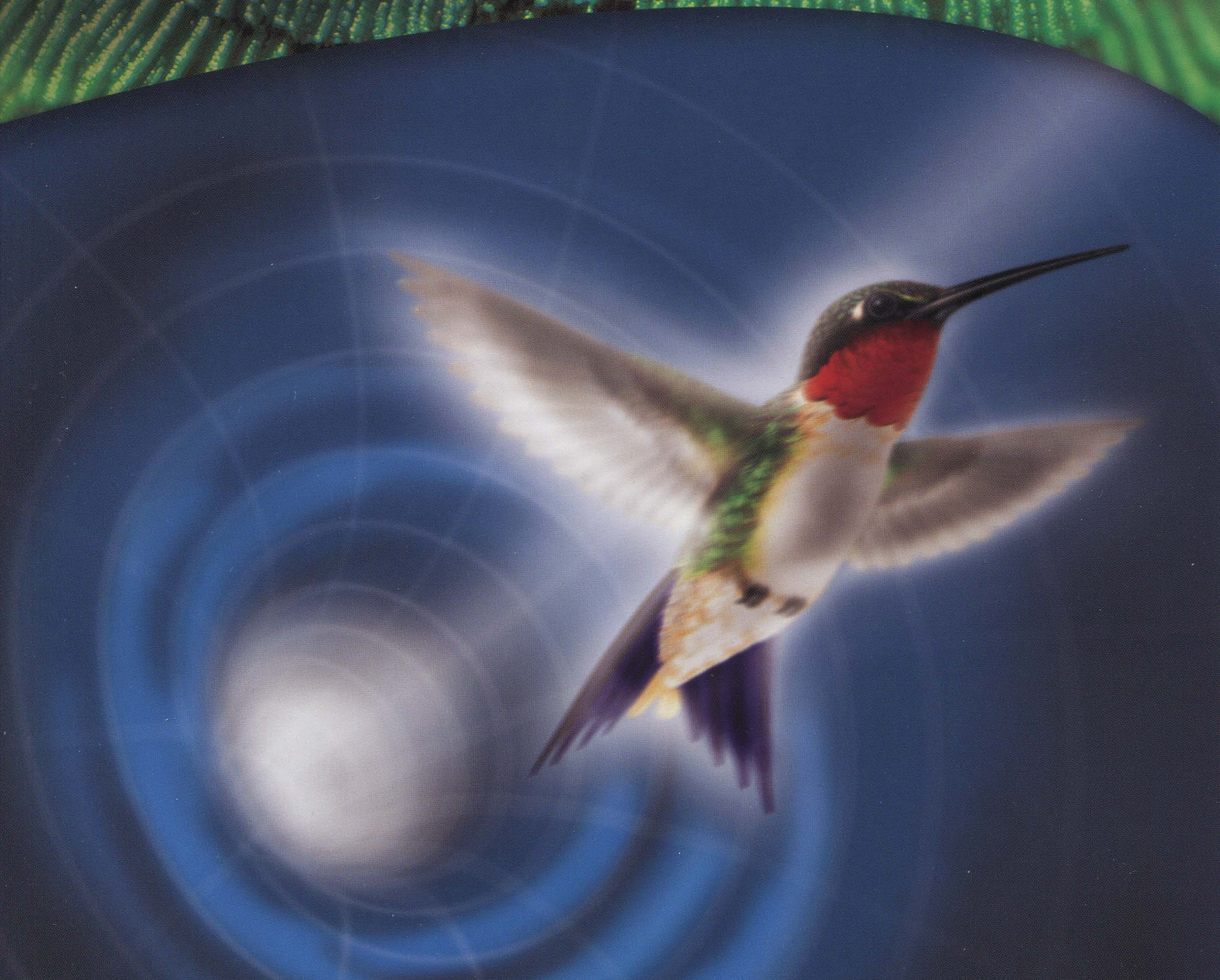
Hummingbird Basic™ Language

Programmer's Guide

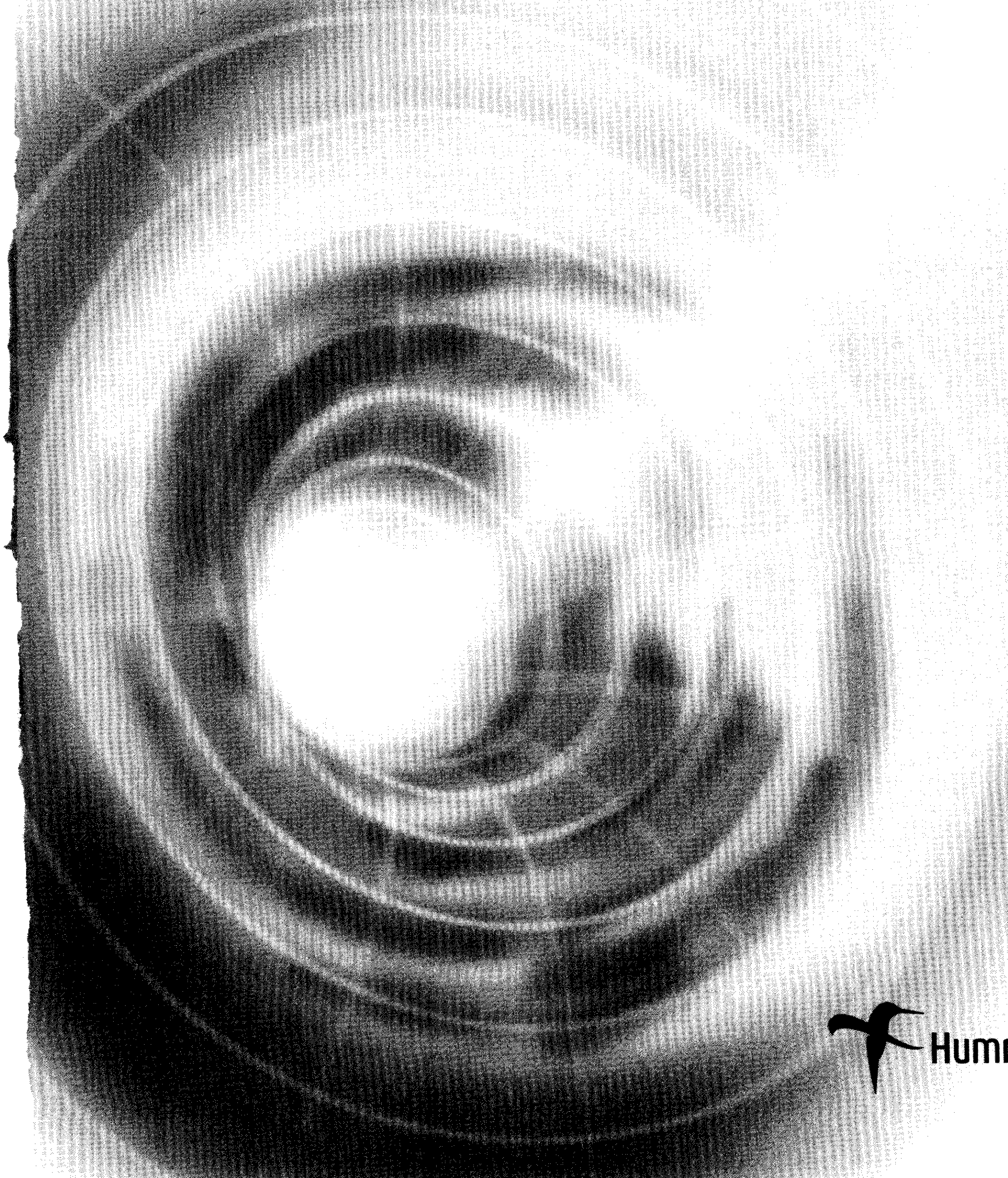


Transforming Information into Intelligence™

Hummingbird™

Hummingbird Basic Language™

Programmer's Guide



8157-6M

Hummingbird™

Hummingbird Basic Language Programmer's Guide
0800 8157-6M
05/31/02

Hummingbird Ltd.
1 Sparks Avenue, Toronto, Ontario, Canada M2H 2W1
Tel: +1-416-496-2200   Toll Free Canada/USA: 1-877-FLY-HUMM (1-877-359-4866)
Fax: +1-416-496-2207
E-mail: support@hummingbird.com or getinfo@hummingbird.com
FTP: ftp.hummingbird.com
For more information, visit www.hummingbird.com

ACKNOWLEDGEMENTS Portions of the code have been contributed by MIT. Portions copyright © Blue Sky Software Corporation. All rights reserved. All other copyrights, trademarks, and tradenames are the property of their respective owners.

Related Documentation and Services

Manuals

All manuals are available in print and online. The online versions require Adobe Acrobat Reader 5.0 and are installed only if you do a Complete installation.

Help

The online Help is a comprehensive, context-sensitive collection of information regarding your Hummingbird product. It contains conceptual and reference information, and detailed, step-by-step procedures to assist you in completing your tasks.

Release Notes

The release notes for each product contain descriptions of the new features and details on release-time issues. They are available in both print and HTML. The HTML version can be installed with the software. Read the release notes before installing your product.

Hummingbird Exposé Online

Hummingbird Exposé Online is an electronic mailing list and online newsletter. It was created to facilitate the delivery of Hummingbird product-related information. It also provides tips, help, and interaction with Hummingbird users. To subscribe/unsubscribe, browse to the following web address:

```
http://www.hummingbird.com/expose/about.html
```

User Groups and Mailing Lists

The user group is an unmoderated, electronic mailing list that facilitates discussion of product-related issues to help users resolve common problems and to provide tips, help, and contact with other users.

To join a user group:

Send an e-mail to `listserv@hummingbird.com`. Leave the Subject line blank. In the body of the e-mail message, type the following:

```
subscribe exceedusers Your Name

subscribe hostexplorer-users Your Name

subscribe nfsmaestro-users Your Name
```

To unsubscribe:

Send an e-mail to `listserv@hummingbird.com`. Leave the Subject line blank. In the body of the e-mail message, type the following:

```
unsubscribe exceedusers Your Name

unsubscribe hostexplorer-users Your Name

unsubscribe nfsmaestro-users Your Name
```

To post a messages to the user group:

Send your e-mail to:

```
exceedusers@hummingbird.com

hostexplorer-users@hummingbird.com

nfsmaestro-users@hummingbird.com
```

To search the mailing list archives:

Go to the following web site:

```
http://www.hummingbird.com/support/usergroups.html
```

Contents

Chapter 1

Introducing Hummingbird Basic

About Hummingbird Basic

Hummingbird Basic is a fully functional language that includes a Workbench for writing and compiling scripts, and a graphical drag-and-drop Dialog Editor for creating and designing an interface. Hummingbird Basic can be used to create scripts for the tasks you frequently perform and want to automate. For example, scripts can be created to automate routine tasks. The following are some common tasks that may require a Hummingbird Basic script:

- If you often edit specific files on your PC, then transfer these files to several UNIX hosts. Create a script using the FTP API functions that will connect to the host, transfer the designated files, and then disconnect.
- If you need to perform the same actions on several IBM 3270 or 5250 hosts at the same time. Create a script file with the HLLAPI functions. This saves you from maintaining the same shell script on a number of different 3270 hosts.
- If you configure your computer differently depending on what you are working on, you could write a script to change your PC configuration back and forth. The script file would allow you to quickly and easily change the configuration without having to manually edit the files each time.

In addition to the Hummingbird Basic statements and functions, there is a set of API and OLE function calls which you can use to customize the following Hummingbird applications:

- FTP
- HostExplorer

Hummingbird Basic also supports a number of Xlib API functions. These functions are used to create X clients for your PC.

Note: Xlib API commands are available only if you purchased the Exceed product. Use only the applications that have OLE API libraries with Hummingbird BASIC.

Development Tools

Hummingbird Basic includes the following development tools:

Workbench A development environment to write, compile and debug your scripts.

