Syntax

Table of Contents

First Edition

This edition applies to release 1.6 of KBasic and to all subsequent released and modfications until otherwise indicated in new editions. Make sure you are using the correct edtion for the level of the product. The term „KBasic“ as used in this publication, refers to the kBasic product set (January 2007).

Conventions Used In This Book / Way Of Writing

normal text appears in writing Arial. Here is an example here: This is normal text

Syntax and source code code appear in writing Courier New. Here the example:

```
Dim i As Integer
```

Important references and keywords are italically deposited: *Arguments*

KBasic-Syntax

The syntax of sub, function or statement in the KBasic help entry shows all elements, which are needed to correctly use the sub, function or statement. How you can understand those information shows the following lines.

Example: Syntax of the MsgBox-Function

MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Arguments, which are inside of [ ], are optional. (Do not write these [ ] in your KBasic code). The only argument, what you have give the MsgBox-Function is the one for the showing the text: 'prompt'.

Arguments for functions or subs can be used with the help of their position or their name. In order to use the arguments defined with their position, you do not have to ignore the position written in the syntax. You must write them exactly in the same order they occur in the syntax. All arguments must be separated by a comma. Example:

```
MsgBox("The answer is right!", 0, "window with answer")
```

If you would like to use a argument with its name, use the name of the argument and colon and equals sign (:= and the value of the argument. You can write these named arguments in any order you wish. Example:

```
MsgBox(title:="window with answer", prompt:="The answer is right!")
```

Some arguments are written inside of {} in the syntax of functions or subs.

Option Compare {Binary | Text}

In the syntax of the 'Option Compare'-statement: { } together with | means that one of the elements must be written.(Do not write these { } in your KBasic code). The following statement defines that text will be compared and sorted without case sensitive.

```
Option Compare Text
```

Syntax of the 'Dim'-Statement

Dim VarName[([Indexes])] [As Type] [, VarName[([Indexes])] [As Type]] …

'Dim' is a keyword in the syntax of the 'Dim'-Statement. The only needed element is VarName (the name of the variable). The following statement creates three variables: myVar, nextVar and thirdVar. These variables are declared as 'Variant'-variables automatically (or 'Double' in 'VeryOldBasic Mode').

```
Dim myVar, nextVar, thirdVar
```

The following example declares a variable of type 'String'. If you declared the datatype of the variable explicitly, it will help KBasic to optimize the RAM-usage and will help you to find errors in your code.

```
Dim myAns As String
```

If you want to declare many variables in one line, you should declare every datatype of each variable explicitly. Variables without declared datatype get the default datatype, which is 'Variant'.

```
Dim x As Integer, y As Integer, z As Integer
```

X and y get in the datatype 'Variant' in the following statement. Only z has the 'Integer' datatype.

```
Dim x, y, z As Integer
```

You have to put ( ) or [ ] (for new style), if you want to declare an array variable. The indexes of the array are optional. The following statement declares a dynamic array named myArray.

```
Dim myArray[]
```

Variable

Declaration

Dim

Public

Private

Protected

Static

As

```
Dim sName As String

Public sName As String

Private sName As String

Protected sName As String


Dim Name[([Index])] [As Type] [, Name[([Index])] [As Type]] ...

Dim Name [= Expression] [As Type]

Dim Name [As Type] [= Expression]
```

```
[Public | Protected | Private | Dim | Static] Name [= Expression] [As Type]
```

Assignment

```
Dim yourName As String
yourName = InputBox("What is your name?")
MsgBox "Your Name is " & yourName
```

User Defined Type

Type

```
Type Name
 Name [(Index)] As Type
  ...
End Type
```

Comment

REM

'

/*

*/

/**

```
REM this is a comment

‘ this is a comment as well

/* start comment and stop comment */

/** start documentation comment and stop documentation comment */
```

Literal

Byte, Short, Integer, Long

```
1, 2, -44, 4453, +78
```

Hex

```
&HAA43
```

Binary

```
&B11110001
```

Octal

```
&O1234
```

Single (Decimal)

```
21.32, 0.344, -435.235421.21, +67.8
```

Double (Decimal)

```
212.23
```

Currency

```
45.3@
```

Date, Time

```
#1993-12-31#
```

String

```
„hello‟
```

Boolean

```
True, False
```

Constant

Const

As

```
Const Border As Integer = 377
```

```
Const Name = Expression

Const Name [As Type] = Expression [, Name [As Type] = Expression] ...

[Public | Protected | Private] Const Name [As Type] = Expression
```

Working With Objects

Create Object

New

```
s = New Control()

s = New Timer(start, ende)

objectVariable = New ClassName[(Arguments)]

objectVariable = New ClassName()

objectVariable = New ClassName
```

Create Class

```
Class oak Inherits tree

Variables / Constants / Properties / Types / Enumerations
Constructors
Destructors
Functions
Subs

End Class
```

Access Class Variable And Instance Variable

```
classname.classVariable
objectname.instanceVariable
```

