B4x Booklets

B4x Basic Language

Edition 1.0

Last update : 2017.05.25

Main contributors: Klaus Christl (klaus), Erel Uziel (Erel)

To search for a given word or sentence use the Search function in the Edit menu.

Updated for following versions:
B4A version 7.00
B4i version 4.01
B4J version 5.51
B4R version 1.80

1 BASIC

BASIC (an acronym for **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use. In 1964, John G. Kemeny and Thomas E. Kurtz designed the original BASIC language at Dartmouth College in the U.S. state of New Hampshire. They wanted to enable students in fields other than science and mathematics to use computers. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to learn (source Wikipedia).

1.1 B4x

B4X is a suite of rapid application development IDEs that allows the creation of applications the following platforms: Google Android, Apple iOS, Java, Raspberry Pi and Arduino. B4X uses a Proprietary dialect of Visual Basic (hereinafter called "B4X") and also features a visual designer that simplifies the process of creating user interfaces.

1.1.1 B4A Android

B4A – The simple way to develop native Android apps.
B4A includes all the features needed to quickly develop any type of Android app.
B4A is used by tens of thousands of developers from all over the world, including companies such as NASA, HP, IBM and others.
Together with B4i you can now easily develop applications for both Android and iOS.

1.1.2 B4i iOS

B4i – The simple way to develop native iOS apps.
B4i is a development tool for native iOS applications.
B4i follows the same concepts as B4A, allowing you to reuse most of the code and build apps for both Android and iOS.
It is the only development tool that allows you to develop 100% native iOS apps without a local Mac computer.

1.1.3 B4J Java Desktop

B4J – Modern “VB6 like” development tool for cross platform desktop, server and IoT solutions
B4J is a **100% free** development tool for desktop, server and IoT solutions.
With B4J you can easily create desktop applications (UI), console programs (non-UI) and server solutions.
The compiled apps can run on Windows, Mac, Linux and ARM boards (such as Raspberry Pi).

1.1.4 B4R Arduino

B4R – Easily build native Arduino & ESP8266 programs.
B4R is a **100% free** development tool for native Arduino and ESP8266 programs.
B4R follows the same concepts of the other B4X tools, providing a simple and powerful development tool.
B4R, B4A, B4J and B4i together make the best development solution for the Internet of Things (IoT).

2 Variables and objects

A **variable** is a symbolic name given to some known or unknown quantity or information, for the purpose of allowing the name to be used independently of the information it represents. A variable name in computer source code usually associated with a data storage location and thus also its contents, and these may change during the course of program execution (source Wikipedia).

There are two types of variables: primitives and non-primitives types.
Primitives include the numeric types: Byte, Short, Int, Long, Float and Double.
Primitives also include: Boolean and Char.

2.1 Variable Types

B4A, B4i, B4J

List of types with their ranges:

B4x	Type	min value	max value
Boolean	boolean	False	True
Byte	integer 8 bits	-2^7 -128	$2^7 - 1$ 127
Short	integer 16 bits	-2^{15} - 32768	$2^{15}-1$ 32767
Int	integer 32 bits	-2^{31} -2147483648	$2^{31}-1$ 2147483647
Long	long integer 64 bits	-2^{63} -9223372036854775808	$2^{63}-1$ 9223372036854775807
Float	floating point number 32 bits	-2^{-149} 1.4E-45	$(2-2^{-23}) * 2^{127}$ 3.4028235 E 38
Double	double precision number 64 bits	-2^{-1074} 2.2250738585072014 E -308	$(2-2^{-52}) * 2^{1023}$ 1.7976931348623157 E 308
Char	character		
String	array of characters		

B4R

List of types with their ranges:
Numeric types:

Byte 0 - 255
Int (2 bytes) -32768 - 32768. Similar to Short type in other B4x tools.
UInt (2 bytes) 0 - 65535. B4R specific.
Long (4 bytes) -2,147,483,648 - 2,147,483,647. Similar to Int type in other B4x tools.
ULong (4 bytes) 0 - 4,294,967,295 B4R specific.
Double (4 bytes) 4 bytes floating point. Similar to Float in other B4x tools.
Float is the same as Double. Short is the same as Int.

The above is true on all boards, including the Arduino Due.

Other types:
Boolean True or False. Practically it is saved as a byte with the value of 1 or 0.
String Strings are made from an array of bytes that end with a null byte (byte with the value of 0).
Object Objects can hold other types of values.

