Overview About KBasic

The following chapter has been used from Wikipedia entry about BASIC and is licensed under the GNU Free Documentation License.

Table of Contents

About The Author

Bernd Noetscher is a software developer and the main developer of the KBasic programming language. In his spare time he goes dancing, reads many books, and plays piano. He is also interested in theatre and cinema. His private website is www.berndnoetscher.de

KBasic is a multi-purpose computer programming language that is suitable for most development needs. The language is designed with Rapid Application Development in mind, providing several tools to shorten development time. The following text introduces language fundamentals and covers a variety of the KBasic Framework.

KBasic is the next generation of BASIC programming languages. As a language, KBasic has the following traits:

Object-Oriented

Visual Basic 6 included limited support for object-oriented design. Encapsulation was supported with Public and Private data types in Classes, as well as special accessor/mutator pairs called Properties. Polymorphism received crude support through the Implements keyword, which would require that one class implement all of the methods in another, skeleton class. Inheritance was neglected completely.

As of KBasic, all of this has changed. KBasic includes full-blown support for object-oriented concepts, including simple inheritance.Everything inherits from the Object base class.

Event-Driven

Events are recognized because they use a certain naming convention (ObjectName_EventName).

KBasic Framework

As the name implies, KBasic runs on top of Trolltech's crossplatform Qt framework, meaning the language has full access to all of the supporting classes in the C++ Qt framework. It's also possible to run KBasic programs under either Windows, Mac OS X or even Linux.

The Integrated Development Environment (IDE) - To simplify application development.

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software that assists computer programmers to develop software.

IDE Contents

The IDE consists of several sections, or tools, that the developer uses while programming. As you view the IDE for a new project you generally have three sections in view:

- The Toolbox on the right
- The Project Window on the right
- The Code / Design view in the middle

Toolbox

The Toolbox is a palette of developer objects, or controls, that are placed on forms or web pages, and then code is added to allow the user to interact with them. An example would be TextBox, CommandButton and ListBox controls. With these three controls added to a Form object the developer could write code that would take text, input by the application user, and added to the ListBox after the button was clicked. Okay, not rocket science, but useful.

Project Window

This is a section that is used to view and modify the contents of the project. A KBasic Project will generally have a Form object with a code page, references to System components and possibly other modules with special code that is used by the application.

Properties Windows

The properties windows shows all the control (like textbox) properties to be change at design time. Most of this properties can be change at run time with some code, but basically most of this properties change the way the control is display on your application.

Code / Design view

This is where the magic takes place. Forms are designed graphically. In other words, the developer has a form on the screen that can be sized and modified to look the way it will be displayed to the

application users. Controls are added to the form from the Toolbox, the color and caption can be changed along with many other items.

This center window of the IDE is also where developers write the code that makes everything in the application work. The code is written in modules, or files, that are either connected to an object (Forms) or called specifically when needed.

Review

The KBasic IDE is a complex and exciting work space for developers, providing many enhancements and upgrades from the days of VB6 or the introduction of C++. This section only serves as a mere introduction to this multi-functional interface.

Getting Started - Making your first application.

In this manual, however, we will be covering the IDE. Programming a form (window) by hand can be challenging if you are not familliar with the language, so starting with a good form designer will help a lot.

Okay, it's now time for your first program!

Start up a KBasic IDE and create a new project (select with one form and one module).

Create a new form, if KBasic did not create one for you automatically and change to the source code view. You should see a plain text editor.

Enter the follwing code:

```
Sub Form_OnOpen()
  Application.MsgBox("Hello World!", ":-)")
End Sub
```

This creates your classic "Hello World!" program.

Press F5 or go to the run menu and select Start to start the program. You should see an alert box that says "Hello World!" and then the main window (with nothing on it) should open. Click the "X" in the title bar like you would to quit any program!

Variables

The entities used to store various types of data. In programing a variable is simply a place to store data. A variable has a name and a data type. In KBasic, a variable is declared using the Dim (short for Dimension) statement. Here is the syntax:

```
Dim varName As varType
```

varName is the name of your Variable. varType is the data type of the variable. Types include String, Integer, Double, Boolean, etc.