Dialog Editor Accessed from Workbench, this drag-and-drop dialog box editor lets you design a dialog box without having to manually code one. When you are finished designing, the code for the dialog box is automatically generated and updated into your script.

Hummingbird Basic Features

If you are familiar with older versions of BASIC (those that predate Windows), you will notice that Hummingbird Basic includes many new features and changes from the language you have learned. Hummingbird Basic 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 versions of BASIC and Hummingbird Basic.

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 the Language

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

Variable Scope

The placement of variable declarations determines their scope.

Data Types

Modern BASIC is now a typed language. In addition to the standard data types—numeric, string, array, and record—Hummingbird Basic also includes variants and objects.

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

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

Dialog Box Handling

Hummingbird Basic 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).

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

Hummingbird Basic also includes statements and functions to display other types of boxes:

- Message Boxes—Notify the user of an event.
- Password Boxes—Do not echo the user's keystrokes on the screen.
- Input Boxes—Prompt for a single line of input.

Financial Functions Hummingbird Basic 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 flow.

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 Hummingbird Basic is an OLE automation controller. Any OLE-enabled application can be communicated with or controlled through a Hummingbird Basic script.

The object data type permits your Hummingbird Basic code to access other software applications by manipulating the available OLE properties and methods of the other application.

Environment Control Hummingbird Basic includes the ability to call another software application and send keystrokes to the application. Other environment control features include the ability to run an executable program, temporarily suspend processing to allow the operating system to process messages, and return values in the operating system environment.

Chapter 2

Hummingbird Basic Scripts

Sample Scripts

Before starting, you may find it useful to review the provided sample scripts. Source files (`.ebs`) and their associated compiled files (`.ebx`) are located in the user directory under

```
Applications
Data\Hummingbird\Connectivity\version\Accessories\Eb
```

The following sample scripts are provided:

dialog.ebs This sample script displays the various types of dialogs that Hummingbird Basic can use. It also stores information as shown below that you either select or press, and displays it when you press Exit.

- Input Boxes
- OK, Cancel Button
- Text Boxes
- Combo Boxes
- Drop Down Lists
- List Boxes
- Option Groups
- Push Buttons

testftp.ebs FTP automation using OLE. This sample script demonstrates how you can use FTP OLE functions to log onto a host and download a file automatically.

dde.ebs This sample script creates a Program Group called "XXX".

filelist.ebs This OLE example is a Hummingbird Basic macro that facilitates the downloading of files from a CMS or TSO account. It must be run from the "Ready" prompt of a CMS or TSO HostExplorer session.

pastword.ebs This macro copies a screen from HostExplorer, starts Microsoft Word and pastes the screen to Word. You need to have HostExplorer running before you run the script.

sendrecv.ebs This Hummingbird Basic macro prompts for the name of a `.bat` file and executes any file transfer commands (that is Send or Receive) found within it. It must be run from the "Ready" prompt of a CMS or TSO HostExplorer session.

> **Note:** This sample script is provided as is, and is intended solely to help you create your own scripts. It is not supported by Hummingbird Ltd.

test1.ebs This sample script lists the index of the field attribute which contains the field at the given position. You can also simply list each row of the screen instead. The current OIA is displayed below the list box. (Demonstrates usage of `host.rows` and `host.columns` methods).

test2.ebs This script demonstrates how to access information using the Field object. In TCP3270, you can access the screen as an entire string, row by row, or using field objects. The advantage of the field objects is that they are not dependent upon their position.

test3.ebs This is a demonstration of configuring TN3270 using the appropriate method. The Cfg3270 sub-object configures the emulator. Anything that can be configured via the user dialogs can be configured using the Cfg3270 object.

test4.ebs This sample script demonstrates how to perform file transfers to a host system. The file transfer is implemented in an asynchronous manner allowing the script to continue to run while the file transfer is taking place. The method `IsXfer` tests if the file transfer is complete. You can also use the `WaitXfer` method to wait until the file transfer completes.

test5.ebs This sample script demonstrates some of the window functions.

Programming Terminology

A program or a script is a logical series of instructions. Each instruction is based on a set of syntax rules. These rules are interpreted by the compiler. If the syntax in your script is clean and there are no errors, the compiler creates an .ebx file which you can run to carry out your task.

The following elements make up a Hummingbird Basic script:

- Variables—Variables are place holders for values. Variables are declared, named, and assigned a data type.
- Statements—Statements define how a task in the script is carried out. They provide the conditional logic or looping for a procedure. They also define the state of a dialog box such as its display and configuration.
- Functions—A function is a construct which, when executed, returns a value. Hummingbird Basic contains a variety of built-in functions you can use in your scripts. You can also write your own functions.

For more information, see "Structure of a Hummingbird Basic Script" on page 12.

- Procedures—A procedure contains a set of variables and statements which you defined for the script. There are two different types of procedures in Hummingbird Basic: functions and subprocedures. A Hummingbird Basic script can contain one main subprocedure. When the script is run, the main subprocedure will be executed first.

For more information, see "Expressions and Operators" on page 26.

- Expressions—An expression is a collection of terms which perform a mathematical or a logical operation. The terms are either variables or functions that are combined with an operator to evaluate a result. There are several types of operators.
- Error Handling—Error handling is a special set of instructions that enable your script to trap errors which may occur while your script is running.

Additional terminology is included in the Glossary.

Structure of a Hummingbird Basic Script

A Hummingbird Basic script is broken up into manageable procedures, each performing a specific task or set of tasks.

There are two procedure types in Hummingbird Basic:

- Subprocedure—Subprocedures define parameters and do not return values.
- Function procedure—Function procedures return values.

A subprocedure is defined with the `Sub...End Sub` statement. You invoke it, either with the `Call` statement, or by entering it on a line by itself. If you use the `Call` statement, enclose any arguments you are passing to the subprocedure in parentheses. For example, the following two statements are equivalent:

```
GetFTP file1,file2,file3

Call GetFTP(file1,file2,file3)
```

A procedure must be defined in the script *before* it is invoked. If you don't place your procedure above a procedure that references it, then use the `Declare` statement to forward declare a procedure.

All Hummingbird Basic scripts must contain a main subprocedure. The main subprocedure is the starting point of the script. All function procedures must eventually trace back to the main subprocedure. Since the main subprocedure usually calls other procedures, it can be placed near the end of the script.

Note: A Hummingbird Basic script can contain only one main subprocedure.

Variable Scope

The placement of variable declarations determines their scope.

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

Functions and Control Statements

Functions and control statements determine the results of your script. A function calculates and returns values as determined by its arguments. A control statement directs the flow of logic during the execution of commands.

Functions and Function Arguments

Functions return values. You can use arguments to pass information required to compute a returned value. Functions may or may not have arguments.

Arguments may or may not be enclosed within parentheses ( ). Whether or not you use parentheses depends on how you want to pass the argument to the function subprocedure. The argument can be passed 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 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 subprocedure, 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 depending on the outcome of `mysub`.

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

```
Call mysub((X))
mysub(X)
```

or

```
y=myfunction((X))
Call myfunction((X))
```

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

```
Call mysub(X)
mysub X
```

or

```
y=myfunction(X)
Call myfunction(X)
```

Externally declared subprocedures and functions (such as `.dll` functions) can take `byVal` arguments. In this case, those arguments are always passed by value.

Named Arguments

When you call a 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%,suppvalue&)
```

`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 identified by name rather than by their 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 is that you do not need to remember the original order in which they were listed in the syntax.

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 using named arguments is that when you call functions or subroutines that have a mix of required and optional arguments, you do not need to use commas as place holders in the syntax for the optional arguments. 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 could 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.

Control Statements

Control statements provide the flow of logic in your script. These statements direct the script as to when, if, and how a set of commands are performed and executed. The following control statements can be included in your script:

```
If...Then... Else

For...Next

Do...Loop

While...Wend

Select Case

On...Goto
```

This example shows the use of an `If...Then...Else` conditional statement:

```
Sub Main
    If myvariable = 0 Then
      msgbox "Are you sure you want to restart?"
    Else
      msgbox "Are you sure you want to quit?"
    End If
End Sub
```

Variables, Constants, and Data Types

Variables store values that are returned from statements and functions. A variable is given a name, and then assigned a data type. Its data type determines the kind of value that is stored by the variable.

Hummingbird Basic supports standard BASIC data types such as Numeric, String, record, array, and Variant data types. With the exception of Variant type variables, the variable you define can contain only data of the declared type. In addition to this, Hummingbird Basic also supports Dialog Box Records and Objects as data types.

Variables and Constants

The following may be defined in a script:

- Dimensioned Variables
- Defined Constants
- Global Variables
- Static Variables

> **Note:** The name you give to a variable or constant can contain letters, numbers, and underscores. It is generally a good idea to give your variables meaningful names so that they can be easily recalled and understood when debugging your script.

To declare a variable in Hummingbird Basic, use the `Dim` statement. When a variable is declared, it is valid only in the commands that follow the declaration.

Dimensioned Variables

For more information on variable scoping, see "Structure of a Hummingbird Basic Script" on page 12.

If a variable is declared at the beginning of your script with the `Dim` statement, it is available throughout the script. To reduce the scope of a variable to a function or a subprocedure, either declare the variable in the function, or in the body of the subprocedure. For example:

```
Function interact(id$)

Dim myvariable as Integer

End Function
```

Defined Constants

Defined constants retain the value they are assigned throughout a script, whenever they are referenced in a function or statement.

Constant variables are declared with the `Const` statement. For example:

```
Const conPI= 3.14159265358979
```

Global Variables

Declare a global variable only if you want to keep the same variable type for all of your related Hummingbird Basic modules. Global data is shared across all loaded modules. If an attempt is made to load a module that has a global variable of a different data type than the existing global variable of the same name, the module load will fail.

Note: It is best to limit global variable usage.

Static Variables

For more information on static variables, see Hummingbird Basic Language Help.

A Static variable retains its value when it is called from one function to another. These variable types are generally used by advanced users.

Data Types

As you name and declare your variable, you assign it a data type. The data type determines what kind of value is stored in the variable. The variable can only contain data of the declared type, except when you implicitly or explicitly declare a variable as a Variant data type.

If a variable is not explicitly defined with the `Dim` or `Global` statements, or is not declared a data type (implicitly declared), then it defaults to the Variant data type.

Note: It is generally good programming practice to explicitly declare all your variables. If variables have not been declared, it may be impossible to track errors that arise in a long and complicated script. To force variable declaration, use the `Option Explicit` command.

The following data types are supported by Hummingbird Basic:

- Variant
- Numeric
- String
- Object

Another way to explicitly declare a variable and its type, without having to type out the entire syntax, is to use data type characters. Data type characters are appended to the end of your variable name.

For example, these two statements are equivalent:

