Keywords

A keyword is a predefined identifier, which has a special meaning for KBasic and which meaning cannot be changed. Some of them are provided for VB6 and QBasic backward compatibility. The following list contains all KBasic keywords.

Table of Contents

Press F1 in KBasic, if you want to jump to one of the following help topics.

$Dynamic , #Else , $End , #ExternalSource , #If , #Region , $Static (Outdated) , Absolute , Abstract , AddressOf , Alias , Ansi , As , Assembly , Auto , Base , Binary , ByRef , ByVal , Call , CallByName , Case , Catch , Chain , Choose , Class , Class_Initialize , Class_Terminate , COM , Common , Compare , Connect , Const , Constructor , Data , Database , Decimal , Declare , Def , Default , DefBool , DefByte , DefCur , DefDate , DefDbl , DefInt , DefLng , DefObj , DefSng , DefStr , DefVar , Delegate , Destructor , Dim , DirectCast , Disconnect , Do , Each , Else , ElseIf , Empty , End , EndIf , Enum , Erase , Event , Exit , Explicit , Finally , For , Friend , Function , Global , GoSub , GoTo , Handles , If , IIf , Implements , Imports , In , Inherits , Interface , Is , Iterate , KBasic , Key , LBound , Let , Lib , Like , Loop , LSet , Me , Mid , Module , MustInherit , MustOverride , MyBase , MyClass , NameSpace , New , Next , Nothing , NotInheritable , NotOverridable , Null , Off , OldBasic , On , Option , Optional , Overloads , Overriddable , Overrides , ParamArray , Parent , Pen , Play , Preserve , Private , Property , Protected , Public , Range , Read , ReadOnly , ReDim , Rem , /** , /* , */ , ' , Repeat , Restore , Resume , Return , RSet , Run , Select , Set , Shadows , Shared , Signal , SizeOf , Slot , Static , Step , Stop , STRIG , Structure , Sub , Swap , Switch , SynClock , System , Text , Then , Throw , Throws , Timer , To , TROFF , TRON , Try , Type , TypeDef , TypeOf , UBound , UniCode , Until , VARPTR , VARPTR$, VARSEG , VeryOldBasic , Wait , Wend , While , With , WithEvents , WriteOnly

The following list is recommended for new application development.

Abstract , AddressOf , Alias , As , ByRef , ByVal , Case , Catch , Choose , Class , Connect , Const , Constructor , Destructor , Dim , Disconnect , Do , Each , Else , ElseIf , End , EndIf , Enum , Erase , Exit , Finally , For , Function , GoTo , If , IIf , In , Inherits , Is , Iterate , LBound , Lib , Loop , Me , Mid , Module , New , Next , Null , Parent , Preserve , Private , Property , Protected , Public , ReDim , Rem , /** , /* , */ , ' , Return , Select , Signal , SizeOf , Slot , Static , Step , Stop , Sub, Switch , Then , Throw , Throws , To , Try , Type , TypeOf, UBound , With

The following list contains the keywords, which are reserved and have no functionality yet.

#Else , #ExternalSource , #If , #Region , Absolute , Ansi , Assembly , Auto , CallByName , Chain , COM , Database , Decimal , Default , Delegate , DirectCast , Echo , Event , Friend , Handles , Implements , Imports , Interface , Key , MustInherit , MustOverride , MyBase , MyClass , NameSpace , NotInheritable , NotOverridable , Overloads , Overriddable , Overrides , Pen , Play , ReadOnly , Repeat , Run , Shadows , Shared , Structure , Swap , SynClock , TROFF , TRON , TypeDef , UniCode , VARPTR , VARPTR$, VARSEG , Wait , WithEvents , WriteOnly

Constants

VB6 backward support constants. VB6! QB!

kbAbort

kbAbortRetryIgnore

kbArchive

kbArray

kbBack

kbBoolean

kbByte

kbCancel

kbCr

kbCrLf

kbCritical

kbCurrency

kbDate

kbDefaultButton1

kbDefaultButton2

kbDefaultButton3

kbDirectory

kbDouble

kbEmpty

kbError

kbExclamation

kbFriday

kbHidden

kbIgnore

kbInformation

kbInteger

kbLf

kbLong

kbMonday

kbNewLine

kbNo

kbNormal

kbNull

kbNullChar

kbNullString

kbOK

kbOKCancel

kbOKOnly

kbObject

kbQuestion

kbReadOnly

kbRetry

kbRetryCancel

kbSaturday

kbShort

kbSingle

kbString

kbSunday

kbSystem

kbTab

kbThursday

kbTuesday

kbUseSystem

kbVariant

kbVolume

kbWednesday

kbYes

kbYesNo

kbYesNoCancel

Descriptions

/*

/* COMMENTS */

The comment symbol (’) is used in many code lines in this book. Comments can explain a procedure or a statement. KBasic ignores all comments while compiling and running your program. To write a comment, use the symbol (’) followed by the text of the comment. Comments are printed in green on screen. KBasic recognizes four ways to write comments, as shown below.

REM this is a comment

‘ this is a comment, too

and like in Java

/* comment begin and comment end */

Comments are extremely helpful when it comes to explaining your code to other programmers. So comments, normally, describe how your program works.

Example

```
rem
'   This is yet another test ' c = 3.14
REM This is another test ' a = 4
print "The end!"  ' another rem here!

'END  : REM definitely the end
```

```
DIM n AS INTEGER
Dim s As String

/**

this is a documentation comment
*/


/*
this is mulitline comment
*/

/*
s = "to be or not to be"

n = 200
*/

REM n = 9999

REM n fkdjfalksjfd
'fdnklfsflsgdngndl dflyjvn

REM This is a test of REM ' x = 2

PRINT "Gloria in exelsis deo."
```

See also ', Rem, /**

/**

/* * COMMENTS */

The comment symbol (’) is used in many code lines in this book. Comments can explain a procedure or a statement. KBasic ignores all comments while compiling and running your program. To write a comment, use the symbol (’) followed by the text of the comment. Comments are printed in green on screen. KBasic recognizes four ways to write comments, as shown below.

REM this is a comment

‘ this is a comment, too

and like in Java

/* comment begin and comment end */

Comments are extremely helpful when it comes to explaining your code to other programmers. So comments, normally, describe how your program works.

Example

```
rem
'   This is yet another test ' c = 3.14
REM This is another test ' a = 4
```

```
print "The end!"  ' another rem here!

'END  : REM definitely the end


DIM n AS INTEGER
Dim s As String

/**

this is a documentation comment
*/


/*
this is mulitline comment
*/

/*
s = "to be or not to be"

n = 200
*/

REM n = 9999

REM n fkdjfalksjfd
'fdnklfsflsgdngndl dflyjvn

REM This is a test of REM ' x = 2

PRINT "Gloria in exelsis deo."
```

See also ', Rem, /*

'

' COMMENTS

The comment symbol (') is used in many code lines in this book. Comments can explain a procedure or a statement. KBasic ignores all comments while compiling and running your program. To write a comment, use the symbol (') followed by the text of the comment. Comments are printed in green on screen. KBasic recognizes four ways to write comments, as shown below.

REM this is a comment

‘ this is a comment, too

and like in Java

/* comment begin and comment end */

Comments are extremely helpful when it comes to explaining your code to other programmers. So comments, normally, describe how your program works.

Example

```
rem
'   This is yet another test ' c = 3.14
REM This is another test ' a = 4
print "The end!"  ' another rem here!

'END  : REM definitely the end


DIM n AS INTEGER
Dim s As String

/**

this is a documentation comment
*/


/*
this is mulitline comment
*/

/*
s = "to be or not to be"

n = 200
*/

REM n = 9999

REM n fkdjfalksjfd
'fdnklfsflsgdngndl dflyjvn

REM This is a test of REM ' x = 2

PRINT "Gloria in exelsis deo."
```

See also /, Rem, /***

$Dynamic

REM $Dynamic or ' $Dynamic VB6! QB!