For example, to declare an Integer named MyInt use:

```
Dim MyInt As Integer
```

Data Types

Data Types define the type of data that a variable can store. Some variables store numbers, others store names. The basic types that can be used in KBasic are:

Byte

8 bits, stores integer values from 0 to 255.

```
Dim bytMyVariable As Byte
```

Short

Meaning short integer. 16 bits (2 bytes), stores integer values from -32,768 to 32,767.

```
Dim shrtMyVariable As Short
```

Integer

32 bits (4 bytes), stores integer values from -2,147,483,648 to 2,147,483,647.

```
Dim intMyVariable As Integer
```

Long

Meaning long integer. 64 bits (8 bytes), stores integer values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

```
Dim lngMyVariable As Long
```

Single

Meaning single-precision floating point. 32 bits (4 bytes), stores floating-point values from -3.40282347e38 to 3.40282347e38. Cannot store an exact zero due to the required normalized form. Smallest positive value greater than zero is 1.401298e-45. Biggest negative value less than zero is -1.401298e-45.

```
Dim sglMyVariable As Single
```

Double

Meaning double-precision floating point. 64 bits (8 bytes), stores floating-point values from -1.7976931348623157e308 to 1.7976931348623157e308. Cannot store an exact zero due to the required normalized form. Smallest positive value greater than zero is 4.94065645841247e-324. Biggest negative value less than zero is -4.94065645841247e-324.

```
Dim dblMyVariable As Double
```

Boolean

Requires 1 bit (1 byte used), stores boolean true/false values.

```
Dim boolMyVariable As Boolean
```

String

Stores any series of characters. e.g. word, phrase, sentence, paragraph, etc.

```
Dim strMyVariable As String
```

Object

32 bits (4 bytes), stores a reference to a KBasic object.

```
Dim objMyObject As Object
```

VB6 support datatypes like Variant, Currency

Using Variables

Assigning Values

A value is the data contained in a variable. To assign a value to a variable that is already declared, use an equal sign.

Examples

For string variables, use double quotes around the value:

```
strMyVariable = "The String"
```

For all others, remove the quotes:

```
bytMyVariable = 1
sbytMyVariable = -2
intMyVariable = 100
lngMyVariable = 1000
sngMyVariable = 1.234
dblMyVariable = 1.567
decMyVariable = 1234567.89D
boolMyVariable = True
objMyObject = New Object
```

To assign a variable the value of another variable, simply replace the value on the right side of the equal sign with the name of the variable that holds the desired data.

You can also assign a value to a variable in the declaration itself.

Dim myVariable As String = "StringValue"

Important: KBasic always assigns the value of the right variable to the left variable. The variable on the left takes the value of the right variable. The variable on the right does not change.

Constants

Constants are like variables that don't change. They take the place of values that you would not like to type over and over. Constants are declared using the keyword "Const". Their values are defined in their declaration - they also use data types. Here is the syntax:

```
Const cnstMyConstant As String = "The very long string"
```

Here is an example:

```
Const cnstPi As Single = 3.14159265
```

Arrays

A variable that stores several data of the same data type.

An array is simply a variable that can store more then one piece of data. The data is stored in a list. If you declare an integer, then that variable can only store one integer. An array of integers can store many integers. Each one is given its own number. For example, this line of code:

```
Dim MyArray(5) As Integer
```

will give an array like this

Index	Data
00	0
01	0
02	0
03	0
04	0
05	0

Branch statements

The various conditional statements in the language for selective code execution.

If...Else...ElseIf Statement

If/Else Statments are used to conditionally execute code based on condition provided. If the condition provided in the If statement evaluates to true, the code in the block is executed. Otherwise, execution would proceed to the optional ElseIf statements, or the Else statement. ElseIf and Else are not required parts of an If statement.

An example of the If/Else/ElseIf branch statement is:

```
'The following variable declarations are for the following example only. They
are not needed in real life.
Dim x As Integer
Dim y As Integer

If x = y Then
  'Whatever will happen if x = y
ElseIf x < y Then
  'Whatever will happen if x < y
Else
  'Whatever will happen if x isn't = to y and x isn't < to y
End If
```

Select Case Statement

Either strings or numbers can be used for a Select Case statement.