Primitive types are always passed by value to other subs or when assigned to other variables.
For example:

```
Sub S1
   Private A As Int
   A = 12                  The variable A = 12
   S2(A)                   It's passed by value to routine S2
   Log(A)  ' Prints 12     Variable A still equals 12, even though B was changed in routine S2.
End Sub

Sub S2(B As Int)           Variable B = 12
   B = 45                  Its value is changed to B = 45
End Sub
```

All other types, including arrays of primitive types and strings are categorized as non-primitive types.
When you pass a non-primitive to a sub or when you assign it to a different variable, a copy of the reference is passed.
This means that the data itself isn't duplicated.
It is slightly different than passing by reference as you cannot change the reference of the original variable.

All types can be treated as Objects.
Collections like lists and maps work with Objects and therefore can store any value.
Here is an example of a common mistake, where the developer tries to add several arrays to a list:

```
Private arr(3) As Int
Private List1 As List
List1.Initialize
For i = 1 To 5
   arr(0) = i * 2
   arr(1) = i * 2
   arr(2) = i * 2
   List1.Add(arr)    'Add the whole array as a single item
Next
arr = List1.Get(0)   'get the first item from the list
Log(arr(0))          'What will be printed here???
```

You may expect it to print 2. However it will print 10.
We have created a single array and added 5 references of this array to the list.
The values in the single array are the values set in the last iteration.
To fix this we need to create a new array each iteration.
This is done by calling Private each iteration:

```
Private arr(3) As Int 'This call is redundant in this case.
Private List1 As List
List1.Initialize
For i = 1 To 5
   Private arr(3) As Int
   arr(0) = i * 2
   arr(1) = i * 2
   arr(2) = i * 2
   List1.Add(arr) 'Add the whole array as a single item
Next
arr = List1.Get(0) 'get the first item from the list
Log(arr(0)) 'Will print 2
```

2.2 Names of variables

It is up to you to give any name to a variable, except reserved words.
A variable name must begin with a letter and must be composed by the following characters A-Z, a-z, 0-9, and underscore "_", no spaces, no brackets etc.
Variable names are case insensitive, that means that Index and index refer to the same variable.

But it is good practice to give them meaningful names.
Example:

Interest = Capital * Rate / 100	is meaningful
n1 = n2 * n3 / 100	not meaningful

For Views (B4A, B4i), Nodes (B4J), it is useful to add to the name a three character prefix that defines its type.
Examples:

`lblCapital`	lbl > Label	Capital > purpose
`edtInterest`	edt > EditText	Interest > purpose
`btnNext`	btn > Button	Next > purpose

2.3 Declaring variables

2.3.1 Simple variables

Variables are declared with the `Private` or the `Public` keyword followed by the variable name and the `As` keyword and followed by the variable type. For details look at chapter Scope.
There exist the `Dim` keyword, this is maintained for compatibility.

Examples:

```
Private Capital As Double
Private Interest As Double
Private Rate As Double
```
Declares three variables as Double, double precision numbers.

```
Private i As Int
Private j As Int
Private k As Int
```
Declares three variables as Int, integer numbers.

```
Private lblCapital As Label
Private lblInterest As Label
Private lblRate As Label
```
Declares three variables as Label views.

```
Private btnNext As Button
Private btnPrev As Button
```
Declares two variables as Button views.

The same variables can also be declared in a short way.

```
Private Capital, Interest, Rate As Double
Private i, j, k As Int
Private lblCapital, lblInterest, lblRate As Label
Private btnNext, btnPrev As Button
```

The names of the variables separated by commas and followed by the type declaration.

Following variable declarations are valid:

```
Private i = 0, j = 2, k = 5 As Int

Private txt = "test" As String, value = 1.05 As Double, flag = False As Boolean
```

View names must be declared if we want to use them in the code.
For example, if we want to change the text in a Label view in the code, like
`lblCapital.Text = "1200"`,
we need to reference this Label view by its name `lblCapital`, this is done with the `Private` declaration.
If we never make any reference to this Label view anywhere in the code no declaration is needed.
Using an event routine for that view doesn't need a declaration either.

To allocate a value to a variable write its name followed by the equal sign and followed by the value, like:

```
Capital = 1200
LastName = "SMITH"
```

Note that for `Capital` we wrote just `1200` because `Capital` is a number.
But for `LastName` we wrote `"SMITH"` because `LastName` is a string.
Strings must always be written between double quotes.

2.3.2 Array variables

Arrays are collections of data or objects that can be selected by indices. Arrays can have multiple dimensions.
The declaration contains the `Private` or the `Public` keyword followed by the variable name `LastName`, the number of items between brackets `(50)`, the keyword `As` and the variable type `String`.
For details look at chapter Scope. There exist the `Dim` keyword, this is maintained for compatibility.

Note: B4R supports only single dimension arrays !