Provided for QBasic backward compatibility.

$End

$End

Stops any execution and compilation after this line.

$Static (Outdated)

REM $Static or ' $Static VB6! QB!

Provided for QBasic backward compatibility.

--------A--------

Abstract

Abstract Class CLASSNAME Inherits PARENTCLASSNAME

It is used when defining classes. Marks a class as abstract.

See also Class

AddressOf

AddressOf(Variable) As Long

Returns the phyiscal address of the variable in memory. This is useful for DLL or SO calls.

Alias

Class CLASSNAME Alias Lib "LIBRARYNAME"…End Class

See Lib for more information (new style).

Declare Sub SUBNAME Lib "LIBRARYNAME" Alias SUBNAME2(ARGUMENTS) VB6! QB!

See Declare for more information (old style).

As

Dim VARIABLENAME As VARIABLETYPE

Sub SUBNAME(ByRef VARIABLENAME As VARIABLETYPE)

It is used whenever you declare variables. After As, you write the type of the variable you currently declare. See the manual for more information on this.

See also Dim

--------B--------

Base

Option Base {0|1} VB6! QB!

Set the default lower bound of arrays.

Provided for QBasic backward compatibility.

Binary

Option Compare {Binary|Text}

Sets the comparision mode of StrComp.

See also StrComp

ByRef

Sub SUBNAME(ByRef VARIABLENAME As VARIABLETYPE)

Defines the given variable to the sub or function, to be handled by reference. This means, that chaning the value of this varialbe affects the original variable as well. See the manual for more information on this.

See also ByVal

ByVal

Sub SUBNAME(ByVal VARIABLENAME As VARIABLETYPE)

Defines the given variable to the sub or function, to be handled by value. This means, that chaning the value of this varialbe DOES NOT affect the original variable as well. See the manual for more information on this.

See also ByRef

--------C--------

Call

Call SUBNAME(ARGUMENTS) VB6! QB!

Calls a sub routine. You do not need to use Call, because it is obsolete and provided for backward compatibility.

Case

Select Case EXPRESSION…Case EXPRESSION…End Select

Select Case EXPRESSION…Case EXPRESSION To EXPRESSION…End Select

Select Case EXPRESSION…Case Is OPERATOR EXPRESSION…End Select

Select Case EXPRESSION…Case Else…End Select

It is used for Select Case, which introduces a multi-line conditional selection statement.

Example

```
Dim i As Double
Dim n As Integer

i = 4

Select Case i
Case 0
  n = 0
Case 1, 2
  n = 1122
Case 4 TO 10
  n = 441000
Case Is = 9
  n = 9999
Case Else
  n = 999999
End Select
```

See also Select

Catch

Try…Catch(VARIABLE AS VARIABLETYPE)…End Catch

It is used for Try Catch, which introduces a exception handling.

Example

```
Try
  test()
Catch (b As rumba)
  Print "tt2: got you!"
  b.dance()
End Catch
```

See also Try

Choose

Choose(Index, Select - 1 [, Select - 2, … [, Select - n]])

Returns one value from a list of values depending on the index.

Example

```
Dim s As String
s = Choose(1, "un", "deux", "troi")
Print s
```

See also If, IIf, Select Case

Class

Classes are needed, when you would like use objects. See the manual for more information.

A simple application can contain a form, while all source code is inside the form module of that form. But if the application grows bigger and bigger it might be useful, to write source codes from different forms at one place, so you need a place to write the source codes down outside the forms. Here comes the classes. Create a class, which contains a methods, which is useful for your forms. You KBasic code is stored in classes or modules. You can archive your code within classes. Every class consists of the declaration part and the methods you have inserted. A class can contain:

- Declarations - for variables, types, enumerations and constants
- Methods (also called procedures) - which are not assigned to a special event or object. You can create as many procedures as you want, e.g. sub-procedures without return value or function-procedures.
- Signal/Slots-Methods - These are special methods, which are only useful together with the bindings. See the documentation of the bindings in the KBasic IDE for more information.
- Properties - Are variables, which are accessed through two special methods (get and set method). Properties are accessable without the need to write braces, when you use them.

You can put several classes or modules in one file, but you should not do that.

Syntax

```
[Abstract] Class Name Inherits ParentClassName

  [Static] Dim Name As Type
  [Static] Public Name As Type
  [Static] Protected Name As Type
  [Static] Private Name As Type
  Const Name As Type
  Public Const Name As Type
  Protected Const Name As Type
  Private Const Name As Type
  ...


  [Public | Protected | Private]
  Enum Name
    Name As Type
    ...
  End Enum
  ...


  [Public | Protected | Private]
  Type Name
    Name As Type
    ...
  End Type
  ...


  [Public | Protected | Private]
  Property Name As Type

    Get
      [Statements]
    End Get

    Set(Argument)
      [Statements]
    End Set

  End Property
  ...


  [Public | Protected | Private]
  Constructor Name([Arguments])
    [Statements]
  End Constructor
  ...


  [Public | Protected | Private]
  Destructor Name( )
    [Statements]
  End Destructor


  [Static] [Public | Protected | Private]
  Function Name([Arguments]) [As Type] [Throws Name, ...]
    [Statements]
  End Function
  ...


  [Static] [Public | Protected | Private]
  Sub Name([Arguments]) [Throws Name, ...]
    [Statements]
  End Sub
  ...


  [Public | Protected | Private]
  Slot Name([Arguments])
    [Statements]
  End Slot
  ...


  [Public | Protected | Private]
  Signal Name([Arguments])
    [Statements]
  End Signal
  ...
```

```
End Class
```

See also Module

Class_Initialize

Sub Class_Initialize() VB6! QB!

It is used as default constructor of custom classes in old VB6 code.

See also Constructor

Class_Terminate

Sub Class_Terminate() VB6! QB!

It is used as destructor of custom classes in old VB6 code.

See also Destructor

Common

Common Shared VARIABLENAME As VARIABLETYPE VB6! QB!

It is provided for old QBasic code.

Compare

Option Compare {Text|Binary}

It is used to set the comparision mode of StrComp by default.

See also StrComp

Connect

Connect(SIGNALOBJECTNAME, Signal(SIGNALDESCRIPTION), SLOTOBJECTNAME, Slot(SLOTDESCRIPTION))

Enables the event of a signal, so that the related functionality (slot sub-procedure) is called whenever the signal appears. This is related to the Qt-Bindings.

Example

```
Connect(internalTimer, Signal(timeout()), Me, Slot(timeout()))
```

See also Signal, Slot, Disconnect

Const

Const NAME [As TYPE] = EXPRESSION

Declares a constants.

Constants are similar to variables but they cannot change values. When you declare a constant you assign a value to it that annot be altered during lifetime of your program.

Example

```
Sub Namer (  )
  Const pi = 3.14
  Print pi
End Sub


Namer()

Sub test
  Dim k As Integer

  k = 9 + 23
  Print k

End Sub

Const a = 123.88 * 2, bb = 6
Const k As Integer = 2

Dim i As Double

i = bb

test

'a = i ' would cause a parser error
```

See also Dim

Constructor

[Public | Protected | Private] Constructor Name([Arguments])

See the manual for more information.

See also Class, Destructor

--------D--------

Data

Data EXPRESSION [, EXPRESSION, ..] VB6! QB!

Represents data inside your program. This is heavily used in old QBasic code.

```
OPTION VERYOLDBASIC

DATA "Salsa"

READ a$


DATA 22

READ t%

'$END


DATA 66, 77

READ t%, txt

RESTORE
'$END

READ a$, txt

DATA 55, 99

READ t%, txt
READ t%, txt
```

See also Restore, Read

Declare

Declare Sub SUBNAME Lib “LIBRARYNAME” Alias SUBNAME2(ARGUMENTS) VB6! QB!

Declaration of external procedures of DLL or SO files. This is the old VB6 style, you ought to use the new syntax style. See Lib for more information.