Select Case statements are usually used to avoid long chains of If/ElseIf/…/ElseIf/Else statements.

An example of a Select Case branch statement using an integer is:

```
Dim nCPU as Integer
Select Case nCPU
    Case 0
        'No CPU!
    Case 1
        'Single CPU
    Case 2
        'Dual CPU machine
    Case 4
        'Quad CPU machine
    Case 3, 5 To 8
        '3, 5, 6, 7, 8 CPU's
    Case Else
        'Something more than 8
End Select
```

Boolean Operators

Boolean operators in KBasic now accomodate for short circuit boolean evaluation, most other languages always apply short citcuit boolean evaluation by default, consider the following boolean statement:

functionA() and functionB()

With this statement, when short-circuit boolean evaluation is used, the second function will only be called if the first function returns true, this is because if functionA returns false it becomes irrelevant to the outcome of the statement whether functionB returns true or not.

However, when no short-circuit boolean evaluation is used, both of the functions will be called irrespective of whether the first part of the statement returns true or false.

Something to note with short-circuit boolean evaluation is that the order of the parameters can become important when it is used.

Due to previous versions of VB6 not having short circuit boolean evaluation, KBasic comes with backward compatibility and add two new boolean logic identifiers which support short-circuit boolean evaluation, so in addition to the standard boolean operators:

```
NOT
AND
OR
XOR
```

There are also two new operators which function using short-circuit boolean evaluation, and they are:

```
ANDALSO
ORELSE
```

Loop statements

The various loop statements in the language for repetitive code execution.

For example:

- Do…Until Loop
- While…Do Loop
- For Loop
- For Each Loop

Operators

Programming languages have a set of operators that perform arithmetical operations, and others such as Boolean operations on truth values, and string operators manipulating strings of text. Computers are mathematical devices, but compilers and interpreters require a full syntactic theory of all operations in order to parse formulae involving any combinations correctly. In particular they depend on operator precedence rules, on order of operations, that are tacitly assumed in mathematical writing.

Conventionally, the computing usage of operator also goes beyond the mathematical usage (for functions). In KBasic, New and AddressOfare operators. Y

So operators are special symbols that are used to represent for example simple computations like addition and multiplication. Most of the operators in KBasic do exactly what you would expect them to do, because they are common mathematical symbols. For example, the operator for adding two integers is +.

Assignation and comparison operators

Assignation

The "=" operator is used for assignation. The operator also serves as a comparison operator (see Comparison).

- To set values:

```
x = 7     ' x is now seven; in math terms we could say "let x = 7"
x = -1294
x = "example"
```

You can use variables in the equal operator, as well.

```
Dim x As Integer
x = 4  ' Anywhere we use x, 4 will be used.
```

Comparison

The "=" operator is used for comparison. The operator also serves as a assignation operator (see Assignation).

- To compare values:

```
If 4 = 9 Then       ' This code will never happen:
```

```
    End  ' Exit the program.
End If
If 1234 = 1234 Then ' This code will always be run after the check:
    MsgBox("Wow!  1234 is the same as 1234.")
      ' Create a box in the center of the screen.
End If
```

You can use variables in the equal operator, as well.

```
If x = 4 Then
    MsgBox("x is four.")
End If
```

Let's try a slightly more advanced operation.

```
MsgBox("Seven equals two is " & (7 = 2) & ".")
' The parentheses are used because otherwise, by order of operations
' (equals is processed last), it would be comparing "Seven equals two is 7" and
"2.".
' Note here that the & operator appends to the string.  We will talk about this
later.
'
'  The result of this should be a message box popping up saying "Seven equals
two is
'   False."   This is because (7 = 2) will return False anywhere you put it.  In
the
'   same sense, (7 = 7) will return True:
MsgBox("Seven equals seven is " & (7 = 7) & ".")
```

You will get an error if you try to assign a constant or a literal a value, such as 7 = 2. You can compare 7 and 2, but the answer will always be False.

In the case of two equal operators appearing in a statement, such as

```
Dim x As Boolean
x = 2 = 7
```

The second equal operator will be processed first, comparing 2 and 7, giving a False. Then the first equal operator will be processed, assigning False to x.