```
Dim bird As String

Dim bird$
```

The following data type characters can be used:

Character	Type	Description
$	Dynamic String	Alphanumeric
%	Integer	1 byte
&	Long Integer	2 bytes
?	Portable integer	
!	Single precision floating point	1 byte
#	Double precision floating point	2 bytes
@	Currency exact fixed point	

Variant

A Variant variable can hold any type of data. This variable changes its data type depending on how it is assigned. To examine the type of data that a Variant variable contains, use the `VarType` function.

Values returned by this function are explained in the table below.

Variant 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

Variant Type	Name	Size of Data	Range
4	Single	4 bytes (float)	-3.402E38 to -1.401E-45 (negative)
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)	January 1st, 0100 to December 31st, 9999
8	String	0 to ~64kbytes	0 to ~64 characters
9	Object	N/A	N/A

Any newly defined Variant defaults to the `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.

`Null` Variants have no associated data, and serve only to represent invalid or ambiguous results. `Null` is not the same as `Empty`, which indicates that a Variant has not yet been initialized.

Numeric

If the variable you declare in your script is a number, you should define its type. There are six Numeric types. These types are shown in the table below.

Type	From	To
Integer	-32,768	32,767
Long	-2,147,483,648	2,147,483,647
Single	-3.402823e+38 0.0, 1.401298e-45	-1.401298e-45, 3.402823466e+38

Type	From	To
Double	-1.797693134862315d+308 0.0, 2.2250738585072014d-308	-4.94065645841247d-308, 1.797693134862315d+308
Currency	-922,337,203,685,477.5808	922,337,203,685,477.5807
PortInt	In Windows it is the same as Integer.	In Windows NT and Windows 95 environments, it is the same as Long.

Note: Hummingbird Basic has no true Boolean variables. Hummingbird 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 using the decimal representation. To represent an octal value, precede the constant with `&O` or `&o`. For example, `&o177`. To represent a hexadecimal value, precede the constant with `&H` or `&h`. For example, `&H8001`.

Note: Constants can also be followed by data type characters.

String

String variables contain text. String length 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. A string can vary in length from 0 to 32,767 characters. There are no restrictions on the type of characters which can be included in a string. For example, the character whose binary value is 0 can also be embedded in strings.

Object

An object is a special data type. Objects let you communicate with another Windows application using OLE automation. You can use Hummingbird Basic as an automation controller to manipulate another application. An object is a complex data type in which the elements of the data type are the methods and properties of the other application.

Properties This determines how an object behaves. For example, width can be a property of a range of cells in a spreadsheet; colors are a property of graphs; and margins are a property of word processor documents.

Methods This causes the application to do something. Examples are: `Calculate` for a spread sheet, `Snap to Grid` for a graph, and `Autosave` for a document.

> **Note:** The Hummingbird Telnet application is an OLE automation server. Telnet contains its own object methods and properties that you can access and manipulate with a Hummingbird Basic script.

Use the `Dim` statement to declare an OLE Object as follows:

```
Dim Telnet as Object
```

Array

An Array is a predefined range or series of variables. You must specify the data type of an array. Hummingbird Basic arrays can be any one of the following:

- Numeric
- String
- Variant
- Record

Arrays of arrays, and dialog box records, are not supported.

Use the following syntax for declaring an array variable:

```
Dim variablename (SubscriptRange, ...) As datatype
```

where `SubscriptRange` is of the format:

```
StartSubscript To EndSubscript
```

For example:

```
Dim lifespan(0 to 75) As Integer
```

Subscripts specify the beginning and ending index for each dimension. If you specify only an ending index, then the beginning index depends on the `Option Base` setting. The `Option Base` statement specifies the lower bound to be used for array subscripts. The lower bound can be either 0 or 1. If no `Option Base` is specified, then the default of 0 is used.

> **Note:** 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.

Dynamic Array

If you do not know what the size of your array is going to be, then use a dynamic array. Dynamic arrays differ from fixed arrays in that you do not specify a subscript range for the array elements when you declare the array. Instead, the subscript range is set using the `ReDim` statement.

For example, you might want to use an array to store a set of values entered by a user, but you do not know in advance how many values the user will enter. In this case, dimension the array without specifying a subscript range, and then execute a `ReDim` statement (which reallocates memory) each time the user enters a new value.

If the dynamic array is dimensioned with the `Dim` statement, then 8 is the maximum number of dimensions it can have. To create dynamic arrays with more dimensions (up to 60), do not `Dim` the array; instead, use 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
```

Record

A record, or record variable, is a data structure containing one or more elements, each of which has a value.

Where an array defines a range of values, all of the same data type (for example, String or Integer), a record variable references a range of values that can be of different data types.

Note: You cannot use data type character suffixes when using record data types.

Before defining a record element as a variable, you must assign each element a type, using the `Type` statement.

The following example defines `phone_number` as a data type:

```
Type phone_number
phone as String
area_code as String
End Type
```

By declaring `phone_number` as a `Type`, you can use it to declare a variable. The elements of each record type are referenced using dot notation. For example:

```
Dim Joe as phone_number
Joe.phone = "967-2222"
```

Note: Records can contain elements that are, themselves, records.

Dialog box records are treated as record data types as well. Elements or controls are referenced using the same `dialogname.controlname` syntax. The difference is that each element is tied to a control of a dialog box.

Expressions and Operators

Expressions perform calculations, set variables, or concatenate strings.

Operators are used in expressions to combine one or more terms. The terms are variables, constants, or functions which are combined with an operator, evaluating to a string or numeric result.

There are several different categories of operators:

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

Numeric Operators

These operators are used in arithmetic expressions:

Operand	Explanation
^	Exponentiation
*,/	Numeric multiplication or division. For division, the result is 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. These can also be used to indicate whether the number is positive or negative.

String Operators

These operators are used to combine or concatenate two or more strings:

Operand	Explanation
&	String Concatenation
+	String Concatenation

Comparison Operators

When using comparison operators with numbers, the operands are widened to the type with the smallest size (Integer is preferred over Long, which is preferred over Double). For String operators, the comparison is case-sensitive, and is based on the collating sequence used by the language specified in the Windows Control Panel.

Operand	Explanation	Returns
>	Greater than	0 for False and -1 for True
<	Less than	0 for False and -1 for True
=	Equal to	0 for False and -1 for True
<=	Less than or equal to	0 for False and -1 for True
>=	Greater than or equal to	0 for False and -1 for True
<>	Not equal to	0 for False and -1 for True

Logical Operators

The logical operators perform logical evaluations on one or more expressions. The result of logical operations is either True or False.

Operand	Explanation
Not	Not operands can be Integer or Long. The operation is performed bitwise (ones complement).
And	And operands can be Integer or Long. The operation is performed bitwise.
Or	Inclusive Or operands can be Integer or Long. The operation is performed bitwise.
Xor	Exclusive Or operands can be Integer or Long. The operation is performed bitwise.
Eqv	Equivalence operands can be Integer or Long. The operation is performed bitwise. (A Eqv B) is the same as (Not (A Xor B)).
Imp	Implication operands can be Integer or Long. The operation is performed bitwise (A Imp B) and is the same as ((Not A) Or B).

Programming Tips and Coding Suggestions

The following tips and suggestions are intended to help reduce the errors returned when creating scripts with Hummingbird Basic.

Naming Variables and Constants

The name you give to a variable or to a constant can contain letters, numbers, and underscores. It is advisable to give variables and constants meaningful names so they can be easily recalled and understood when debugging a script.

Global Variables

Limit the use of global variables to avoid a module load failure. Global data is shared across all loaded modules, so when you attempt to load a module which has a different data type variable than that of the existing global variable with the same name, it results in the module failing to load.

Declaring Variables

Explicitly declare all variables, especially so that error tracking is possible in long and complicated scripts. Use the `Option Explicit` command to force the use of variable declarations.

Option Base

The `Option Base` statement specifies the lower bound to be used for array subscripts. This statement is not allowed inside a procedure, and it must precede any use of arrays in the module. Only one `Option Base` statement is allowed per module.

Dynamic Array

Eight is the maximum number of dimensions for a dynamic array being dimensioned using the `Dim` statement. However, to create dynamic arrays with more dimensions (up to 60), use the `ReDim` statement instead of the `Dim` statement inside your procedure.

Runtime Error

Have a routine in your script that handles runtime errors, such as if the user tries to log onto a non-existent host, or enters text into a field where only numbers are accepted.

Controls

Before aligning the controls for a dialog box, click the Grid toolbar button to turn the grid on.

Compatibility

You can use a single set of source code to create applications that run on Windows NT/95/98/Me/2000. To create an application, load the source code into Hummingbird Basic and make an `.ebx` file.

Checking for the Existence of PC Files

Hummingbird Basic does not provide any built-in means of indicating whether a particular file is on a PC. The usual BASIC technique to check if a file exists is to use either the `DIR` or the `DIR$` function, as shown below. To do this, pass the file name to the `DIR` function and check the return value of the function. If the function returns nothing, then that file does not exist.

```
TheFile$ = Dir$ ("C:\Program Files\Hummingbird\Connectivity\
version\Exceed\exceed.exe
If len(theFile$) < 1 then
  msgbox "no such file"
else
  msgbox theFile$
end if
```

To find a file on a Unix computer, use the same technique, but instead of `DIR$`, use the string returned by the `UNIX ls` *`file name`* command.

Using Win32 API

You do not need the Win32 SDK to make Windows API calls from Hummingbird Basic. Take advantage of Windows API functions to extend the Hummingbird Basic functionality, provided they are properly declared.

```
Declare function GetUserName Lib "advapi32.dll" Alias
"GetUserNameA" (ByVal lpBuffer As String, nSize AS Long) As
Long

Sub Main
    strBuffer$ = String$ (255, 0)
    RetVal& = GetUserName (strBuffer$, 255)
    UserName$ = Trim$ (strBuffer$)
    UserName$ = Left$ (UserName$, Len(UserName$) - 1)
    MsgBox UserName$, , Len(UserName$)
End Sub
```

Network Logon Name

To retrieve a user's network logon name, make the following API call:

```
Declare function GetUserName Lib "advapi32.dll" _
      Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As
Long) As Long
    sub main
      strBuffer$ = String$ (255, 0)
      RetVal& = GetUserName(strBuffer$, 255)
      UserName$ = Trim$ (strBuffer$)
      UserName$ = Left$(UserName$, Len(UserName$) - 1)
      msgbox UserName$, ,Len(UserName$)
    end sub
```

Always Visible Message Box

At times, a message box that was hidden behind other windows may appear giving the impression your application is hung. When this happens, check the Taskbar to discover the message box. If that is problematic, then use the `MessageBox` API function, instead of the `MsgBox` function which allows you to call the message box with the `MB_SYSTEMMODAL` flag, as shown below. This method always displays your message box on top of all other windows.

```
Declare Function MessageBox Lib "user32" Alias "MessageBoxA" _
 (ByVal hwnd As Long, ByVal lpText As String, ByVal lpCaption As
_
String, ByVal uType As Long) As Long

Const MB_ICONEXCLAMATION = &H30&
Const MB_yesno = &H4&
Const IDYES = 6
Const IDNO = 7
Const text = "Please click on one of the buttons below."
Const msg_$ = "Now click on your desktop anywhere outside this
box!"
Const caption_$ = "HUMMINGBIRD Basic Tips"

Sub Main
    dim boxCaption$
    dim boxMsg$
    boxType& = MB_SYSTEMMODAL + MB_ICONEXCLAMATION + MB_YESNO
    if (MessageBox (0, text, caption_$, boxType&) = IDYES) then
        boxCaption$ = "YES Pressed !"
        boxMsg$ = msg_$
' if you click outside this message box it will stay visible
    else
        boxCaption$ = "NO Pressed !"
        boxMsg$ = msg_$
    end if

    MsgBox boxMsg$, ,boxCaption$
End Sub
```

Working with Windows Registry

The following example shows the usage of some of the main registry functions, and how they have to be declared.

```
Declare function RegOpenKey Lib "advapi32.dll" _
    Alias "RegOpenKeyA" (ByVal hkey?, ByVal SubKey$, key&) As
Long
Declare function RegSetValueEx Lib "advapi32.dll" _
    Alias "RegSetValueExA" (ByVal hkey&, ByVal subKeyStr$,
ByVal _
    fdwType&, ByVal dattype%, ByVal data$, ByVal datLen&) As
Long
Declare function RegCloseKey Lib "advapi32.dll" (ByVal hkey&)
As Long

    Function SetValue$(keyname$, value$)
        dim key&

        if RegOpenKey (HKEY_CLASSES_ROOT, "", key) <>
ERROR_SUCCESS then
            SetValue = "Cannot open key: HKEY_CLASSES_ROOT"
            Exit Function
        end if

        if RegSetValueEx (key, keyname, REG_SZ, 0, value,
len(value)) <> ERROR_SUCCESS then
            SetValue = "Cannot set value of key: " + keyname
        end if

        if RegCloseKey (key) <> 0 then
            SetValue = "Cannot close key: " + keyname
        end if
    End function

Sub Main
```

OLE Functions

Use OLE automation to work with FTP and Telnet using Hummingbird Basic.

The following two examples show you how you could execute an FTP session.