Example

```
' following declaration on Windows returns the computer name

Declare Function GetComputerName Lib "kernel32" Alias _
"GetComputerNameA"(ByVal lpBuffer As String, nSize As Integer) As Integer
```

See also Lib

DefBool

DefBool RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefByte

DefByte RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefCur

DefCur RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefDate

DefDate RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefDbl

DefDbl RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefInt

DefInt RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefLng

DefLng RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefObj

DefObj RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefSng

DefSng RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefStr

DefStr RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

DefVar

DefVar RANGE VB6! QB!

Provided for QBasic compatibility.

Example

```
' set all variables beginning with letter A till M to be of type Integer in this
example
DefInt A - M
```

Destructor

[Public | Protected | Private] Destructor Name()

See the manual for more information.

See also Class, Constructor

Dim

Dim VARIABLENAME[([Indexes])] [As [New] Typ] [, VARIABLENAME[([Indexes])] [As [New] VARIABLETYPE]]

Before using variables, you must declare them. You must define the name and the data type of a variable. The 'Dim'-statement declares a variable. See the manual for more information.

Example

```
Dim A2 As Integer, B2 As Integer
```

See also Class, Module, Private, Protected, Public

Disconnect

Related to the Qt-Bindings. Sorry. Not implemented yet. Normally, you do not disconnet signals from slots.

See also Signal, Slot, Connect

Do

Do…Loop Until EXPRESSION

Do While EXPRESSION…Loop

Loop-statements

The statements that control decisions and loops in KBasic are called control structures. Normally every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands depending upon a condition. Some loops repeat commands while a condition is 'True,' other loops repeat commands while a condition is 'False.' There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

Use the following loops when you are not sure how often a command should be repeated: 'Do', 'While', 'Loop', 'Until' or ('Wend') . There are two different ways to use the keyword 'While' in order to test a condition within a 'Do…Loop'-statement. You can test the condition before the commands inside the loop are executed or you can test the condition after the commands of the loop have been executed at least once. If the condition is 'True' (in the following procedure 'SubBefore') the commands inside the loop execute.

Example

```
Sub SubBefore()
  Counter = 0
  myNumber = 20
  Do While myNumber > 10
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop
  MsgBox "Loop has been executed " & Counter & " time(s)."
End Sub
```

```
Sub SubAfter()
  Counter = 0
  myNumber = 9
  Do
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop While myNumber > 10
  MsgBox "Loop has been executed " & Counter & " time(s)."
End Sub
```

See also While, Loop, Until, For

--------E--------

Each

The statements that control decisions and loops in KBasic are called control structures. Normally every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands depending upon a condition. Some loops repeat commands while a condition is 'True,' other loops repeat commands while a condition is 'False.' There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

For Each VARIABLENAME In EXPRESSION…Next

The For-Each loop is useful when you want to iterate over a list of objects provided by the KBasic Framework.

See also For

Else

If EXPRESSION Then … Else … EndIf

A single decision is used to execute a set of statements if a condition is set ('If'-statement). If the condition is 'True' then the statements after the 'Then' are executed and the statements after the 'Else' are skipped. If the condition is 'False,' the statements after the 'Else' are executed.

Example

```
Dim i As Integer
Dim n As Integer

If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

See also If, ElseIf, Then, EndIf, Select

ElseIf

If EXPRESSION Then … ElseIf … EndIf

A single decision is used to execute a set of statements if a condition is set (’If’-statement). If the condition is ‘True’ then the statements after the ‘Then’ are executed and the statements after the ‘Else’ are skipped. If the condition is ‘False,’ the statements after the ‘Else’ are executed.

Example

```
Dim i As Integer
Dim n As Integer

If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

See also Else, If, Then, EndIf, Select

Empty

Empty VB6! QB!

The data type ‘Variant’ is automatically used if you do not specifiy a data type for an argument, constant, procedure or variable. An exception, is using a variable in ‘VeryOldBasic’-Mode. It has no ‘Variant’ but ‘Double’-data type. Variables of type ‘Variant’ can contain strings, dates, time values, boolean values or even numerical values.

Additionally, a ‘Variant’ can store the following values:

- ‘Null’
- ‘Nothing’ (old style)
- ‘Empty’

Example

```
Dim v As Variant
v = Empty
v = Null
```

See also Null, Nothing

End

End

End Constructor

End Destructor

End Sub

End Function

End Signal

End Slot

End Module

End Class

End With

End Catch

End Type

End Enum

End If

End Select

End Set

End Get

End Property

Either it ends the execution of a program immediately, or it is used to close the current sub, function or other language structure. See the list above.

Example

```
Dim i As Integer
End
i = 2 ' won't be executed, because of End in line before
```

See also Stop

EndIf

If EXPRESSION Then … Else … EndIf

A single decision is used to execute a set of statements if a condition is set (’If’-statement). If the condition is ‘True’ then the statements after the ‘Then’ are executed and the statements after the ‘Else’ are skipped. If the condition is ‘False,’ the statements after the ‘Else’ are executed.

Example

```
Dim i As Integer
Dim n As Integer

If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
```

```
Else
  n = 33333
End If
```

See also Else, ElseIf, Then, If, Select

Enum

Enum ENUMNAME…End Enum

Enumeration is a new way of grouping constants related to the same subject. It is a list of names and every name has a constant value.

Example

```
Enum Level
   Mo = -1
   Di = 0
   Fr = 1
   Sa = 1 + Fr AND 2
End Enum

Enum test
   Entry
   Entry2
   Security = Entry
End Enum

Print 3 + Level.Mo
Print test.Entry
Print test.Security
```

See also Type

Erase

Erase VARIABLENAME

The command ‘Erase’ deletes arrays and frees the memory of arrays.

Example

```
Type o
  s As String * 100
End Type

Dim oo As o

oo.s = "33"

Erase oo

Print Len(oo.s)
```

See also Dim, ReDim

Exit

Exit For

- Explicit leave of for loop.

Exit Do

- Explicit leave of do loop.

Exit Sub

- Explicit leave of sub.

Exit Function

- Explicit leave of function.

Example

```
Sub doingSomething

  Print "did something"

  Exit Sub

  Print "end of sub"

End Sub


Function doingSomething2() As Variant

  Print "did something"

  Exit Function

  Print "end of function"

End Function

For i As Integer = 1 To 11
  Exit For
  Print "xyz"
Next

doingSomething()
doingSomething2()
```

Explicit

Option Explicit {On|Off} VB6! QB!

Declaration of variables automatically / implicit declaration

For historical reasons it is possible to use variables without declaration. It is supported within a special mode in which all variables are declared automatically when using a variable name that has not been used before. Important! You should not use this mode unless you want to use old BASIC code without changing it. Good programming style dictates declaring all variables with their types. This also makes it easier for others to understand your code and minimizes typing errors.

In order to activate implicit declaration, write the following line in top of your program:

Use of the ‘Option Explicit’-statement (supported within ‘Mode OldBasic’ and ‘Mode VeryOldBasic’ only) As mentioned previously, you can implicitly declare a variable in KBasic by using a assignment statement or expression with the variable name. All implicitly declared variables have the data type ‘Variant’ (’Option OldBasic’) or ‘Double’ (’Option VeryOldBasic’). Variables of type ‘Variant’ need more space in memory than most of the other data types. Your program is faster when you explicitly declare your variables with the smallest data type possible. Another advantage is that using declaration avoids name collisions or typing errors.

To explicitly declare variables within ‘Option OldBasic’ or ‘Option VeryOldBasic,’ write on top of all module and class files ‘Option Explicit On.’ This creates a compilation error when a variable name is not declared. If you are using the ‘KBasic Mode’ you must declare all variables. It is like ‘Option Explicit On.’ By the way, the ‘KBasic Mode’ is set to default in all programs.