Arithmetic operators

Describes the operators that work on numerical values, such as the '+' in addition.

KBasic provides a basic set of operators to calculate simple arithmetic.

```
+   Addition
-   Subtraction
*   Multiplication
/   Division
\   Integer division
Mod Remainder Division
^   Exponentiation
&   String concatenation

7 + 2     produces 9
7 - 2     produces 5
7 * 2     produces 14
7 / 2     produces 3.5
```

```
7 \ 2     produces 3
7 Mod 2   produces 1
7 ^ 2     produces 49
"7" & "7" produces "77"
```

Let's look at a short example of arithmetic operations before we jump into the operators themselves.

In this example we will also be using some basic variables. The Dim operator creates the variable.

```
Dim Commission As Single
Dim Sales As Single
Sales = 3142.51
Commission = 0.3 * Sales  ' Calculate 30% commission.
```

First, we set the total sales to 3142.51.

The * operator calculates multiplication, so the last line is equivelant to Commission = 0.3 * Sales. This means that our second step is multiplying 0.3 and Sales. Sales is 3142.51, so our result should be the product of 0.3 and 3142.51.

Why the funny symbols?

With the exception of addition and subtraction, the symbols used are different to the ones used in real life. This is simply because the other symbols are not available on a standard keyboard.

Addition

This adds two numbers together, and is denoted by the "+" symbol. If strings are involved it may also do String concatenation. Examples:

```
Dim x As Integer
x = 7 + 2     ' Results in 9.
x = 25 + -4   ' Results in 21.
Dim StringA As String
StringA = "A string" + "Another string" ' Results in "A stringAnother string"
```

Subtraction

This subtracts two numbers, and is denoted by the "-" symbol. Examples:

```
Dim x As Integer
x = 7 - 2    ' Results in 5.
x = 25 - -4  ' Results in 29.
```

Multiplication

This multiplies two numbers, and is denoted by the "*" symbol. Examples:

```
Dim x As Integer
x = 7 * 2   ' Results in 14.
x = 25 * -4 ' Results in -100.
```

Division

There are more types of division than the one denoted by the "/" symbol. There is also integer division and remainder division.

- Division

This is the most commonly used form of division and is denoted by the "/" operator. Examples:

Dim x As Single ' (note that we must use the Single class to have decimals) x = 7 / 2 ' Results in 3.5. x = 25 / 4 ' Results in 6.25.

- Integer division

This divides two numbers, and gives the result without the remainder if the quotient is a decimal. Examples:

Dim x As Integer x = 7 \ 2 ' Results in 3. x = 25 \ 4 ' Results in 6.

- Remainder Division

This divides two numbers, and gives the result's remainder if the quotient is a decimal. This is denoted by the operator "Mod." Examples:

Dim x As Integer x = 7 Mod 2 ' Results in 1. x = 25 Mod 4 ' Results in 1.

Exponentiation

This is raising a number to a power. For example:

```
Dim x As Integer
x = 7 ^ 2   ' Results in 49.
```

This results in the number 49 being assigned to the variable x. It can also be used to calculate the square root of a number. The square root of a number is the number raised to the power of 0.5.

```
Dim x As Single
x = 7 ^ 0.5 ' Results in a number around 2.645.
```

Note: It is necessary to ensure that the variables be correctly declared to get the desired results. The following example works, but will produce the wrong result. This is because the Integer class does not allow decimal places (just like mathematical integers.)

```
Dim x As Integer
x = 7 ^ 0.5 ' Results in 3.
```

Since x is declared as an Integer type, the value square root, a real number, is stored incorrectly.

Any nth root of number is the can be calculated by raising the number to the power of 1 / n:

```
Dim x As Single
Dim n As Single
n = 7
x = 2 ^ (1 / n)
```

String operators

Describes the operators used for string and character manipulation, such as the '&' and '+' operators.

String concatenation

The "&" operator joins two strings together. Example:

Dim String1 As String = "123" Dim String2 As String = "456" Dim String3 As String String3 = String1 & String2 ' Results in "123456".