1 You have to declare an object as a data type before you can use the object's methods.

```
    dim FtpEngine As Object
    dim FtpSession As Object
    dim FtpSessions As Object
' Must first initialize Ftp Engine
    Set FtpEngine = CreateObject ("HclFtp.Engine")

'Create collection of sessions
    on error goto FtpSessionsError
    Set FtpSessions = FtpEngine.Sessions

    'Create FTP session
    on error goto FtpSessionError
    SetFtpSession = FtpSessions.NewSession

    FtpSessions.LocalDefaultDirectory = "c:\temp"
'normally should be_ taken via dialog
```

2 Make all other initializations.

```
    FtpSession.ConnectToHost
    FtpSession.Userlogin
    FtpSession.Mget "hostfiles"'transfer files
    FtpSession.DisconnectFromHost
'close connection and destroy objects
    Set FtpSession = Nothing
    SetFtpSessions = Nothing
    FtpEngine.Quit
    Set FtpEnging = Nothing
```

The following example shows how the start of a Telnet session can look:

```
    ' if current EMPTY telnet session exists, get it as a tn
object or step to the next line:
    Set tn = GetObject (, "Hummingbird.Telnet")
'if failed to get existing object, create new telnet_ object
    If tn is Nothing then
       Set tn = CreateObject("Hummingbird.Telnet")
    end if

    loginEvent = tn.LookForString(loginPrompt)
'look for the login_ and password prompt
    passwordEvent = tn.LookForString(passwordPrompt)
```

Use the methods and properties of the `tn` object.

Error-Handling and Debugging

Error-handling refers to a set of functions and statements that trap errors arising during the execution of the script. Error-handling is generally one of the most problematic processes.

Error Types

After you compile or run your script, any or all of the following types of errors may be detected:

- Syntax errors—These are errors which occur in the script as a result of misspelling a statement or function or using either one incorrectly, for example, errors in language syntax and programming logic. To help you fix syntax errors, the Hummingbird Script Editor highlights language syntax errors in red after a script is compiled.

  Note: A common syntax error is typing `Endif` instead of `End If`. There is a space between the word End and the word If.

- Logic errors—These are errors that occur because of faulty logic, for example, infinite loops and incorrect values returned by functions. These types of errors generally cause unexpected results during the execution of your script.
- Runtime errors—These errors occur because the user takes an unforeseen action. For example, the user tries to log on to a host that does not exist, or types text into a field that accepts only numbers. You should have a routine for these scenarios included in your script that handles runtime errors. Runtime errors are handled through a set of error-handling functions and statements.

Debugging Scripts for Syntax and Logic Errors

The debugger assists you in locating and correcting syntax and logic errors in your Hummingbird Basic program. It allows you to slow down or suspend the execution of your program so that the flow of the program and the contents of declared variables can be examined. Debug mode is invoked in the following ways:

- Clicking the Step Into toolbar button—This causes the execution of the Main subprocedure in the current script file. Execution is suspended and the debugger is activated. The first line of the Main subprocedure is highlighted.
- Setting breakpoints in the current buffer—Execution is suspended when one of the lines that contains a breakpoint is about to be executed. The debugger is activated, and it highlights the line containing the breakpoint.
- Pressing the Pause toolbar button when a program is executing—Execution is suspended, and the debugger is activated. The line that was about to be executed is highlighted.
- During execution, the program encounters an unhandled runtime error—Execution is suspended, the debugger is activated, and the line containing the error is highlighted.

When in debug mode, the Call Stack Control displays all Hummingbird Basic subprocedures and function calls that got you to the current line. Open the Variables window to examine the contents of variables in the currently selected call frame.

Note: Lines that contain syntax errors appear in red text. The Error Messages and a short description of the error, if available, are displayed in the Output window.

Handling Runtime Errors

Hummingbird Basic provides the following functions and statements to deal with runtime errors in your script:

Function/Statement	Explanation
`Assert`	Trigger an error, if a condition is false.
`Erl`	Return the line number where a runtime error occurred.
`Err Function`	Return a runtime error code.
`Err Statement`	Set the runtime error code.
`Error`	Generate an error condition.
`Error Function`	Return a string representing an error.
`On Error`	Control runtime error handling.
`Resume`	End an error-handling subprocedure.

Trapping Errors

Hummingbird Basic provides two methods for handling errors:

On Error Resume Next Use this statement to bypass an error and continue to execute the script. The `On Error Resume Next` statement must appear before the line that produces the error.

On Error Goto label Use this statement to direct the execution of the script to the specified label. When this error trap is set, it remains in effect until the procedure finishes running. You can redirect the error trap with another `On Error` statement in the procedure. If you want to cancel the existing error trap without setting up another one, use the `On Error GoTo 0` statement.

All error handling subprocedures begin with the `On Error` statement and end either with the `Resume` statement or the `Goto` statement. Unless an `On Error` statement is used, any run-time error terminates the execution of the script. Error-handling procedures are embedded within a subprocedure, usually near the end of a subprocedure. If a `Goto` statement is used, the `Resume` statement is expected at the end of the error-handling code.

To display a description of an error, use the `Error(err)` function as shown below:

```
err = 11
msgbox Error$(Err)
```

The "Division by zero" message is displayed.

Examples of Trapping General Errors

The following examples illustrate the different methods of error trapping.

Example 1

This example places error-handling code immediately following the statement in which the error occurred. It uses the `Resume Next` statement to direct the code to continue execution when an error has occurred.

```
Sub Main
        Dim userdir
inl:    userdrive = InputBox("Enter Drive:",,"C:")
        On Error Resume Next
        Err = 0
        ChDrive userdrive
        If Err = 68 then
            MsgBox "Invalid Drive. Try Again."
            Goto inl
        End If
End Sub
```

The On Error statement identifies the line of code to go to if an error occurs. In this case, the Resume Next parameter continues execution on the next line of code after the error. In this example, the line of code that handles errors is the If statement. It uses the Err statement to determine which error code is returned.

Example 2

This example places error-handling code immediately following a label.

Note: Resume is placed at the end of the error-handling code.

```
Sub Main
    Dim userdir, msgtext
        on error goto Errhdlr1
in2:    userdir = InputBox("Enter Directory.")
' error generated here
        Chdir userdrive & "\" & userdir
       MsgBox "New Default Directory is: " & userdrive & "\" &
userdir
        Exit Sub

Errhdlr1:' handle error here
        Select Case Err
            Case 75
                msgtext = "Path is invalid"
            Case 76
                msgtext = "Path not found"
            Case else
                msgtext = "Error" & err & "" & Error$ & "
occured"
        End Select
        MsgBox msgtext & "Try Again."
        Resume in2' resume normal execution
End Sub
```

The On Error statement used in Option 2 specifies a label to jump to if an error occurs. The code segment is part of the main subprocedure, and it uses the Err statement to determine which error code is returned. To make sure your code does not accidentally fall through to the error handler, precede it with an Exit statement.

Examples of Trapping Runtime Errors

These examples show the two ways to set and trap user-defined errors. Both examples use the `Error` statement to set the user-defined error to the value 30000.

Example 1

To trap the error, the following example places error-handling code directly before the line of code that could cause an error.

```
Sub Main
        Dim custname as String
        On Error Resume Next
in1: Err = 0
        custname = InputBox$("Enter customer name:")
        if custname = "" then
            Error 30000' generate error here
            Select Case Err' handle error here
                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
```

Example 2

The following example contains a labeled section of code that handles any user-defined errors. You can also generate an error code in a subprocedure, and then have the main procedure handle it (similar to example 1 on page 39).

```
Sub Main
        Dim custname as String
        on Error Goto Errhandler
in1:    Err = 0
custname = InputBox$("Enter customer name:")
```

```
        If custname = "" then
            Error 30000' generate error here
        End If
        MsgBox "The name is: " &custname
        Exit Sub
Errhandler:
        Select Case Err' handle error here
            Case 30000
                MsgBox "You must enter a customer name."
            Case Else
                MsgBox "Undetermined Error. Try Again."
        End Select
        Resume in1
End Sub
```

Trappable Errors

The following table lists the runtime errors that Hummingbird Basic returns. These errors can be trapped by `On Error`. The `Err` function can be used to query the error code, and the `Error` function can be used to query the error text.

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

Error code	Error Text
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

Error code	Error Text
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

Chapter 3

Using Development Tools to Edit Scripts

About Hummingbird Basic Workbench

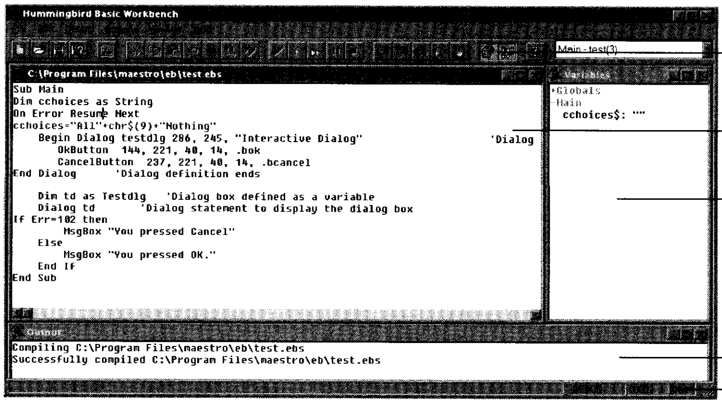
Hummingbird Basic includes an easy-to-use development environment and a graphical dialog box editor. This chapter describes how to use a development tool to write, compile, and debug your scripts.

The Hummingbird Basic Workbench is a special text editor you can use to write, edit, compile and debug your scripts. By default, Hummingbird Basic script files are stored in your *home* directory. The script source files have an `.ebs` file extension. A compiled script file has an `.ebx` file extension.

The Hummingbird Basic Scripting Tool is similar to the Workbench, but only one file can be opened at a time. To start Hummingbird Basic, select it from the Windows Start menu.

The Workbench Interface

The Workbench is divided into the following areas:



```
Sub Main
Dim cchoices as String
On Error Resume Next
cchoices="All"+chr$(9)+"Nothing"
    Begin Dialog testdlg 286, 245, "Interactive Dialog"                     'Dialog
       OkButton  144, 221, 40, 14, .bok
       CancelButton  237, 221, 40, 14, .bcancel
End Dialog      'Dialog definition ends

    Dim td as Testdlg   'Dialog box defined as a variable
    Dialog td        'Dialog statement to display the dialog box
If Err=102 then
        MsgBox "You pressed Cancel"
    Else
        MsgBox "You pressed OK."
    End If
End Sub
```

```
Compiling C:\Program Files\maestro\eb\test.ebs
Successfully compiled C:\Program Files\maestro\eb\test.ebs
```

Code Window

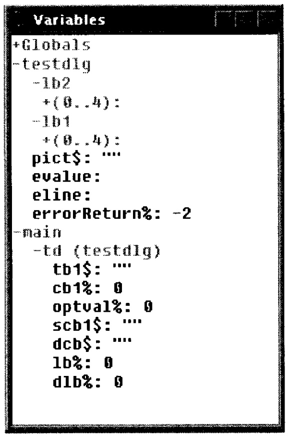
Statements and functions are typed into the Code window. To get help on a specific function or statement, click the right mouse button while the cursor is on the statement or function. Alternatively, highlight the statement or function in the Code window and press F1.

For more information about structuring your scripts, see "Structure of a Hummingbird Basic Script" on page 12.

A Hummingbird Basic script must contain one main subprocedure. Functions referenced in your main subprocedure must be declared before the main subprocedure.

Variables Window

Select Variables on the Window menu to display the Variables window. This window displays the variables you declared in your script. A plus sign beside a heading in magenta text indicates there is an expandable list. Place the cursor next to a plus sign and double-click to see all the variables.



```
+Globals
-testdlg
  -lb2
   +(0..4):
  -lb1
   +(0..4):
  pict$: ""
  evalue:
  eline:
  errorReturn%: -2
-main
  -td (testdlg)
    tb1$: ""
    cb1%: 0
    optval%: 0
    scb1$: ""
    dcb$: ""
    lb%: 0
    dlb%: 0
```

There are three main headings in the Variables window:

- Globals—All global variables declared in any Hummingbird Basic module are shown under this heading.
- Name of your script—The name of the currently loaded script appears as the heading. Variables are listed by their scope in the script.
- The name of the Current Subprocedure—This heading lists all declared variables in the current subprocedure.