Access Class Method Or Intance Method (Function Or Sub)

```
objectname.instanceVariable = 99
```

Class Method Or Intance Method

```
Static Sub myClassMethod()
...
End Sub

Sub myInstanceMethod
...
End Sub
```

Access Class Type

```
objectname.typefield
```

Access Class Enum

```
objectname.enumfield
```

Access Class Property

```
objectname.classproperty
```

Call Method

```
objectname.myMethod()
```

Current Instance Of Object

Me

Parent

```
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()
```

Class Of Object

TypeOf

```
If TypeOf myObject Is myClass Then
```

```
TypeOf objectVariable Is ClassName
```

Hidden Variable

```
Parent.myVariable  ' access parent class
myVariable         ' access current class (me)
```

Hidden Method (Sub Or Function)

```
Parent.myMethod()  ' access parent class
myMethod           ' access current class (me)
```

Overwrite Method

```
Class A
  Dim i As Integer

  Function f()
    Return i
  End Function

End Class

Class B Inherits A
  Dim i As Integer    ' hides variable i in A

  Function f()        ' overwrites method f() in A
    i = Parent.i + 1       ' access A.i
    Return Parent.f() + i  ' acces A.F()
  End Function
End Class
```

Scope modifier

Private

Protected

Public

```
Class plane

  Private wings As Integer
  Protected wings2 As Integer

  Private Function countWings()
    ...
  End Function

End Class
```

Array

Dim

```
Dim variableName(Index) As Type  ' old style

Dim variableName[Index] As Type

Dim variableName[Index, Index, ...] As Type

Dim variableName[Index To Index] As Type

Dim variableName[Index To Index, Index To Index, ...] As Type
```

Access Array

```
i(3) = 10 ' old style
i[3] = 10
o[3, 88] = 10
```

Lower And Upper Bound Of Array

UBound

LBound

```
UBound (arrayVariable[,(Dimension])

LBound (arrayVariable[,(Dimension])
```

Dim With Explicit Lower Bound

```
Dim i [50 To 100] As Integer
```

Multi-Dimension

```
Dim i(100, 50, 400)

Dim sngMulti(1 To 5, 1 To 10) As Single
```

Dynamic Array

```
Dim a() As Integer
Redim

Redim variableName(Index) ' old style

Redim variableName[Index]

Redim variableName[Index, Index, ...]
Redim variableName[Index To Index]

Redim variableName[Index To Index, Index To Index, ...]
```

Delete Array

Reset Array

Erase

```
Erase arrayVariable[, arrayVariable]
```

Flow Control - Decision

Single Decision

If

Then

Else

End If

```
If Expression Then Statement

If Expression Then Statement : Else Statement

If Expression Then LineNo

If Expression Then LabelName:

If Expression Then
  [Statements]
End If

If Expression Then
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
ElseIf Expression
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
```

```
ElseIf Expression
  [Statements]
ElseIf Expression
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
ElseIf Expression
  [Statements]
End If
```

IIf – Short If

```
IIf(Expression, ThenReturnExpression, ElseReturnExpression)
```

Multi Decision

Select Case

Case

End Select

```
Select Case Expression
Case Expression
  [Statements]
Case Expression
  [Statements]
End Select

Select Case Expression
Case Expression
  [Statements]
Case Expression To Expression
  [Statements]
Case Is Expression
  [Statements]
Case Else
  [Statements]
End Select
```

Switch – Short Select Case

```
Switch(Expression, ReturnExpression[, Expression, ReturnExpression, ... ])
```

Choose – Short Select Case

```
Choose(Expression, ReturnExpression[, ReturnExpression, ... ])
```

Uncoditional Jump

GoTo

```
GoTo {lineno | label:}

GoTo myExit:
GoTo nextStep:
```

With

```
Sub FormatOrder ()
        With myclass.
                .Value = 30
                .Font.Bold = True
        End With
End Sub

Sub setValue ()
        With j(3)
          .e.bkname = "Frankfurter Zoo"
          With .e
            .isbn ( 99 ) = 333
          End With
        End With
End Sub
```

Edit

Flow Control - Loop

For Next

To

Step

```
For variable = beginExpr To endExpr [Step Expression]
  [Statements]
Next [variable]
```

Do While ... Loop

```
Do While Expression
  [Statements]
Loop
```

Do ... Loop Until

```
Do
  [Statements]
Loop Until Expression
```

Do ... Loop While

```
Do
  [Statements]
Loop While Expression
```

Do Until ... Loop

```
Do Until Expression
  [Statements]
Loop
```

While ... Wend

```
While Expression
  [Statements]
WEnd
```

While ... End While

```
While Expression
  [Statements]
End While
```

Explicit Leave Of Loop

```
Exit For
Exit Do
```

Explicit Test of Loop Condition

```
Iterate For
Iterate Do
```

Subs / Procedures

Sub-Procedure

Sub

End Sub

```
Sub Name([Argumente])
  [Statements]
End Sub

Sub Name([Argumente]) [Throws Name, ...]
  [Statements]
End Sub
```