Important! You must declare dynamic arrays or fixed arrays at all times. Furthermore, you can mix the modes in your program. One file is set ‘Option OldBasic,’ another file is set ‘Option VeryOldBasic.’ You are free to set ‘Option Explicit Off’ for the old modes.

Example

```
OPTION OLDBASIC
OPTION EXPLICIT OFF ' turn off

'OPTION BASE 0 ' 1  standard 1
i$ = "Heyoi"

' turn runtime over/underflow check on
'OPTION RANGE ON

' let's do an overflow!

DIM a AS INTEGER ' 32-bit integer
a = 2147483647  ' the maximum positive signed integer
a = a + 1  ' this is overflow...  a is now -2147483648
```

See also Option

--------F--------

Finally

Try…Catch(ARGUMENT)…Finally…End Catch

The ‘Try-Catch’-statement is needed to catch an exception. ‘Try’ encloses the normal code, which should run fine. 'Catch' contains the commands that should be executed if an exception has been raised. 'Finally' has some commands that should be executed whether an exception has happened or not.

'Finally' is useful when you have file accessing or database closing commands that must always execute even when an exception occurs. If you have defined a 'Catch'-statement and 'Finally'-statement and the matching exception is raised, first the 'Catch'-statement executes,then the 'Finally'-statement.

Example

```
Try
  test()
Catch (b As rumba)
  Print "tt2: got you!"
  b.dance()
Finally
  Print "tt2: will be always executed, whatever happend"
End Catch
```

See also Try, Catch

For

The statements that control decisions and loops in KBasic are called control structures. Normally every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands depending upon a condition. Some loops repeat commands while a condition is 'True,' other loops repeat commands while a condition is 'False.' There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

For VARIABLENAME = EXPRESSION To EXPRESSION [Step EXPRESSION]…Next

- The For-Next loop is useful when you know how often statements should be repeated. For-Next defines a loop that runs a specified number of times.

For Each VARIABLENAME In EXPRESSION…Next

- The For-Each loop is useful when you want to iterate over a list of objects provided by the KBasic Framework.

Example

```
Dim ctr As Integer

For ctr = 1 To 5
  Print "Z";
Next
```

See also Next, Each

Function

Function SUBNAME(ARGUMENTS) As RETURNTYPE…End Function

A sub-procedure can have arguments, variables, expressions, or constants that are given to the sub-procedure when calling it. Function-procedures return values.

Example

```
' function example

Function divide(dividend As Double, divisor As Double) As Double

  Return dividend / divisor

End Function

Print divide(18, 9)
```

See also Sub

--------G--------

Global

Global VARIABLENAME As VARIABLETYPE VB6! QB!

Provided for QBasic compatibility.

GoSub

GoSub {LINENO|LABEL} VB6! QB!

On EXPRESSION GoSub {LINENO|LABEL, LINENO|LABEL…} VB6! QB!

Provided for QBasic compatibility.

Example

```
OPTION VERYOLDBASIC

DIM i%

i% = 1
i% = 2

ON i% GOSUB one, two

PRINT "THE END"
```

```
END

one:
PRINT "one"
RETURN

two:
PRINT "two"
RETURN
```

GoTo

GoTo {LINENO|LABEL}

Unconditional jumping

Programmers can also use unconditional branches. This type of branching can be performed using the 'GoTo'-instruction. 'GoTo' forces program execution to branch to a specific line number or label. Because line numbers in KBasic programs are now obsolete, you do not have to worry about how to use them. You may, however, want to use labels. 'GoTo' performs a unconditional jump. 'GoTo' is always executed, without a condition.

Example

```
DIM b AS INTEGER
DIM n AS INTEGER

b = 45
GOTO bernd
b = 99999
bernd:

n = 0
ok:
n = n + 1
IF n < 5 THEN GOTO ok
```

--------I--------

If

If EXPRESSION Then … Else … EndIf

A single decision is used to execute a set of statements if a condition is set ('If'-statement). If the condition is 'True' then the statements after the 'Then' are executed and the statements after the 'Else' are skipped. If the condition is 'False,' the statements after the 'Else' are executed.

Example

```
Dim i As Integer
Dim n As Integer
```

```
If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

See also Else, ElseIf, Then, EndIf, Select

IIf

IIf(EXPRESSION, THENRETURNEXPRESSION, ELSERETURNEXPRESSION)

IIf returns a value of two values depending on an expression.

Example

```
Function testing(Test1 As Integer) As String
  Return IIf(Test1 > 1000, "big", "small")
End Function

Print testing(5)
Print testing(5555)
```

See also If

Inherits

[Abstract] Class CLASSNAME Inherits PARENTCLASSNAME

Inherits defines, which is the parent class of the current class. Classes are needed, when you would like use objects. See the manual for more information.

See also Class

Is

TypeOf VARIABLENAME Is CLASSNAME

- Returns true, if variable is an object of that class.

Select Case EXPRESSION…Case Is OPERATOR EXPRESSION…End Select

- Used in a select case statement for comparision.

See also TypeOf

Iterate

Iterate For

- Manually test loop condition for a 'For'-loop.

Iterate Do

- Manually test loop condition of a 'Do'-loop.

See also Exit, Do, For

--------K--------

KBasic

Option KBasic

Using one of the following commands you can switch KBasic modes: If you want to use KBasic's newest features (default), turn it on.

Example

```
Option KBasic
Print 21
```

See also Option, OldBasic, VeryOldBasic

--------L--------

LBound

LBound(VARIABLENAME [, Index])

Getting lower bound of an array. LBound is used for the lower bound.

Example

```
TYPE book
  bkname AS STRING * 100

  isbn(1000) AS INTEGER
END TYPE

TYPE address
  books(50) AS book
  age AS INTEGER
  name AS STRING * 100
END TYPE

DIM j(5 TO 10) AS book


PRINT LBOUND(j, 1)
```

See also UBound

Let

Let VARIABLE = EXPRESSION VB6! QB!

Provided for QBasic compatibility.

Lib

Class CLASSNAME Alias Lib "LIBRARYNAME"…End Class

Declaration of external procedures of DLL or SO files. This is the new KBasic style, you ought to use the new syntax style.

Declare Sub SUBNAME Lib "LIBRARYNAME" Alias SUBNAME2(ARGUMENTS)

This is the old style.

Example for new style

```
' zunächst die benötigten API-Deklarationen

Class comdlg32 Alias Lib "comdlg32.dll"

  Static Function ChooseColor_Dlg Alias "ChooseColorA"_
  (lpcc As CHOOSECOLOR_TYPE) As Integer

  Type CHOOSECOLOR_TYPE
    lStructSize As Integer
    hwndOwner As Integer
    hInstance As Integer
    rgbResult As Integer
    lpCustColors As Integer
    flags As Integer
    lCustData As Integer
    lpfnHook As Integer
    lpTemplateName As String
  End Type

  ' Anwender kann alle Farben wählen
  Const CC_ANYCOLOR = &H100
  ' Nachrichten können "abgefangen" werden
  Const CC_ENABLEHOOK = &H10
  ' Dialogbox Template
  Const CC_ENABLETEMPLATE = &H20
  ' Benutzt Template, ignoriert aber den Template-Namen
  Const CC_ENABLETEMPLATEHANDLE = &H40
  ' Vollauswahl aller Farben anzeigen
  Const CC_FULLOPEN = &H2
```

```
    ' Deaktiviert den Button zum Öffnen der Dialogbox-Erweiterung
    Const CC_PREVENTFULLOPEN = &H4
    ' Vorgabe einer Standard-Farbe
    Const CC_RGBINIT = &H1
    ' Hilfe-Button anzeigen
    Const CC_SHOWHELP = &H8
    ' nur Grundfarben auswählbar
    Const CC_SOLIDCOLOR = &H80

End Class

Dim CC_T As comdlg32.CHOOSECOLOR_TYPE, Retval As Integer
Dim BDF(16) As Integer

'Einige Farben vordefinieren (Benutzerdefinierte Farben)
BDF(0) = RGB(255, 255, 255)
BDF(1) = RGB(125, 125, 125)
BDF(2) = RGB(90, 90, 90)