Output Window

To open the Output window, either select Output window on the Window menu or click the Output toolbar button.

The Output window provides information about your script after it has been compiled. This window indicates whether the script has been successfully compiled or not. If errors were detected, then they are displayed by an Error Message. Clicking the Next or Previous toolbar button highlights each error in the script.

Status Bar

The status bar indicates the mode in which you are currently working. There are three modes: Edit, Debug and Run. In Edit mode, you can write and compile your script. In Debug mode, you can check for syntax errors and create breakpoints. To revert to Edit mode when you are in Debug mode, click Stop on the toolbar. In Run mode the compiled script is executing. To stop running the script and revert to Edit mode, click Pause. The status bar also lists the number of errors in your script after it has finished compiling.

Call Stack Control

The Call Stack control is visible only while you are in Debug mode. This control indicates which subprocedure the script is executing. This is useful when you are debugging your script for errors. The Call Stack control can also be used to jump to a subprocedure in an open module by selecting one from the drop-down list box.

Creating a Script File at a Glance

You can use Hummingbird Basic scripts for many tasks. These examples describe situations where Hummingbird Basic scripts are beneficial:

- Repetitive tasks—Downloading a file from a remote host to a directory on your PC while you are doing something else.
- Create a simpler interface—Connecting to a host by specifying your login information, selecting the appropriate settings file, and then running a frequently used program in the background while you are doing something else.

- Exchange information between applications—Create a Hummingbird Basic script with OLE automation to transfer data from a Telnet session to an Excel spreadsheet.

The process of creating script files is as simple or as complex as the series of tasks you want to automate.

Creating a script can be broken down into these steps:

1 Identify the task you want to automate and divide it into a sequence of actions.

2 Translate the sequence of actions into Hummingbird Basic commands, and then type them into the Hummingbird Basic Workbench.

   a) Write your script file.

   b) Save your script file.

   c) Compile your script file.

   d) Run and test your script file.

   e) Debug your script file if there are problems.

3 Install a program item icon for your script file.

The following sections describe a simplified process for developing scripts.

To translate the task into a Hummingbird Basic script:

Refer to "Structure of a Hummingbird Basic Script" on page 12 for more information on the order of the functions and statements. For more information on writing Error-Handling routines, see "Error-Handling and Debugging" on page 34.

1 Plan your script by writing down an outline of tasks and end results that you want to accomplish with a script.

2 Find the Hummingbird Basic functions and statements you need in the Hummingbird Basic Language Reference Help.

3 Include Error Handling routines that deal with runtime errors, and any other anticipated user actions in your script.

Compiling and Running a Script File

Before you compile your script, open the Output window. Any error messages that occur in the script appear after the script has finished compiling. To compile your script, either click Check on the toolbar or click Compile on the File menu.

Errors detected in the compiled script appear in red text. To view the errors sequentially through the script, click Next Error and Previous Error on the Edit menu.

Running a Script File

You can run the script only if it has been successfully compiled.

Note: The phrase "successfully compiled" indicates that the script is free of syntax errors. There may be other types of errors in your script, such as runtime or logic errors. Executing the script allows you to test for these other types of errors.

To execute a successfully compiled script file, either click Run on the File menu or click Execute on the toolbar.

Running a Script in Animated Mode

When a script is run in Animated mode, each line of code is highlighted in the Code window as it is executed. This mode is useful for examining loops and other control statements in your script. To run your script in Animated mode, either click Animate on the toolbar or click Animate on the Debug menu.

The following toolbar buttons are available to help you compile and run your script file:

Toolbar Button	Explanation
Output Window	Opens the output window.
✓ Check Script	Compiles your script. All errors will be listed in an open Output window.
▸ Execute Script	Runs a successfully compiled script.
▸▸ Run Script in Animated Mode	Runs a successfully compiled script in animated mode.

Chapter 4

Designing Dialog Boxes

About Dialog Editor

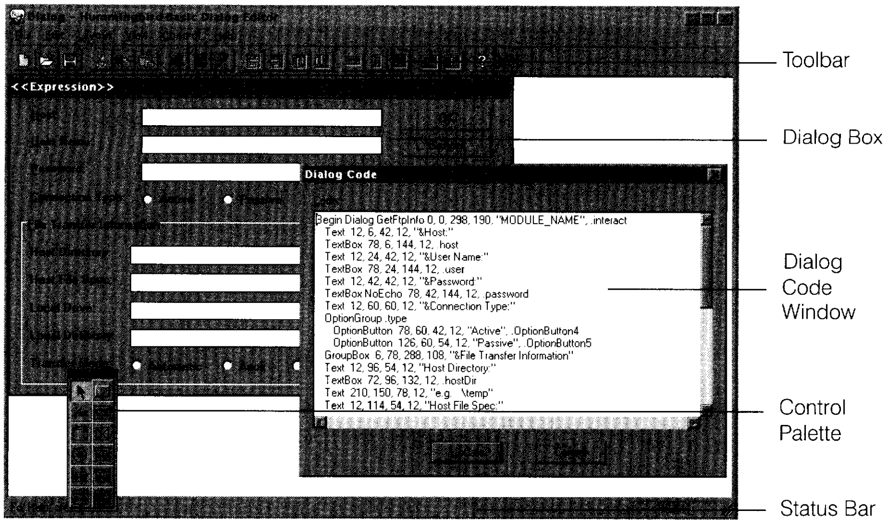
Hummingbird Basic provides both functions and statements, and a graphical Dialog Editor to create dialog boxes. You can run Dialog Editor from either the Workbench's Edit menu or click the Dialog toolbar button.

Dialog Editor lets you create and design dialog boxes by dragging and dropping controls on to a form. As you drop the controls, code is automatically generated and can be dynamically updated into your script as you design the dialog box.

When you first run Dialog Editor, it provides you with a standard-sized dialog box that contains an OK button and a Cancel button. To add a new control, select one on the Control menu, or click the equivalent button on the Control Palette and drag it onto the dialog box window.

Dialog Editor Interface

Dialog Editor is divided into the following areas:



```
Begin Dialog GetFtpInfo 0, 0, 298, 190, "MODULE_NAME", .interact
  Text  12, 6, 42, 12, "&Host:"
  TextBox  78, 6, 144, 12, .host
  Text  12, 24, 42, 12, "&User Name:"
  TextBox  78, 24, 144, 12, .user
  Text  12, 42, 42, 12, "&Password:"
  TextBox NoEcho  78, 42, 144, 12, .password
  Text  12, 60, 60, 12, "&Connection Type:"
  OptionGroup .type
    OptionButton  78, 60, 42, 12, "Active", .OptionButton4
    OptionButton  126, 60, 54, 12, "Passive", .OptionButton5
  GroupBox  6, 78, 288, 108, "&File Transfer Information"
  Text  12, 96, 54, 12, "Host Directory:"
  TextBox  72, 96, 132, 12, .hostDir
  Text  210, 150, 78, 12, "e.g.   \temp"
  Text  12, 114, 54, 12, "Host File Spec:"
```

Toolbar

The toolbar contains the most frequently used commands from the drop-down menus. To get a short description of the toolbar button, place the mouse pointer over top of a button and wait a few seconds for the ToolTip to appear.

Dialog Box

This is the area where you create the dialog box. The dialog box you create will appear in your running script exactly as it appears in the Dialog Box window. By default, when the Editor is first opened there is an OK button and a Cancel button.

Dialog Code Window

This window lets you view and edit the code for the dialog box that you are creating. Click Update to integrate the generated code into your open module.

Note: The Dialog Code window must be closed in order to add or alter controls in the dialog box.

Control Palette

The Control Palette contains all of the controls that can be added to a dialog box. The following table explains what each control is and how to use it.

Control Palette Button	Explanation
PushButton Control	The PushButton control is used to create standard command buttons in the dialog box.
OptionButton Control	The OptionButton is used to present a set of choices. Each option button belongs to a particular OptionGroup, which is configurable from the OptionButton Group drop-down combo box in the OptionButton Properties dialog box.
Text Control	The Text control is used to label other controls that do not have a visible label. To use them as a navigation aid, place them immediately before the control they are labeling in the Tab Order.
TextBox (Edit) Control	The TextBox control accepts text input from a user. A TextBox control is customized (size, position, and so on), by double-clicking it and making the appropriate settings in the TextBox Properties dialog box.
CheckBox Control	The CheckBox control is used to present the user with a two state switch. The switch can be On/Off, Yes/No, Enable/Disable, and so forth.
ListBox Control	The ListBox control is used to present users with a choice from a list of strings.
DropComboBox Control	The DropComboBox control is similar to the DropListBox Control, except that users may type in a new string in addition to selecting one from the list of strings.

Control Palette Button	**Explanation**
GroupBox Control	The GroupBox control visually groups controls in a dialog box. In addition, they can be used to provide a navigational hierarchy to the dialog box user.
DropListBox Control	The DropListBox control differs from the ListBox control in appearance only. If a string from the control is selected, it appears in the control. When the user clicks the down arrow, the control expands to present the list of strings.
Picture Control	The Picture control is used to place bitmaps into the dialog box. Picture controls get their contents from either the clipboard or a Windows bitmap (`.bmp`) file.

Adding Controls to a Dialog Box

Different controls gather specific types of information from the user. An effectively designed interface also helps the user to enter the correct data and to navigate through your program.

There are two ways to add new controls to your dialog box:

- Select a control either on the **Control Palette** or on the **Control** menu then drag out a rectangle in the dialog box. The control of the selected type is created and sized to that rectangle.
- Use the drag-and-drop method to place a control of a default size into your dialog box.

To use the drag-and-drop method:

1 On the Control Palette, click the control you want to add.

2 Press and hold the mouse button, then move the mouse into the dialog box window. A rectangle appear indicating the placement of the control you want to create. You can move the rectangle with the mouse.

3 Release the mouse button to place the control.

Note: To abort creating the control, move the mouse outside of the dialog box window, and release the button.

Aligning Controls in the Dialog Box

There are a number of commands from the Layout menu that can help you align and lay out controls on the dialog box.

To align the controls, select one by clicking it with the mouse. To select multiple controls, drag a rectangle across all of the controls you want selected. Selected controls have a dotted black outline. When the controls are selected, choose a command from the Layout menu.

The following commands are available from this menu:

- Align Controls—Allows you to move selected controls left, right, top, bottom, vertically, or horizontally.
- Space Evenly—Allows you to space selected controls evenly, down, or across.
- Center in Dialog—Allows you to center the selected dialog either vertically or horizontally in the dialog box.
- Arrange Buttons—Allows you to arrange the selected button control to the right or the bottom of the dialog box.
- Make Same Size—Causes selected controls to size exactly the same.
- Size to Content—Causes a control that accepts user input to size itself according to its content.

You may find it useful to turn the grid on before you begin aligning the controls. The grid is enabled by clicking the Grid toolbar button. To change the incremental units of the grid, select Options on the Edit menu. Enabling the Snap To Grid check box in the Option dialog box aligns the controls to the nearest grid unit.

Setting the Tab Order

When the tab order is set, press the tab key to shift the focus from control to control. Setting the tab order allows you to specify the order of control focus when the tab key is pressed.

To set the tab order:

1. On the Layout menu, click Set Tab Order. Small numbers will appear on the left corner of each control.
2. Click each control in the order you want the focus to shift when the user tabs through the dialog box. As you click, a new number appears on each control.

Note: One of the controls in the dialog window will be the primary control. It is identified by the darker black outline when selected. The primary control is always the first in the tab order. All controls will be set relative to the primary control.