Function-Procedure

Function

End Function

```
Function Name([Argumente]) [As Type]
  [Statements]
End Function

Function Name([Argumente]) [As Type] [Throws Name, ...]
  [Statements]
End Function
```

Argument

```
Name As Type

[ByVal | ByRef] Name As Type

[ByVal | ByRef] Name [As Type]

[ByVal | ByRef] Name [()][As Type]

[ByVal | ByRef] [Optional] Name [()][As Type] [= Expression]
```

Named Argument

Optional Argument

```
Sub PassArg(strName As String, intAlter As Integer, gebDatum As Date)
        Print strName, intAlter, gebDatum
End Sub

PassArg(Frank", 26, #2-28-79#)

PassArg(intAlter:=26, gebDatum:=#2/28/79#, strName:="Frank")

MsgBox(Title:="Aufgabe-Dialogfeld", Prompt:="Aufgabe erledigt!")

Sub OptionaleArg(strPLBereich As String, Optional strLand As String =
"Deutschland")
...
End Sub


Sub OptionaleArg(strLand As String, Optional intZBezirk As Integer, _
Optional strLand As String = "Deutschland")
        If IsMissing(intZBezirk) And IsMissing(strLand) Then
                Print strPLBereich
        ElseIf IsMissing(strLand) Then
                Print strPLBereich, intZBezirk
        ElseIf IsMissing(intZBezirk) Then
                Print strPLBereich, strLand
        Else
```

```
                Print strPLBereich, intZBezirk, strLand
        End If
End Sub
```

Default Argument

```
Sub OptionaleArg(strLand As String = "Deutschland")
  Print strLand ' even strLand is not passed it contains Deutschland
End Sub
```

ParamArray

```
Sub nadja(ByRef z As Integer, ByVal h As Double, Optional j As Integer,
ParamArray b() As Variant)

  Print "z = " + z
  Print "h = " + h
  If Not IsMissing(j) Then
    Print "j = " + j
  End If

  Dim i As Integer

  For i = LBound(b) To UBound(b)
    Print "b(" + i + ") = " + b(i)
  Next i

End Sub


nadja(j := 33, h := 12.2, z := m, b[12] := "12 hello", b[5] := 555, b[7] := "7
ho")
```

Call Of Sub or Function

```
Sub Main()
        MultiBeep 56
        Meldung
End Sub

Sub MultiBeep(Anzahl)
        For n As Integer = 1 To Anzahl
                Beep
        Next n
End Sub

Sub Meldung()
        MsgBox "Zeit für eine Pause!"
End Sub
```

Explicit Leave Of Procedures

```
Exit Sub
Exit Function
```

Functions

Function

End Function

```
Function Name([Argumente])[As Type]
  [Statements]
End Function

Function Name([Argumente])[As Type] [Throws Name, ...]
  [Statements]
End Function
```

Return Function Value

```
Return Expression
```

Return Expression

```
FunctionName = Expression
```

Property

Access Property

```
varname.classproperty = 99
Print varname.classproperty
```

Property

Property Set

Property Get

Set

End Set

Get

End Get

End Property

```
Property Set Name(Argument)
  [Statements]
```

```
End Property

Property Get Name(Argument) As Type
  [Statements]
End Property


Property Name As Type

  Get
    [Statements]
  End Get

  Set(Argument)
    [Statements]
  End Set

End Property
```

User defined Type

Access Type

```
Type
  varname.typefield = 99
End Type

Type Name
 Name [(Index)] As Type
  ...
End Type
```

Enumeration

Access Enum

```
varname.enumfield = 99

Enum
End Enum


Enum Name
  Name [= Expression]
  ...
End Enum
```

Class

Class

Abstract

Inherits

Constructor

Destructor

Sub

Function

Signal

Slot

End Class

```
[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
```

Module

Create Module

```
Module oak

Variables / Constants / Types / Enumerations
Functions
Subs

End Module
```

Access Module Variable

```
modulename.moduleVariable
moduleVariable
```

Access Module Sub Or Function

```
modulename.moduleSub(99)
```

Module Sub Or Function

```
Sub myModuleSub
...
End Sub

Function myModuleFunction
...
End Function
```

Call Module Sub Or Function

```
modulename.myModuleSub()
```

Access Module Type

```
modulename.typefield
```

Access Module Enum

```
modulename.enumfield
```

Module

End Module

```
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
```

Error Handling

New Exception

Throw

```
Throw ExceptionObject
```

Exception

Try

Catch

End Catch

```
Try
  [Statements]
Catch (Name As Exception)
  [Statements]
End Catch

Try
  [Statements]
Catch (Name As Exception)
  [Statements]
Catch (Name As Exception)
  [Statements]
End Catch

Try
  [Statements]
Catch (Name As Exception)
  [Statements]
Finally
  [Statements]
End Catch
```

Exception In Procedure (Sub Or Function)

```
Catch (Name As Exception)
  [Statements]
Finally
  [Statements]
End Catch
```