'Print Len(CC_T) 'Strukturgröße
With CC_T
  .lStructSize = Len(CC_T) 'Strukturgröße
  .hInstance = 0'App.hInstance    'Anwendungs-Instanz
  .hwndOwner = 0 'Me.hWnd 'Fenster-Handle
  .flags = comdlg32.CC_RGBINIT Or comdlg32.CC_ANYCOLOR Or comdlg32.CC_FULLOPEN
Or comdlg32.CC_PREVENTFULLOPEN 'Flags
  .rgbResult = RGB(0, 255, 0)      'Farbe voreinstellen
  .lpCustColors = AddressOf(BDF(0)) 'Benutzerdefinierte Farben zuweisen
End With

Retval = comdlg32.ChooseColor_Dlg(CC_T) 'Dialog anzeigen


If Retval <> 0 Then
  MsgBox Hex$(CC_T.rgbResult) 'gewählte Farbe als Hintergrund setzen
Else
  MsgBox "Das Auswählen einer Farbe ist fehlgeschlagen," & _
  "oder Sie haben Abbrechen gedrückt", kbCritical, "Fehler"
End If
```

Like

EXPRESSION Like EXPRESSION VB6! QB!

Provided for VB6 compatibility.

- ? Match any single character.
- * Match zero or more characters.
- # Match a single digit (0-9).
- no special meaning of ?,*,#, [inside []; for] outside von []
- inside of [] search for more than one character is allowed, defined without separator
- [A-Za-z0-9],.+-

- [*] escape with []
- []] now allowed but …][…]]…
- [] zero length string
- [A-Za-z0-9[*]],.+-
- [!charlist] Match any char not in the list.
- - at the end or the beginnig [] even -
- #[#][*]*[?]

Example

```
CLS

DIM i

i = "aab" LIKE "aab"

PRINT i

'PRINT "abcdfgcdefg" LIKE "" ' False
'PRINT "abcg" LIKE "a*g" ' True
'PRINT "abcdefcdefg" LIKE "a*cde*g" ' True
'Print "abcdefgcdefg" Like "a*cd*cd*g" ' True
'Print "abcdefgcdefg" Like "a*cd*cd*g" ' True
'Print "00aa" Like "####" ' False
'Print "00aa" Like "????" ' True
'PRINT "00aa" LIKE "##??" ' True
'PRINT "00aa" LIKE "*##*" ' True
'PRINT "hk" LIKE "hk*" ' True
'PRINT "00aa" LIKE "*[1-9]*" ' True
'PRINT "*?x]" LIKE "[*?a-z]]"

'PRINT "10" LIKE "[!0-9a-z]" ' True

'PRINT "" LIKE "[]"

PRINT "-*?0x-" LIKE "[-*?0-9a-z-]"
```

Loop

Do…Loop Until EXPRESSION

Do While EXPRESSION…Loop

Loop-statements. See Do fore more information.

See also Do

LSet

LSet STRINGVARIABLE = EXPRESSION VB6! QB!

Provided for QBasic compatibility.

--------M--------

Me

Me

Use current instance or object

The keyword 'Me' references the current instance (or object) in which the code is currently executed. 'Parent' refers to the current parent object. Normally it is the current class (user defined class or form class).

Example

```
Sub changeObjectColor(Object1 As Object)
        Object1.BackColor = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub

changeObjectColor(Me) ' statement inside a class


Class movies

  Protected sMovieName As String

  Sub printName
    print sMovieName
  End Sub

  Constructor movies(s As String)
    sMovieName = s
  End Constructor

End Class


Class movies2 Inherits movies

  Constructor movies2(ByRef s As String)
    Parent.movies(s + "2")
  End Constructor

End Class


Dim k As Integer = 9


Dim m As New movies2("final fantasy")

m.printName()
```

See also Parent

Mid

Mid(STRINGVARIABLE, Position As Integer, Length As Integer) = STRINGEXPRESSION

Replaces text inside a string by another text.

Example

```
OPTION OLDBASIC

DIM txt AS STRING, replacement AS STRING, originaltxt AS STRING

replacement = "The power of KBasic"
originaltxt = "********************"
FOR i = 1 to LEN(replacement)
         MID(originaltxt, 2, i) = replacement
         PRINT originaltxt
NEXT i
```

See also Left

Module

A simple application can consist of only one form while the complete source code is in one form module. As your applications grow larger you probably would like to use the same code in different forms. To do so, place this code in a global module file as it is accessible by the entire application.

You KBasic code is stored in classes or modules. You can archive your code within modules. Every module consists of the declaration part and the procedures you have inserted.

A module can contain:

- Declarations - for variables, types, enumerations and constants
- Procedures - which are not assigned to a special event or object. You can create as many procedures as you want, e.g. sub-procedures without return value or function-procedures.

You can put several classes or modules in one file but you should not.

See the manual for more information.

Example

```
Module Name

  Dim Name As Type
  Public Name As Type
  Private Name As Type
  Const Name As Type
  Public Const Name As Type
  Private Const Name As Type
  ...


  [Public | Private]
  Enum Name
    Name As Type
    ...
  End Enum
  ...


  [Public | Private]
  Type Name
    Name As Type
    ...
  End Type
  ...


  [Public | Private]
  Function Name([Arguments]) [As Type] [Throws Name, ...]
    [Statements]
  End Function
  ...


  [Public | Private]
  Sub Name([Arguments]) [Throws Name, ...]
    [Statements]
  End Sub
  ...

End Module
```

See also Class

--------N--------

New

VARIABLE = New CLASSNAME(ARGUMENTS)

Dim VARIABLE As VARIABLETYPE = New CLASSNAME(ARGUMENTS)

Create new objects: Either you create an object based on a built-in class of KBasic, or you create it from your own defined class.

A variable, which contains the object and is declared, does not create the object at once. This variable only has the data type of an object and represents only a reference to an object still to be created. Create objects using the special keyword ‘New.’ This creates an object by giving it the desired class name and returns a reference to the new object. The reference is stored in a variable of the class type. In this way, creating objects in KBasic is the same as for Java or C++.

‘New’ is followed by the class name of the new object. The class has a special procedure called constructor, called by ‘New’. Using ‘New’ creates a new instance (or object) of a class. The matching constructor of that class is executed. When using or declaring constructors, consider two important things first: The name of the constructor matches the name of the class it means The return value is always an instance of that class (implicitly). There is no return type declaration necessary, nor is the keyword ‘Sub’ or ‘Function’ used. This constructor does some initial work for an object and returns a reference to the new created object.

See the manual for more information.

Example

```
Class a Inherits b

  Sub t()

    Dim k As Integer = Parent.v
    Print k

  End Sub
'
End Class


Class b

  Public v As Integer

End Class

Dim aa As New a
aa.v = 99
aa.t
```

See also Dim

Next

For VARIABLENAME = EXPRESSION To EXPRESSION [Step EXPRESSION]…Next

- The For-Next loop is useful when you know how often statements should be repeated. For-Next defines a loop that runs a specified number of times.

For Each VARIABLENAME In EXPRESSION…Next

- The For-Each loop is useful when you want to iterate over a list of objects provided by the KBasic Framework.

Example

```
Dim ctr As Integer

For ctr = 1 To 5
  Print "Z";
Next
```

See also For, Each

Nothing

Nothing VB6! QB!

The data type 'Variant' is automatically used if you do not specifiy a data type for an argument, constant, procedure or variable. An exception, is using a variable in 'VeryOldBasic'-Mode. It has no 'Variant' but 'Double'-data type. Variables of type 'Variant' can contain strings, dates, time values, boolean values or even numerical values.

Additionally, a 'Variant' can store the following values:

- 'Null'
- 'Nothing' (old style)
- 'Empty'

Example

```
Dim v As Variant
v = Nothing
```

See also Null

Null

Null

The data type 'Variant' is automatically used if you do not specifiy a data type for an argument, constant, procedure or variable. An exception, is using a variable in 'VeryOldBasic'-Mode. It has no 'Variant' but 'Double'-data type. Variables of type 'Variant' can contain strings, dates, time values, boolean values or even numerical values.

Additionally, a 'Variant' can store the following values:

- 'Null'
- 'Nothing' (old style)
- 'Empty'

Example

```
Dim v As Variant
v = Null
```

See also Nothing

--------O--------

Off

Option Explicit {On|Off} VB6! QB!

Option Range {On|Off} VB6! QB!

Example

```
OPTION OLDBASIC
OPTION EXPLICIT OFF ' turn off

'OPTION BASE 0 ' 1  standard 1
i$ = "Heyoi"

' turn runtime over/underflow check on
'OPTION RANGE ON

' let's do an overflow!