Setting Control Properties

Once the controls are placed and aligned on the dialog box, you can begin setting specific properties for each of the controls added. Control properties are settings that affect the attributes and the behavior of the control.

Examples of control properties include position and size, and whether or not an expression is attached to the control. Also, most of the controls have a Control ID. The Control ID is an identifier that you use to reference and access the control from a function in your Hummingbird Basic script.

To display the properties for a control:

- Double-click the control for which you would like to edit the properties.
- If a single control is selected, press the Enter key.
- To access Dialog Properties, double-click an empty area of the dialog box.

Dialog Box Properties

The following properties can be set for a dialog box:

Dialog ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Caption Type & Caption These fields allow you to enter a title for the dialog box. There are three caption types to choose from:

- None—If the caption type is set to None, then the application's default caption is used, and the Caption field is disabled.
- String—Select this type to enter a title for the dialog box into the Caption field.
- Expression—Select this type to enter a Hummingbird Expression into the Caption field.

Macro Function Name Enter the name of the function you are using to update fields with. The function name is appended to the `Begin Dialog` statement. This field is only used in dynamic dialog boxes.

ButtonGroup ID To reference a group of related buttons through the dialog box, enter a name for the group of buttons.

Size and Position A dialog box is positioned relative to the upper left corner of the application. By default, dialog boxes are centered on the application.

- X and Y Position—To specify the position of the dialog box, enable the Edit dialog position box and type the desired values in the X and Y fields. Type either numeric values or Hummingbird Basic expressions into the X and Y fields. If you type a new numeric value in either of these fields, Dialog Editor moves the dialog box accordingly. If you type an expression (non-numeric value), the position of the control or dialog box is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the dialog box accordingly.

Button Control Properties

Button controls are the command buttons that you put on to your dialog box. The following properties can be set for a button control:

Button Type There are three different kinds of push buttons you can add to a dialog box:

- OK—This is like a normal button, except its label cannot be modified. There can be only one OK Button in a dialog box.
- Cancel—This is like a normal button, except its label cannot be modified. There can be only one Cancel Button in a dialog box.
- Normal—If the button is not an OK or a Cancel button, then use this type. This button allows you to assign a label and an ID.

Button Label This property inserts text on to the button. If you selected either an OK button or a Cancel button, then its label cannot be changed. If you want to assign a shortcut key for the selected control, type an ampersand (`&`) before the letter you want to use as a shortcut key. For example, if you type the label for a help button as `H&elp`, users will be able to access help by pressing Alt+E.

Button ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, you should assign IDs using a consistent naming convention.

Use label as a macro input expression If you want to assign a Hummingbird Basic expression as the label, then enable this check box. The label will then be calculated or interpreted when the script is executed.

Size and Position A dialog box is positioned relative to the upper-left corner of the application. By default, dialog boxes are centered on the application. Controls are positioned relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. Type either numeric values or Hummingbird Basic expressions into the X and Y fields. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

OptionButton Control Properties

Option button controls allow a user to enable or disable a function. Option buttons have the following property settings:

OptionButton label This property inserts text beside the button. If you want to assign a shortcut key for the selected control, type an ampersand (&) before the letter you want to use as a shortcut key. For example, if you type the label for a help button as `H&elp`, users will be able to access help by pressing Alt+E.

OptionButton ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

OptionButton group This option allows you to enter a single string for a group of related option buttons. When referring to the group in your function, you can then use this string.

Use label as a macro input expression If you want to assign a Hummingbird Basic expression as the label, then enable this check box. The label will then be interpreted when you execute the script containing this dialog box.

Size and Position A dialog box is positioned relative to the upper-left corner of the application. By default, dialog boxes are centered on the application. Controls are positioned relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. You can type either numeric values or Hummingbird Basic expressions into these fields. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

Text Control Properties

Use text controls to label another control that typically does not have a label. The following properties are available for Text Controls:

Text Label This property inserts a label for a control. If you want to assign a shortcut key for the selected control, type an ampersand (`&`) before the letter you want to use as a shortcut key. For example, if you type the label for a help button as `&Help`, users will be able to access help by pressing Alt+H.

Text ID The ID is a string you can assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Use label as a macro input expression If you want to assign a Hummingbird Basic expression as the label, then enable this check box. The label will then be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic places controls relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

TextBox (Edit) Control Properties

The following properties can be set for TextBox controls:

TextBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Password\no Echo When this option is enabled, any text the user types into the text field is echoed back as asterisks. This feature is used if the `textbox` control will accept passwords as input.

Size and Position Hummingbird Basic positions controls relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

CheckBox Control Properties

Check boxes provide the user with the ability to enable or disable a function in the program. The following properties can be set for a `CheckBox` control:

CheckBox Label This property inserts a label for a control. If you want to assign a shortcut key for the selected control, type an ampersand (`&`) before the letter you want to use as a shortcut key. For example, if you type the label for a help button as `H&elp`, users will be able to access help by pressing Alt+H.

CheckBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Use label as a macro input expression If you want to assign a Hummingbird Basic expression as the label, then enable this check box. The label will then be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the dialog box is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

ListBox Control Properties

A ListBox provides a list of strings from which to choose. You can also add new strings at runtime. The following properties are available for a ListBox control:

ListBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

ListBox Contents This field allows you to enter the strings that will form the contents of the ListBox.

Use content as a macro input expression If you want to assign a Hummingbird Basic expression to the contents of the ListBox, enable this check box. The contents of the ListBox will be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box or control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

StaticComboBox Control Properties

A StaticComboBox is a text box with an attached list box. When the user selects a value from the list box, it is placed in the text box. The following properties can be set for a StaticComboBox control:

StaticComboBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

StaticComboBox Contents This field allows you to enter the strings which will make up the contents of the StaticComboBox. A user can then select one of the strings from the box.

Use content as a macro input expression If you want to assign a Hummingbird Basic expression to the contents of the StaticComboBox, enable this check box. The contents of the StaticComboBox will be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

DropComboBox Control Properties

A DropComboBox is a text box with an attached list box. The list box remains hidden until the user selects the arrow beside the text box to drop down the list box. When the user selects a value from the list box, it is placed in the text box.The following properties can be set for a DropComboBox control:

DropComboBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

DropComboBox Contents This field allows you to enter the strings which will make up the contents of the DropComboBox. A user can then select one of the strings from the box.

Use content as a macro input expression If you want to assign a Hummingbird Basic expression to the contents of the DropComboBox, enable this check box. The contents of the DropComboBox is interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper left corner of the dialog box:

- X and Y Position—To specify the position of the dialog box, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box or control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

DropListBox Control Properties

A DropListBox is a list box that remains closed, showing only one value, until the user selects the arrow on the right-hand side to expand it. The following properties can be set for a DropListBox control:

DropListBox ID The ID is a string you assign to identify the control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

DropListBox Contents This field allows you to enter the strings which will make up the contents of the DropListBox. A user can then select one of the strings from the box.

Use content as a macro input expression If you want to assign a Hummingbird Basic expression to the contents of the DropListBox, then enable this check box. The contents of the DropListBox will be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box or control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

GroupBox Control Properties

GroupBox controls are used as a design feature to group a series of related controls together. The following properties can be set for a GroupBox control:

GroupBox Label This is the title of the group box. The title you type here, appears on the dialog box.

GroupBox ID The ID is a string you assign to identify the GroupBox control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Use label as a macro input expression If you want to assign a Hummingbird Basic expression as the label, then enable this check box. The label will then be interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic places controls relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non1-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box or control. Enter a value in pixels in the Width and Height fields. Dialog Editor sizes the control accordingly.

Picture Control Properties

Pictures are graphics that are used in dialog boxes and windows. The following properties can be set for picture controls:

Picture source This property indicates the source of the bitmap to be displayed: Clipboard or File.

Picture file name Type the name of the bitmap file to display in your dialog box.

Picture ID The ID is a string you assign to identify the GroupBox control in your Hummingbird Basic script file. For easy recognition, assign IDs using a consistent naming convention.

Suppress Message Enabling this check box causes the picture control not to display the "missing picture" warning if the picture for the dialog box cannot be located.

Use file name as a macro expression If you selected File as the picture source, enable this check box to assign a Hummingbird Basic expression corresponding to the file name. The file name is interpreted when you execute the script containing this dialog box.

Size and Position Hummingbird Basic positions controls relative to the upper-left corner of the dialog box:

- X and Y Position—To specify the position of the control, type the desired values in the X and Y fields. These values can be either numeric values or Hummingbird Basic expressions. If you type a new numeric value in either of these fields, Dialog Editor moves the control accordingly. If you type an expression (non-numeric value), the position of the control is interpreted when you execute the script file containing this dialog box.
- Width and Height—These fields allow you to change the size of the dialog box or control. Enter a value in pixels in the **Width** and **Height** fields. Dialog Editor sizes the control accordingly.

Integrating a Dialog Box into Your Script

A dialog box must be defined and declared before you can refer to it in your script. Dialog boxes are defined using the `Begin Dialog...End Dialog` statements.

To integrate a dialog box into your script follow these steps:

1. Define the dialog box with the `Begin Dialog...End Dialog` statements and dialog box definition statements such as `TextBox`, `OkButton`.
2. Create a dynamic dialog function to handle dialog box interactions.
3. Display the dialog box using the Dialog Function.

Defining the Dialog Box

The `Begin Dialog`...`End Dialog` statement defines a dialog box. The last parameter to the `Begin Dialog` statement is the name of a function, prefixed by a period. This function handles interaction between the dialog box and the user.

After defining your dialog box, you must declare a variable of this data type. In the following example, the variable named td refers to the dialog box named testdlg.

```
Begin Dialog testdlg 286, 245, "Interactive Dialog", .interact
<statements that define the controls on your dialog box>
End Dialog
Dim td as testdlg
```

For more information, see "Writing a Dialog Function" on page 74.

If you are writing a function to accept user input and to define what occurs in the dialog box, then enter the function at the end of the Begin Dialog statement. In the above example this is a function called interact.

If you use Dialog Editor, the Begin Dialog.... End Dialog statement is inserted into your code. You must add the function parameter to the Begin Dialog statement and the variable information after the End Dialog statement.

Displaying the Dialog Box

To display the dialog box, you can use the Dialog function. In a Dialog function, the argument to display a dialog box is the variable name that you previously declared. From the example above, this would be td.

Dialog Statements and Functions

The dialog function and the dialog statement differ slightly in their use:

- The Dialog Function—This function both displays a dialog box and returns a number when the user presses any of the command buttons.
- The Dialog Statement—This statement displays a dialog box.

In most cases, use the Dialog Function. If you use a Dialog statement to display the dialog box, then you have to write an error-handling routine at the end of your main subprocedure using the On Error statement.

Dynamic dialog box functions and statements can be used only while a dialog box is displayed on the screen and is calling a dialog control function. These functions and statements are used to get or set information about a particular control in a dialog box.

The functions and statements in this category are:

Function	Explanation
`DlgControl Function`	Returns the numeric ID of a control.
`DlgEnable Function`	Returns True (-1) if the specified control is enabled, or 0 (False) if it is not.
`DlgEnable Statement`	Enables or disables a control.
`DlgFocus Function`	Returns the ID of the control having input focus.
`DlgFocus Statement`	Sets focus to a control.
`DlgListBoxArray Function`	Returns the contents of a list box or combo box.
`DlgListBoxArray Statement`	Sets the contents of a list box or combo box.
`DlgText Function`	Returns the text value for a control.
`DlgText Statement`	Sets the text for a control.
`DlgValue Function`	Returns the value of a control.
`DlgValue Statement`	Set the value of a control.
`DlgVisible Function`	Returns True (-1) if the specified control is visible, or False (0) if it is not.
`DlgVisible Statement`	Makes a control visible or invisible.