This will result in String3 being equal to "123456"

The "+" operator may be used in place of "&". However, it is not recommended.

Logical operators

Describes the operators used in boolean logic, such as the 'And' and 'Or' operators.

Not

The Not operator returns True when the condition is False. Otherwise, it returns False. Example,

```
If Not (1 = 2) Then
   MsgBox("(1 = 2) is False. So Not False is True")
End If
```

Truth Table Condition	Not Condition
True	False
False	True

And

The And operator returns True when the conditions of the left and right are True. Otherwise, it returns False. Both conditions are evaluated before the result is returned. Example,

```
If (1 = 1) And (2 = 2) Then
   MsgBox("(1 = 1) is True. (2 = 2) is True. So True And True is True")
End If
```

Truth Table Condition1	Condition2	Condition1 And Condition2
True	True	True
True	False	False
False	True	False
False	False	False

AndAlso

The AndAlso operator returns False when the condition on the left hand side is False. Else, it returns True when the conditions of the left and right are True. Otherwise, it returns False. The condition on the right hand side is never evaluated when that on the left hand side is False. This is called short-circuited logic.

Truth Table Condition1	Condition2	Condition1 AndAlso Condition2
True	True	True
True	False	False
False	-	False

Or

The Or operator returns True when the condition on either side is True. Otherwise, it returns False. Both conditions are evaluated before the result is returned.

Truth Table Condition1	Condition2	Condition1 Or Condition2
True	True	True
True	False	True
False	True	True
False	False	False

OrElse

The OrElse operator returns True when the condition on the left hand side is True. Else, if the condition on the right hand side is True, it returns True. Otherwise, it returns False. The condition on the right hand side is never evaluated when that on the left hand side is True. This is called short-circuited logic.

Truth Table Condition1	Condition2	Condition1 OrElse Condition2
True	-	True
False	True	True
False	False	False

Xor

The Xor operator returns True when the condition on the left hand side or right hand side is True, but not when both are True. Xor means "Exclusive OR".

Truth Table Condition1	Condition2	Condition1 Xor Condition2
True	True	False
True	False	True
False	True	True
False	False	False

GUI Programming

Like its predecessor, KBasic excels in creating graphical user interfaces by selecting items from the toolbox and adding to a particular form. While working with forms, you can use the toolbox to drag different controls to the form you are designing, resize them and relocate them using the mouse, and set the control's properties in a corresponding properties window to quickly develop the user interface. Events handlers for each control's most common event can be quickly created by double-clicking on the control to create a new event handler and be send to that event handler in the code window.

Some Visible controls

TextBoxes

A TextBox is used to get and display text. A TextBox can only display text in one font, so it can't be used for word processing. A TextBox will only display text on a single line. For example:

Labels

Labels are used to display text. Unlike the textbox, it is not meant to accept input from the user. Often a Label is used to describe another control, and is often used as a prompt for a Textbox.

CommandButtons

CommandButtons are controls that are usually raised that the user most often can click on to perform some action defined by the programmer. Once the programmer has added the button control to a form, he can define an event handler to perform an action when the button is clicked.

CheckBoxes

A check box indicates a two-way choice or state (true/false) which can be edited by the user. Check boxes are shown on the screen as a square box that can contain white space (for false) or a check mark (for true). Adjacent to the check box is normally shown a caption describing the meaning of the check box. Inverting the state of a check box is done by clicking the mouse on the button, or the caption. KBasic allows the programmer to set the caption.

RadioButtons

A radio button allows the user to choose exactly one of a predefined set of options. Radio buttons are arranged in groups of two or more and displayed on screen a list of circular holes that can contain white space (for unselected) or a dot (for selected). Each radio button can show a caption describing the choice that this radio button represents.

Editor

Can easily save Richtext files with Colours and Fonts.

Some other controls not visible at design time

Some of the controls you can add while designing your form do not actually appear on the form, but you can still use the toolbox to add them to the form, where they are keep in a try below the form for easy reference. For example:

Timer Control

A Timer Control is a control that will execute code at intervals. It is not visible at runtime. The interval can be set in the properties in the measurement of milliseconds. The timer control will continue to repeat its code at your interval until the control is disabled.