DIM a AS INTEGER ' 32-bit integer
a = 2147483647  ' the maximum positive signed integer
a = a + 1  ' this is overflow...  a is now -2147483648
```

See also Explicit, Range, On

OldBasic

Option OldBasic VB6! QB!

Using one of the following commands you can switch KBasic modes: If you want to use old VB6 code, turn it on.

Example

```
Option OldBasic
Print 21
```

See also Option, KBasic, VeryOldBasic

On

On Error GoTo {LINENO|LABEL} VB6! QB!

On EXPRESSION GoSub {LINENO|LABEL, LINENO|LABEL…} VB6! QB!

Option Explicit {On|Off} VB6! QB!

Option Range {On|Off} VB6! QB!

Example

```
OPTION OLDBASIC
OPTION EXPLICIT OFF ' turn off

'OPTION BASE 0 ' 1  standard 1
i$ = "Heyoi"

' turn runtime over/underflow check on
'OPTION RANGE ON

' let's do an overflow!

DIM a AS INTEGER ' 32-bit integer
a = 2147483647  ' the maximum positive signed integer
a = a + 1  ' this is overflow...  a is now -2147483648
```

See also Explicit, Range, Off

Option

Option Compare {Binary|Text}

Sets the comparision mode of StrComp.

Option Explicit {On|Off} VB6! QB!

See Explicit for more.

Option Range {On|Off} VB6! QB!

See Range for more.

Option Base {0|1} VB6! QB!

See Base for more.

Option KBasic

See KBasic for more.

Option OldBasic VB6! QB!

See OldBasic for more.

Option VeryOldBasic VB6! QB!

See VeryOldBasic for more.

Optional

[ByVal | ByRef] Optional VARIABLENAME[()][As VARIABLETYPE] VB6! QB!

Optional maybe used, when you do not call the sub with all arguments. Arguments, which are marked as optional might be left out.

This is provided for VB6 backward compatibility. You are strongly encouraged to use the default values of arguments instead. See the manual for more information.

Example

```
Sub jump(meter As Integer, Optional high As Integer)

  If Not IsMissing(high) Then
    Print "high jump"
  Else
    print "normal jump"
  End If

End Sub

jump(12)
jump(12, 33)
```

See also IsMissing

--------P--------

ParamArray

ParamArray VARIABLENAME() As Variant VB6! QB!

This is provided for VB6 backward compatibility. You are strongly encouraged not to use ParamArray at all.

Example

```
Sub jump(meter As Integer, Optional high As Integer)

  If Not IsMissing(high) Then
    Print "high jump"
  Else
    print "normal jump"
  End If

End Sub

jump(12)
jump(12, 33)
```

Parent

Parent

Use current instance or object

The keyword 'Me' references the current instance (or object) in which the code is currently executed. 'Parent' refers to the current parent object. Normally it is the current class (user defined class or form class).

Example

```
Class movies

  Protected sMovieName As String

  Sub printName
    print sMovieName
  End Sub

  Constructor movies(s As String)
    sMovieName = s
  End Constructor

End Class


Class movies2 Inherits movies

  Constructor movies2(ByRef s As String)
    Parent.movies(s + "2")
  End Constructor

End Class


Dim k As Integer = 9


Dim m As New movies2("final fantasy")

m.printName()
```

See also Me

Preserve

ReDim Preserve VARIABLENAME[Index]

Preserve does not clear the array, when you change the size of the array by using ReDim.

Example

```
Sub te
```

```
  Dim i[10] As Integer

  i[0] = 99
  i[1] = 88
  i[2] = 77
  i[3] = 66
  i[4] = 55
  i[5] = 44

  ReDim Preserve i[20]

  Print i[0]

End Sub

te()
```

See also Dim, ReDim

Private

Private VARIABLENAME[([Indexes])] [As [New] Typ] [, VARIABLENAME[([Indexes])] [As [New] VARIABLETYPE]]

Before using variables, you must declare them. You must define the name and the data type of a variable. Use of the 'Private'-Statement Use the 'Private'-statement to declare private variables in module scope or class scope, making the variable accessible only from the same scope (module scope, all module procedures, class scope, all class methods).

See the manual for more information.

See also Class, Module, Dim, Protected, Public

Property

Property PROPERTYNAME As PROPERTYTYPE…Get…End…Get…Set(ARGUMENT)…End Set…End Property

Most objects and controls have properties that you can think of as nothing more complicated than attributes. For example, a 'CommandButton' control has a property called 'Caption' that determines the text that appears in the button. Many other controls, such as the familiar 'Label' control, also have a 'Caption' property. Some properties are common in KBasic whereas others are associated with only a single control or object. Those properties are built into KBasic, but it is also possible to define your own properties within your user defined class.

Defining a property is like defining a procedure. In fact, a property contains two property-procedures. One property-procedure reads the property (Get) and the other writes the property (Set). Additionally, you have to declare a variable that contains the value of the property. Why should I use a property, when I use a variable instead? It is easier to access a variable but it might be useful to check something when accessing a variable; with properties you are able to check values or conditions when you try to access the property. It is like data hiding. You can code so that the value of the property is always correct.

Syntax

```
Property Name As Type

  Get
    [Statements]
  End Get

  Set(Argument)
    [Statements]
  End Set

End Property
```

See also Class, Sub, Function

Protected

Protected VARIABLENAME[([Indexes])] [As Typ] [, VARIABLENAME[([Indexes])] [As VARIABLETYPE]]

Use the 'Protected'-statement to declare protected variables in class scope, making the variable accessible from within the same scope (class scope, all class methods, sub-classes, all sub-classes methods). This allows you to underline the inheritance hierarchy of your classes.

See the manual for more information.

See also Class, Module, Dim, Private, Public

Public

Public VARIABLENAME[([Indexes])] [As Typ] [, VARIABLENAME[([Indexes])] [As VARIABLETYPE]]

Use of the 'Public'-Statement You can use the 'Public'-statement to declare public variables in module scope or class scope, making the variable accessible from everywhere.

See the manual for more information.

See also Class, Module, Dim, Private, Protected

--------R--------

Range

Option Range {On|Off} VB6! QB!

Turns off or on the runtime overflow checking while computing operations. WARNING! Option

Range is ALWAYS off, even you turn it on. There is no way have runtime checking yet, because it is not implemented.

Example

```
OPTION OLDBASIC
OPTION EXPLICIT OFF ' turn off

'OPTION BASE 0 ' 1  standard 1
i$ = "Heyoi"

' turn runtime over/underflow check on
'OPTION RANGE ON

' let's do an overflow!