Most of these functions and statements take the Control ID as the first argument. For example, consider the following check box definition:

```
CheckBox 20, 30, 50, 15 "My checkbox", .check1
```

Use the following command to disable the check box:

```
DlgEnable "check1", 0
```

The following function returns -1 if the check box is selected, or 0 if it is not:

```
DlgValue ("check1")
```

Control IDs are case-sensitive. In dynamic dialog box functions and statements, control IDs are in quotation marks and do not include the period that is required in control definitions (between `Begin Dialog ...End Dialog` statements).

Dynamic dialog functions and statements can also work with numeric IDs, which are automatically assigned in the order in which dialog controls are defined. For example, if a check box is the first control defined in the dialog record, `DlgValue (0)` is equivalent to `DlgValue ("Check1")`. Control numbering begins at 0. Labels are not numbered.

The example below creates a dialog box with a drop-down combo box within it, and the three buttons: **OK**, **Cancel**, and **Help**. The Dialog Function used here enables the subprocedure to trap when the user clicks any of these buttons.

```
Sub Main
    Dim cchoices as String
    cchoices = "All" + Chr$(9) + "Nothing"
        Begin Dialog UserDialog 180, 95, "Hummingbird Dialog
Box"
               Text 9, 3, 69, 13, "File name:", .Text1
               ButtonGroup .ButtonGroup1
               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
```

Writing a Dialog Function

A function defines the behavior of the dialog box. For example, your function could disable a check box based on the user's action. The body of the function uses the Hummingbird Basic statements and functions prefixed with `Dlg` to define dialog box actions.

To define the function itself, use the `Function...End Function` statement, or declare it using the `Declare` statement before using the `Begin Dialog` statement.

The name of the function is entered in dot notation at the end of the `Begin Dialog` statement. In the example below, `interact` is appended to the end of the `Begin Dialog` statement. `Interact` is a function that determines what occurs when a user presses a button on the dialog box.

```
Begin Dialog testdlg 286, 245, "Interactive Dialog", .interact
<statements that define the controls on your dialog box>
End Dialog
Dim td as testdlg
```

The function receives the following three parameters from the `Begin Dialog` statement:

- The Identifier parameter—The first argument, `id$`, identifies the control associated with the call to the Dialog Function. It is the same value which appeared in the definition of the control. This is the control ID string that identifies each of the buttons and fields in your dialog box.
- The Action parameter—`Action%` is an integer between 1 and 5 identifying the reason why the Dialog Function is called.
- The Suppval parameter—This parameter supplies additional information to the dialog box function, `suppval&` gives more specific information than the *`action`* argument.

The Dialog Function does not return until the dialog box is closed. To leave the dialog box open after the user clicks a command button (such as the OK button), return a non-zero `suppval`.

The following table explains the meaning of each value that `action%` can contain:

Value	Meaning
1	Corresponds to dialog box initialization. This value is passed before the dialog box becomes visible.
2	Corresponds to choosing a command button or changing the value of a dialog box control (except for typing in a text box or combo box).
3	Corresponds to a 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	Corresponds to a change of control focus. `Id$` is the ID of the control gaining focus, and `suppvalue&` contains the numeric ID of the control losing focus. A Dialog Function cannot display a message box or dialog box in response to an action value 4.
5	Corresponds to an idle state. When the dialog box is initialized (action 1 is passed), the Dialog Function will be continuously called with action 5, if no other action occurs. If the dialog function wants to receive this message continuously, while the dialog box is idle, it should return a non-zero value. If 0 (zero) is returned, action 5 will be passed only while the user is moving the mouse. For this action, `Id$` is equal to empty string ("") and `suppvalue&` is equal to the number of times action 5 was passed before.

When action% is 2 or 3, suppval& depends on the type of the control. The following table summarizes the possible values for suppval:

Control	Suppval
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 Button	2
Push button	An internal button ID. This is not the same as the numeric ID of the button control.

In most cases, the return value of the Dialog Function is ignored. The exceptions are the return values from action% 5 (as discussed above), and from action% 2. If action% 2 is called because the user clicked the OK button, Cancel button, or a command button (as indicated by id$), and the Dialog Function returns a non-zero value, the dialog box will not be closed. To close the dialog box when a user clicks a button, return a 0 to the function.

You can use the information these parameters provide to change the behavior of the dialog. For example:

```
Function interact%(Id as String, Action as Integer, Suppval as
Long Integer)
If Id = "bcancel" and action = 2 Then interact = 0
End If
End Function
```

This example shows that if the user presses the Cancel button, the dialog box closes. Id = bcancel (the button ID for cancel), Action = 2 indicates that the user has chosen a command button. If this occurs, interact = 0, which causes the dialog box to close. If the function returned 1, for example interact = 1, then the dialog box would stay open.

Putting It All Together

The following script shows a dialog box with a text field, a check box, and a hide/show picture button. When you enter text into the text field, it becomes the title for the group box. Clicking the check box enables or disables the **Bell** button. When you click the **Hide** button, the picture is pasted to the Windows clipboard. Note the position and order of the dynamic dialog box functions. Comments are preceded by an apostrophe (') and are ignored by the compiler.

```
option explicit' force declarations
dim pict$' name of the picture file
dim evalue' last error value
dim eline' last error line
const errorReturn = -2' use -2, as -1 = OK, 0 = Cancel and
positive
' numbers are used by other buttons

function interact%(id$, action%, suppval&)
' start of dialog function
    dim s$' scratch string
    dim i?' scratch portint
    on error goto ehandler' error handling
    select case action' switch on the action type
    case 1' dialog box initialization
        dlgValue "cb1", 1' set the checkBox 'ON'
        dlgFocus "tb1"' force focus to text field
        exit function' exit

    case 2' control changes, allow
    case 3' text field changes, allow
    case 4' change of focus
        interact = 1' make sure event continues
        exit function' exit
    end select

    interact = 1' default = Don't terminate
    select case  id' switch on the control
    case "tb1"' text field
        msgbox "Sample Text Field was changed", 64, "Change Of
Focus"
```

```
        case "hide"' hide control
            if dlgVisible("pict") = 0 then' check the state
                dlgVisible "pict", 1' make picture visible
                dlgVisible "bird", 1' make the option visible
                dlgVisible "clipboard", 1' make the option visible
                dlgtext "pg", "Picture"' make the text visible
                dlgtext "hide", "Hide &Picture"' change button
text
            else
                dlgVisible "pict", 0' hide the picture
                dlgVisible "bird", 0' hide the option
                dlgVisible "clipboard", 0' hide the option
                dlgtext "pg", ""' set the text to Null
                dlgtext "hide", "Show &Picture"' change button
text
            end if

        case "bird"' switch to bird picture
            DlgSetPicture "pict", pict, 0

        case "clipboard"' switch to clipboard
            DlgSetPicture "pict", "", 3

        case "bbell"' sound the bell
            beep

        case "cb1"' CheckBox
            dlgEnable "bbell", suppval' enable/disable bell

        case "copy"' update group text
            dlgText "g1", DlgText("tb1")

          case "bok", "bcancel"
            interact = 0' terminate

        case "berror"
            s = "abc"
            i = cint(s)' invalid conversion

        end select
        exit function

    ehandler:' error handler label
        evalue = err' save the error
        eline = erl' save the error line
        resume postError

    postError:
        dlgend errorReturn' exit as error

    end function
```

```
Sub Main' start of Main subprocedure
    dim i?' variable to hold result of dialog box
    pict = homeDir' get bird picture
    if right$(pict, 1) <> "\" then pict = pict + "\"
    pict = pict + "BIRDY3.BMP"
  Begin Dialog testdlg 286, 245, "Interactive Dialog",
.interact
    OKButton 144, 221, 40, 14, .bok
    CancelButton 237, 221, 40, 14, .bcancel
    GroupBox 7, 11, 133, 107, "Group", .g1
    Text 13, 24, 62, 8, "Sample Text Field:"
    TextBox 13, 40, 120, 13, .tb1
    CheckBox 13, 66, 35, 10, "Bell On", .cb1
    Button 64, 64, 60, 14, "&Bell", .bbell
    Button 13, 92, 120, 14, "&Sample Text Field To Group Name",
.copy
    GroupBox 144, 11, 133, 107, "Picture", .pg
    Picture 173, 25, 75, 51, pict, 0, .pict
    OptionGroup .optval
    OptionButton 171, 80, 24, 10, "Bird", .bird
    OptionButton 203, 80, 42, 10, "Clipboard", .clipboard
    Button 171, 97, 80, 14, "Hide &Picture", .hide
    Button 190, 221, 40, 14, "&Error", .berror
  End Dialog
  dim td as testdlg' dialog box testdlg declared as variable
    do' loop handles when clicking Cancel or OK
    select case dialog(td)
    case -1
        if msgbox("Dialog terminated by OK. Restart?", 36,
"TestDlg") = 7 then exit do
    case 0
        if msgbox("Dialog terminated by Cancel. Restart?", 36,
"TestDlg") = 7 then exit do
    case errorReturn
        if msgbox(error$(evalue) + "on line" + cstr(eline) + ".
Restart?", 36, "TestDlg") = 7 then exit do
    case else
        if msgbox("Dialog terminated by a button other than OK
or Cancel. Restart?", 36, "TestDlg") = 7 then exit do
    end select
  loop' end of loop
  End Sub
```

Chapter 5

Hummingbird Basic Language Reference

This chapter provides a quick reference to the statements and functions available in Hummingbird Basic. The functions and statements are separated into categories by type. Each function and statement is accompanied by a short description.

For information about the specific syntax and usage of a statement or function, see HostExplorer Programming Help.

Hummingbird Basic Statements and Functions

Arrays

Function	Description
`Erase`	Re-initialize contents of an array.
`LBound`	Return the lower bound of an array's dimension.
`ReDim`	Declare dynamic arrays and reallocate memory.
`UBound`	Return the upper bound of an array's dimension.

Compiler Directives

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

Control Flow

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

Dates and Times

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

Declarations

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

Defining Dialog Boxes

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

Running Dialog Boxes

Function	Description
`Dialog Function`	Display a dialog box, and return the button pressed.
`Dialog Statement`	Display a dialog box.
`DlgControlId`	Return the numeric ID of a dialog control.
`DlgEnable Function`	Return whether a dialog control is enabled or disabled.
`DlgEnable Statement`	Enable or disable a dialog control.
`DlgEnd`	Close the active dialog box.
`DlgFocus Function`	Return the ID of the dialog control having input focus.
`DlgFocus Statement`	Set focus to a dialog control.
`DlgListBoxArray Function`	Return the contents of a list box or combo box.
`DlgListBoxArray Statement`	Set the 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`	Return whether a control is visible or hidden.
`DlgVisible Statement`	Show or hide a dialog control.

Dynamic Data Exchange (DDE)

Function	Description
`DDEAppReturnCode`	Return a code from an application on a DDE channel.
`DDEExecute`	Send 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`	Retrun data from an application on a DDE channel.
`DDETerminate`	Close a DDE channel.

Environment Control

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

Error-Handling Functions

Function	Description
`Assert`	Trigger an error if a condition is false.
`Erl`	Return the line number where a runtime error occurred.
`Err Function`	Return a runtime error code.
`Err Statement`	Set the runtime error code.
`Error`	Generate an error condition.
`Error Function`	Return a string representing an error.
`On Error`	Control runtime error-handling.
`Resume`	End an error-handling procedure.

Disk and Directory Control

Function	Description
`ChDir`	Change the default directory for a drive.
`ChDrive`	Change the default drive.
`CurDir`	Return the current directory for a drive.
`Dir`	Return a file name that matches a pattern.
`MkDir`	Make a directory on a disk.
`RmDir`	Remove a directory from a disk.

File Control

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

File Input/Output

Function	Description
`Close`	Close a file.
`Eof`	Check for end of file.
`FreeFile`	Return the next unused file number.
`Get`	Read bytes from a file.
`Input Statement`	Read data from a file or from the keyboard.
`Line Input`	Read a line from a sequential file.
`Loc`	Return the 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.
`Open`	Open a disk file or device for I/O.
`Print`	Print data to a file or to the screen.