DIM a AS INTEGER ' 32-bit integer
a = 2147483647  ' the maximum positive signed integer
a = a + 1  ' this is overflow...  a is now -2147483648
```

See also Explicit, On, Off

Read

Read VARIABLENAME VB6! QB!

Reads data inside your program from data arrays. This is heavily used in old QBasic code.

Example

```
OPTION VERYOLDBASIC

DATA "Salsa"

READ a$


DATA 22

READ t%

'$END


DATA 66, 77

READ t%, txt

RESTORE
'$END

READ a$, txt

DATA 55, 99

READ t%, txt
READ t%, txt
```

See also Restore, Data

ReDim

ReDim [Preserve] VARIABLENAME[Index]

ReDim [Preserve] VARIABLENAME[Index, Index, …]

Preserve does not clear the array, when you change the size of the array by using ReDim.

Example

```
Sub te

  Dim i[10] As Integer

  i[0] = 99
  i[1] = 88
  i[2] = 77
  i[3] = 66
  i[4] = 55
  i[5] = 44

  ReDim Preserve i[20]

  Print i[0]

End Sub

te()
```

See also Dim, Preserve

Rem

Rem COMMENTS

The comment symbol (') is used in many code lines in this book. Comments can explain a procedure or a statement. KBasic ignores all comments while compiling and running your program. To write a comment, use the symbol (') followed by the text of the comment. Comments are printed in green on screen. KBasic recognizes four ways to write comments, as shown below.

REM this is a comment

‘ this is a comment, too

and like in Java

/* comment begin and comment end */

Comments are extremely helpful when it comes to explaining your code to other programmers. So comments, normally, describe how your program works.

Example

```
rem
'   This is yet another test ' c = 3.14
REM This is another test ' a = 4
print "The end!"  ' another rem here!

'END  : REM definitely the end


DIM n AS INTEGER
Dim s As String

/**

this is a documentation comment
*/


/*
this is mulitline comment
*/

/*
s = "to be or not to be"

n = 200
*/

REM n = 9999

REM n fkdjfalksjfd
'fdnklfsflsgdngndl dflyjvn

REM This is a test of REM ' x = 2

PRINT "Gloria in exelsis deo."
```

See also ', /*, /**

Restore

Restore VB6! QB!

Restore the internal pointer to the next element of the data array. This is heavily used in old QBasic code.

Example

```
OPTION VERYOLDBASIC

DATA "Salsa"
```

```
READ a$


DATA 22

READ t%

'$END


DATA 66, 77

READ t%, txt

RESTORE
'$END

READ a$, txt

DATA 55, 99

READ t%, txt
READ t%, txt
```

See also Read, Data

Resume

Resume 0 VB6! QB!

Resume {LINENO|LABEL} VB6! QB!

Resume Next VB6! QB!

This is provided for VB6 backward compatibility. You are strongly encouraged to use the new modern exception handling. See Try for more information.

See also On

Return

Return EXPRESSION

- Return a value from current sub or function.

Return VB6! QB!

- Return to the line after GoSub was used. Provided for QBasic compatibility.

Example

```
  Function f()
    Return 12
  End Function
```

See also Sub, Function, GoSub

RSet

RSet STRINGVARIABLE = EXPRESSION VB6! QB!

Provided for QBasic compatibility. The EXPRESSION is aligned right inside the string, if it is a fixed size string of a user defined type.

--------S--------

Select

Select Case EXPRESSION…Case EXPRESSION…End Select

Select Case EXPRESSION…Case EXPRESSION To EXPRESSION…End Select

Select Case EXPRESSION…Case Is OPERATOR EXPRESSION…End Select

Select Case EXPRESSION…Case Else…End Select

The 'Select Case'-statement is much more complicated than the 'If'-statement. In some situations, you may want to compare the same variable or expression with many different values and execute a different piece of code depending on which value it equals to. This is exactly what the 'Select Case'-statement is for. 'Select Case' introduces a multi-line conditional selection statement. The expression given as the argument to the 'Select Case'-statement will be evaluated by 'Case'-statements following. The 'Select Case'-statement concludes with an 'End Select'-statement. As currently implemented, 'Case'-statements may be followed by string values, in this case complex expression can be performed. The test expression can be 'True,' 'False,' a numeric, or string expression. 'Select Case' executes the statements after the 'Case'-statement matching the 'Select Case'-expression, then skips to the 'End Select'-statement. If there is no match and a 'Case Else'-statement is present, then execution defaults to the statements following the 'Case Else.'

Example

```
Dim i As Double
Dim n As Integer

i = 4

Select Case i
Case 0
  n = 0
Case 1, 2
```

```
  n = 1122
Case 4 TO 10
  n = 441000
Case Is = 9
  n = 9999
Case Else
  n = 999999
End Select
```

See also If

Set

Set OBJECTVARIABLE = EXPRESSION VB6! QB!

Assignment-statements give or assign a variable a value or expression with variables, constants, numbers, etc. An assignment always includes a sign.

Use the 'Set' statement in VB6 to assign an object to an object-variable. 'Set' is not needed in KBasic and must not be used.

Shared

Common Shared VARIABLENAME As VARIABLETYPE VB6! QB!

It is provided for old QBasic code.

Signal

Signal SIGNALNAME(ARGUMENTS)…End Signal

Useful for the Qt-Bindings, for defining custom signals. Not implemented yet.

See also Slot

SizeOf

SizeOf(VARIABLE)

Returns the size of the variable in bytes.

See also Len

Slot

Slot SLOTNAME(ARGUMENTS)…End Slot

Useful for the Qt-Bindings, for defining custom slots. See Qt-Bindings example for more information.

See also Signal

Static

Static VARIABLENAME[([Indexes])] [As Typ] [, VARIABLENAME[([Indexes])] [As VARIABLETYPE]]

Before using variables, you must declare them. You must define the name and the data type of a variable. TUse of the 'Static'-Statement Static has three different meanings, depending upon the context used.

- Static inside a class, but outside a method. If you use a 'Static'-statement instead of a 'Dim'-statement, the variable is declared as class variable, meaning it can be used without instance (objects) of this class. A class-variable exists only one time at runtime.
- Static outside a class, but inside a procedure (sub or function) or method. If you use a 'Static'-statement instead of a 'Dim'-statement, the variable is declared as local static variable. The variable, once it has been declared, it is not destroyed by leaving the procedure. The next time the procedure is entered, the value of the variable still exists. Therefore, a local static variable is only one time declared when using recursive calls of a procedure.
- Static outside a class, but before a procedure (sub or function) or method. If you use a 'Static'-statement before the keyword 'Sub' or 'Function' all local declared variables are declared as 'Static' variables, which means that a variable, once it has been declared, it is not destroyed after leaving the procedure. The next time the procedure is entered, the value of the variable still exists. Therefore, a local static variable is only one-time declared when using recursive calls of a procedure.

You can add the keyword 'Static' to other keywords ('Public', 'Protected', 'Private').

See the manual for more information.

Example

```
STATIC SUB myMsgbox(i AS INTEGER)
  DIM s AS STRING

  IF i = 0 THEN s = "Je suis Bernd. Tu't appelles comment?"

  PRINT s

END SUB

myMsgbox (0)
myMsgbox (1)
```

See also Class, Module, Private, Protected, Public, Dim

Step

The statements that control decisions and loops in KBasic are called control structures. Normally every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands depending upon a condition. Some loops repeat commands while a condition is 'True,' other loops repeat commands while a condition is 'False.' There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

For VARIABLENAME = EXPRESSION To EXPRESSION [Step EXPRESSION]…Next

- The For-Next loop is useful when you know how often statements should be repeated. For-Next defines a loop that runs a specified number of times.

Example

```
Dim ctr As Integer

For ctr = 1 To 5
  Print "Z";
Next
```

See also For

Stop

Stop

It ends the execution of a program immediately

Example

```
Dim i As Integer
Stop
i = 2 ' won't be executed, because of Stop in line before
```

See also End

Sub

Sub SUBNAME(ARGUMENTS)…End Sub

A sub-procedure can have arguments, variables, expressions, or constants that are given to the sub-procedure when calling it.

Example

```
' sub example

Sub theMusic
  Print "represents cuba"
  Print "your hips make a shift..."
  Print "I'm the one to find you in the mood..."
  Print "CUBA!"
  Print "represents cuba"
  Print "represents cuba"

End Sub


theMusic() ' first use of sub
theMusic() ' 2nd use
theMusic() ' 3rd use
```

See also Function

Switch

Switch(EXPRESSION, RETURNEXPRESSION[, EXPRESSION, RETURNEXPRESSION, …])

'Switch' returns a value depending on an expression.

Example

```
Dim s As String
Dim i As Integer
i = 1
s = Switch(i = 1, "Bible", i = 2, "Casanova")
Print s
```

See also IIf

System

System

Stops the execution of the program and returns to the operating system. Acts like End or Stop. Provided for QBasic backward compatibility.

See also End, Stop

--------T--------

Text

Option Compare {Binary|Text}

Sets the comparision mode of StrComp.

See also StrComp

Then

If EXPRESSION Then … Else … EndIf

A single decision is used to execute a set of statements if a condition is set ('If'-statement). If the condition is 'True' then the statements after the 'Then' are executed and the statements after the 'Else' are skipped. If the condition is 'False,' the statements after the 'Else' are executed.

Example

```
Dim i As Integer
Dim n As Integer

If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

See also Else, ElseIf, If, EndIf, Select

Throw

Throw OBJECTVARIABLE

It is used to raise an exception, which might change the control flow as defined.

In KBasic, the strange events/errors that may cause a program to fail are called exceptions. Exception handling is an important feature of KBasic. An exception is a signal showing that a non-normal stage has been reached (like an error). To create an exception means to signal this special stage. To catch an exception is to handle an exception; performing some commands to deal with this special stage to return to a normal stage.

Example

```
CLASS rumba

  SUB dance
    PRINT "dance"
  END SUB

END CLASS


PUBLIC SUB test() THROWS rumba
  'EXIT SUB

  THROW NEW rumba  ' return rumba = new rumba
  ' return rumba = 0
END SUB

PUBLIC SUB tt

  test()
  ' 1. if rumba gesetzt, goto catch rumba
  ' goto finally
  ' 2. if throws and if rumba gesetzt, goto parent, throw rumba
CATCH (b AS rumba)
  ' dim b as rumba = rumba
  PRINT "got you!"
  b.dance()
  ' goto finally
FINALLY
  PRINT "will be always executed, whatever happend"

END SUB

tt()
```

See also Try

Throws

Sub SUBNAME() Throws VARIABLETYPE [, VARIABLETYPE…]

Defines any exception the current sub or function might throw. Any exception, which might occur in a sub code, must be catch by a Try…Catch statement or must be thrown by the current sub itself by using Throws.

See also Throw, Try

Timer

On Timer(SECONDS As Long) GoSub LINENO|LABEL VB6! QB!

Calls a gosub sub every seconds. Provided for QBasic backward compability.

Example

```
OPTION VERYOLDBASIC

 ON TIMER(1) GOSUB Update
 TIMER ON
 CLS
 PRINT "Time: "; TIME$
 t = TIMER
 WHILE k < 10
     k = TIMER - t
 WEND
 END

 Update:
     LOCATE 1, 8: PRINT TIME$
     RETURN
```

To

For VARIABLENAME = EXPRESSION To EXPRESSION [Step EXPRESSION]…Next

- The For-Next loop is useful when you know how often statements should be repeated. For-Next defines a loop that runs a specified number of times.

Dim VARIABLENAME[Index To Index, Index To Index…] VB6! QB!

Example

```
Dim ctr As Integer

For ctr = 1 To 5
  Print "Z";
Next
```

See also For, Dim

Try

Try…Catch(VARIABLE AS VARIABLETYPE)…End Catch

It is used for Try Catch, which introduces a exception handling.

Example

```
Try
  test()
Catch (b As rumba)
  Print "tt2: got you!"
  b.dance()
End Catch
```

Type

Type TYPENAME…End Type

A user defined data type is very useful when object-orientated programming is not available. It is like a class, many different variables are held together but without methods. Many kind of data types are allowed inside a user defined data type, even other user defined data types can be included.

Example

```
TYPE book
  bkname AS STRING * 100

  isbn(1000) AS INTEGER
```

```
END TYPE

TYPE address
  a(50) AS book
  age AS INTEGER
  name AS STRING * 100
  nn(100) AS INTEGER
END TYPE

DIM j(10) AS address


j(6).nn(99) = 123
j(6).a.isbn(10) = 1000
j(0).nn(0) = j(6).nn(99) + j(6).a.isbn(10)

PRINT j(0).nn(0)
```

See also Enum

TypeOf

TypeOf VARIABLENAME Is CLASSNAME

Returns true, if variable is an object of that class.

See also Is

--------U--------

UBound

UBound(VARIABLENAME [, Index])

Getting upper bound of an array. LBound is used for the upper bound.

Example

```
TYPE book
  bkname AS STRING * 100

  isbn(1000) AS INTEGER
END TYPE

TYPE address
  books(50) AS book
  age AS INTEGER
  name AS STRING * 100
END TYPE

DIM j(5 TO 10) AS book
```

```
PRINT UBOUND(j, 1)
```

See also LBound

Until

Do…Loop Until EXPRESSION

Do While EXPRESSION…Loop

Loop-statements

The statements that control decisions and loops in KBasic are called control structures. Normally every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands depending upon a condition. Some loops repeat commands while a condition is ‘True,’ other loops repeat commands while a condition is ‘False.’ There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

Use the following loops when you are not sure how often a command should be repeated: ‘Do’, ‘While’, ‘Loop’, ‘Until’ or (’Wend’) . There are two different ways to use the keyword ‘While’ in order to test a condition within a ‘Do…Loop’-statement. You can test the condition before the commands inside the loop are executed or you can test the condition after the commands of the loop have been executed at least once. If the condition is ‘True’ (in the following procedure ‘SubBefore’) the commands inside the loop execute.

Example

```
Sub SubBefore()
  Counter = 0
  myNumber = 20
  Do While myNumber > 10
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop
  MsgBox "Loop has been executed " & Counter & " time(s)."
End Sub

Sub SubAfter()
  Counter = 0
  myNumber = 9
  Do
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop While myNumber > 10
  MsgBox "Loop has been executed " & Counter & " time(s)."
End Sub
```

See also While, Loop, Until, For

--------V--------

VeryOldBasic

Option VeryOldBasic VB6! QB!

Using one of the following commands you can switch KBasic modes: For very old BASIC code, like QBasic, turn it on.

Example

```
Option VeryOldBasic
Print 21
```

See also Option, KBasic, OldBasic

--------W--------

WEnd

While…WEnd VB6! QB!

Provided for VB6 backward compatibility.

See also Do, Loop

While

While…WEnd VB6! QB!

- Provided for VB6 backward compatibility.

While…End While VB6! QB!

- Provided for VB.NET compatibility.

Do While EXPRESSION…Loop

- Loop-statement. See Do fore more information.

See also Do, Loop

With

With…End With

With-Statement A very useful statement is the 'With'-statement. When using the 'With'-statement, you are able to group assignments or statements that reference the same object. This makes your code more readable in addition to reducing redundant code.

Example

```
TYPE book
  bkname AS STRING * 100
  isbn(1000) AS INTEGER
END TYPE


TYPE zoo
  e AS book
END TYPE


DIM j(1 TO 10) AS zoo



WITH j(3)

  .e.bkname = "Frankfurter Zoo"

  WITH .e
    . isbn ( 99 ) = 333
  END WITH

END WITH

PRINT j(3).e.bkname
PRINT j(3).e.isbn(99)



END



CLASS rumba

  PUBLIC SUB dance_rumba()
    PRINT "rumba!!!"

    WITH ME
      .test()
    END WITH

  END SUB

  PRIVATE SUB test()
    PRINT "test"
  END SUB

END CLASS


DIM m AS NEW rumba
```

```
WITH m
     .dance_rumba()
 /*jjj*/  '   .dance_rumba()
'      .dance_rumba() :.dance_rumba()
END WITH
```