Function	Description
`Put`	Write data to an open file.
`Reset`	Close all open disk files.
`Seek Function`	Return the current position for a file.
`Seek Statement`	Set the current position for a file.
`Spc`	Send the given number of spaces for output.
`Tab`	Move the print position to the given column.
`Unlock`	Control access to some or all of an open file by other processes.
`Width`	Set the output-line width for an open file.
`Write`	Write data to a sequential file.

Financial Functions

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

Numeric Functions

Function	Description
`Abs`	Return the absolute value of a number.
`Exp`	Return the value of *e* raised to a power.
`Int`	Return the integer part of a number.
`Fix`	Return the integer part of a number.
`IsNumeric`	Determine whether a value is a legal number.
`Log`	Return the natural logarithm of a value.
`Rnd`	Return a random number.
`Sgn`	Return a value indicating the sign of a number.
`Sqr`	Return the square root of a number.

Trigonometric Functions

Function	Description
`Atn`	Return the arc tangent of a number.
`Cos`	Return the cosine of an angle.
`Sin`	Return the sine of an angle.
`Tan`	Return the tangent of an angle.

Objects

Function	Description
`Class List`	List of available classes.
`Clipboard`	Access the Windows Clipboard.
`CreateObject`	Create an OLE automation object.
`GetObject`	Retrieve an OLE object from a file, or get the active OLE object for an OLE class.
`Is`	Determine whether two object variables refer to the same object.
`Me`	Get the current object.
`New`	Allocate and initialize a new OLE object.
`Nothing`	Set an object variable to not refer to an object.
`Object`	Declare an OLE automation object.
`Typeof`	Check the class of an object.
`With`	Execute statements on an object or a user-defined type.

Screen Input/Output

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

String Functions

Function	Description
`GetField`	Return a substring from a delimited source string.
`Hex`	Return the hexadecimal representation of a number as a string.
`InStr`	Return the position of one string within another.
`LCase`	Convert a string to lower case.
`Left`	Return the left portion of a string.
`Len`	Return the length of a string or size of a variable.
`Like Operator`	Compare a string against a pattern.
`LTrim`	Remove leading spaces from a string.
`Mid Function`	Return a portion of a string.
`Mid Statement`	Replace a portion of a string with another string.
`Oct`	Return the octal representation of a number as a string.
`Right`	Return the right portion of a string.
`RTrim`	Remove trailing spaces from a string.
`SetField`	Replace a substring within a delimited target string.
`Space`	Return a string of spaces.
`Str`	Return the string representation of a number.
`StrComp`	Compare two strings.
`String`	Return a string consisting of a repeated character.
`Trim`	Remove leading and trailing spaces from a string.
`UCase`	Convert a string to uppercase.

String Conversions

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

Variants

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

Calling External Functions in a .dll

The Hummingbird Basic language contains an extensive set of API (Application Programming Interface) calls that can be used to customize some of the applications included in the Hummingbird product line.

API refers to a set of specialized functions that allow you to communicate directly with the Windows application layer.

The following applications contain custom API function calls:

- FTP
- HostExplorer

For information about using TN3270 or TN5250 API function calls, refer to HostExplorer Programming Help, located in the HostExplorer folder.

Sample Script: Calling External Functions in a .dll

The following sample script demonstrates how to declare and call a function from an external `.dll`. The `.dll` in this example is called `user.dll` and it contains a function called `GetTickCount&`.

```
Declare Sub MessageBox LIB "user32" Alias "MessageBoxA" (ByVal
h%, ByVal t$, ByVal c$, ByVal u%)
Declare Function GetTickCount& LIB "kernel32.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 displays a message box

Sub Say(what$)
    Dim min, sec, hrs
        sec = GetTickCount () /1000
        min = sec / 60 : sec = sec mod 60
        hrs = min / 60 : min = min mod 60
    Dim eTime as variant
```

```
        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
```

Using Dynamic Data Exchange

Dynamic Data Exchange allows two applications to communicate and to exchange data. One of these applications can be your Basic program. To *talk* to another application and send it data, you need to open a connection with the application (called a DDE channel) using the statement `DDEInitiate`. However, if you have OLE automation available, we recommend you use it instead of DDE, since OLE is used more.

Note: 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
- 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 file name to get or send data to, although there are some reserved DDE topic names, such as `System`. Refer to the documentation for the application to get a list of topic names.

After opening a channel to the 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 with the DDETerminate function.

> **Note:** There are a limited number of channels available for you to use at one time. Close channels as soon as you are finished using them. You can use up to 10 channels.

The other DDE command available in Hummingbird Basic is DDEAppReturnCode. This command is used for error checking. After getting or sending text, or executing a command, 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.

DDE Sample Script

The following sample script opens the Microsoft Word application and uses DDERequest to obtain a list of available topics:

```
Sub main
   Dim channel as Integer
   Dim appname as String
   Dim topic as String
   Dim path as string
   Dim msgtext as string
   Dim ttext as string

   appname="Excel"
   topic="Sheet1"
   path="d:\office97\office\"
   channel = -1
   ttext = "Hello, world"
   x=Shell(path & appname & ".EXE")
   channel = DDEInitiate(appname, topic)
   If channel= -1 then
      msgtext="Excel not found -- please place on your path."
   Else
      On Error Resume Next
      DDEPoke channel, "R3C2", ttext
      DDEExecute channel, "[SELECT(" + Chr$(34) + "R4C4" +
Chr$(34) + ")]"
      DDETerminate channel
      If Err<>0 then
         msgtext="DDE Access failed."
      End If
   End If
End sub
```

Appendix A

Technical Support and Accessibility

Accessibility

Hummingbird products are accessible to all users. Wherever possible, our software was developed using Microsoft Windows interface standards and contains a comprehensive set of accessibility features.

Keyboard shortcuts All menus have an associated keyboard shortcut. To access any menu, press Alt and the underlined letter in the menu name as it appears on the interface. For example, to access the File menu in any Hummingbird application, press Alt + F.

Once you have opened a menu, you can access a menu item by pressing the underlined letter in the menu item name, or you can use the arrow keys to navigate the menu list. For menu items with an associated keyboard shortcut, the shortcut is listed on the menu to the right of the item.

Directional arrows Use the directional arrows on the keyboard to navigate through menu items or to scroll vertically and horizontally. You can also use the directional arrows to navigate through multiple options. For example, if you have a series of radio buttons, you can use the arrow keys to navigate the possible selections.

Tab key sequence To navigate through a dialog box, press the Tab key. Selected items appear with a dotted border. You can also press Shift + Tab to go back to a previous selection within the dialog box.

Spacebar Press the Spacebar to toggle check boxes on and off or to select buttons in a dialog box.

Esc Press the Esc key to close a dialog box without implementing any new settings.

Enter Press the Enter key to select the highlighted item or to close a dialog box with the new settings. You can also press the Enter key to close all About boxes.

ToolTips ToolTips appear for all functional icons. This feature lets users use Screen Reviewers to make interface information available through synthesized speech or through a refreshable Braille display.

Microsoft Accessibility Options

Microsoft Windows environments contain accessibility options that let you change how you interact with the software. This feature can add sound, increase the magnification, and create sticky keys.

To access the Microsoft Windows Accessibility options, open Control Panel and click Accessibility.

If you installed the Microsoft Accessibility components for your Windows system, you can also find other Accessibility tools on the Start menu under Programs/Accessories/Accessibility.

To add the Accessibility components:

1 Navigate to Control Panel and Open Add/Remove Programs.

2 On the Windows Setup tab, select the Accessibility Options check box and click Apply.

3 Click OK.

Technical Support

You can contact the Hummingbird Technical Support Department Monday to Friday between 8:00 a.m. and 8:00 p.m. Eastern Time.

Hummingbird Ltd.
1 Sparks Avenue, North York, Ontario, Canada M2H 2W1

	Canada and the USA	International
Technical Support:	1-800-486-0095	+1-416-496-2200
General Enquiry:	1-877-FLY-HUMM	+1-416-496-2200
Main:	+1-416-496-2200	
Fax:	+1-416-496-2207	
E-mail:	support@hummingbird.com	
FTP:	ftp.hummingbird.com	
Online Request Form:	www.hummingbird.com/support/nc/request.html	
Web Site:	www.hummingbird.com/about/contact.html	

Glossary

Application Programming Interface (API)
A set of routines, protocols, and tools that programmers use to build software applications. Most operating systems have an API which programmers use to write applications that are consistent with that operating environment. APIs ensure that all programs using that API have a similar interface. This makes it easier for users to learn new programs.

Breakpoint
A location in a program at which execution is halted so that a programmer can examine the status of the program, the contents of variables, and so on. A breakpoint is set and cleared within a debugger, and is usually implemented by inserting at that point some kind of jump, call, or trap instruction that transfers control to the debugger.

Compiler
A program that translate all of the source code of a program written in a high-level language into object code prior to execution of the program.

Control
A control statement determines the results of your script. It also directs the flow of logic during the execution of commands.

DDE
Dynamic Data Exchange. DDE allows communication and data exchange between two applications through connections called DDE channels.

Debug
To detect, locate, and correct logical or syntactical errors in a program, or malfunctions in hardware.

Dialog box
In a graphical user interface, a special window displayed by the system or application to solicit a response from the user.

Emulation
The process of a computer, device, or program imitating the function of another computer, device, or program. Terminal emulation drivers included in communications software enable a PC to emulate a terminal type. This makes it possible for a user to log on to a mainframe.

Error
A value or condition that is not consistent with the true, specified, or expected condition. In computers, an error results when an event does not occur as expected, or when impossible or illegal maneuvers are attempted. In data communications, an error occurs when there is a discrepancy between the transmitted and received data.

Error-Handling
A special set of instructions that enable your script to trap errors that may occur while your script is running.

Expression
A collection of terms that perform a mathematical or a logical operation. The terms are either variables or functions that are combined with an operator to evaluate a result. There are several types of operators.

Function
A construct which, when executed, calculates and returns a value as determined by its arguments. Hummingbird Basic contains a variety of built-in functions you can use in your scripts. You can also write your own functions.

Interpreter
A program that translates, and then executes, each statement in a program written in an interpreted language.

Logic Error
Occurs because of incorrect coding that causes unexpected results (such as infinite loops or incorrect values returned by functions) during the execution of the script. These types of errors generally cause unexpected results during the execution of your script.

Object Linking and Embedding (OLE)
A compound document standard that allows you to create objects with one application and link or embed the objects in a second application. Embedded objects retain their original format and links.

Windows and Macintosh operating systems support OLE.

Operator
A symbol or other character indicating an operation that acts on one or more elements.

Procedure

A procedure contains a set of variables and statements that you defined for the script. There are two different types of procedures in Hummingbird Basic: functions and subprocedures. A Hummingbird Basic script can contain one main subprocedure. When the script is run, the main subprocedure is executed first.

Runtime Error

Can be caused by an unforeseen action taken by the user, a coding error, or the data your script is using (the script attempts to read a file containing no data). Runtime errors are handled through a set of error-handling functions and statements.

Statement

An instruction written in a high-level programming language that defines how a task in the script is carried out. It provides the conditional logic or looping for a procedure. It also defines the state of a dialog box, such as its display and configuration.

Syntax Error

Usually the result of spelling a statement or a function incorrectly. It can also be the result of using either a statement or function incorrectly. To help you fix syntax errors, the Hummingbird Script Editor highlights language syntax errors in red after a script is compiled.

Trappable error

See Error.

Variable

Placeholders for values that are declared, named, and assigned a data type.

Index

Hummingbird Ltd. — Corporate Headquarters
1 Sparks Avenue • Toronto, Ontario • M2H 2W1 • Canada
Toll Free Canada/U.S.A: +1 877 FLY HUMM (359 4866)
Tel: +1 416 496 2200 • **Fax:** +1 416 496 2207 • **Email:** getinfo@hummingbird.com

For more information, visit **www.hummingbird.com**

8157-6M
Printed in Canada.

Hummingbird™