Examples:

`Public LastName(50) As String`	One dimension array of strings, total number of items 50.
`Public Matrix(3, 3) As Double`	Two dimensions array of Doubles, total number of items 9.
`Public Data(3, 5, 10) As Int`	Three dimensions array of integers, total number of items 150.

The first index of each dimension in an array is 0.

```
LastName(0), Matrix(0,0), Data(0,0,0)
```

The last index is equal to the number of items in each dimension minus 1.

```
LastName(49), Matrix(2,2), Data(2,4,9)
```

```
Public LastName(10) As String
Public FirstName(10) As String
Public Address(10) As String
Public City(10) As String
```

or

```
Public LastName(10), FirstName(10), Address(10), City(10) As String
```

This example shows how to access all items in a three dimensional array.

```
Public Data(3, 5, 10) As Int

For i = 0 To 2
   For j = 0 To 4
      For k = 0 To 9
         Data(i, j, k) = ...
      Next
   Next
Next
```

A more versatile way to declare arrays is to use variables.

```
Public NbPers = 10 As Int
Public LastName(NbPers) As String
Public FirstName(NbPers) As String
Public Address(NbPers) As String
Public City(NbPers) As String
```

We declare the variable `Public NbPers = 10 As Int` and set its value to 10.
Then we declare the arrays with this variable instead of the number 10 as before.
The big advantage is if at some point we need to change the number of items, we change only ONE value.

For the Data array we could use the following code.

```
Public NbX = 2 As Int
Public NbY = 5 As Int
Public NbZ = 10 As Int
Public Data(NbX, NbY, NbZ) As Int
```

And the access routine.

```
For i = 0 To NbX - 1
   For j = 0 To NbY - 1
      For k = 0 To NbZ - 1
         Data(i, j, k) = ...
      Next
   Next
Next
```

Filling an array with the Array keyword :

```
Public Name() As String
Name = Array As String("Miller", "Smith", "Johnson", "Jordan")
```

2.3.3 Array of views / nodes (objects)

Views / nodes or objects can also be in an Array. The following code shows an example:
In B4A and B4i user interface objects are called *views* and called *nodes* in B4J.

In the example below the Buttons are added to the parent view / node by code.

B4A

```
Sub Globals
   Private Buttons(6) As Button
End Sub

Sub Activity_Create(FirstTime As Boolean)
   Private i As Int

   For i = 0 To 5
      Buttons(i).Initialize("Buttons")
      Activity.AddView(Buttons(i), 10dip, 10dip + i * 60dip, 150dip, 50dip)
      Buttons(i).Tag = i + 1
      Buttons(i).Text = "Test " & (i + 1)
   Next
End Sub

Sub Buttons_Click
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

B4i

```
Sub Process_Globals

   Private Buttons(6) As Button
End Sub

Private Sub Application_Start (Nav As NavigationController)

   Private i As Int
   For i = 0 To 5
      Buttons(i).Initialize("Buttons")
      Page1.RootPanel.AddView(Buttons(i), 10dip, 10dip + i * 60dip, 150dip, 50dip)
      Buttons(i).Tag = i + 1
      Buttons(i).Text = "Test " & (i + 1)
   Next
End Sub

Sub Buttons_Click
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

B4J

```
Sub Process_Globals

   Private Buttons(6) As Button
End Sub

Sub AppStart (Form1 As Form, Args() As String)

   Private i As Int
   For i = 0 To 5
      Buttons(i).Initialize("Buttons")
      MainForm.RootPane.AddNode(Buttons(i), 10, 10 + i * 60, 150, 50)
      Buttons(i).Tag = i + 1
      Buttons(i).Text = "Test " & (i + 1)
   Next
End Sub

Sub Buttons_MouseClicked (EventData As MouseEvent)
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

The Buttons could also have been added in a layout file, in that case they must neither be initialized, nor added to the parent view / node and the Text and Tag properties should also be set in the Designer.
In that case the code would look like this:

B4A

```
Sub Globals
   Private b1, b2, b3, b4, b5, b6, b7 As Button
   Private Buttons() As Button
End Sub

Sub Activity_Create(FirstTime As Boolean)

   Buttons = Array As Button(b1, b2, b3, b4, b5, b6, b7)
End Sub

Sub Buttons_Click
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

B4i

```
Sub Process_Globals

   Private b1, b2, b3, b4, b5, b6, b7 As Button
   Private Buttons(6) As Button
End Sub

Private Sub Application_Start (Nav As NavigationController)

   Buttons = Array As Button(b1, b2, b3, b4, b5, b6, b7)
End Sub

Sub Buttons_Click
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

B4J

```
Sub Process_Globals

   Private b1, b2, b3, b4, b5, b6, b7 As Button
   Private Buttons(6) As Button
End Sub

Sub AppStart (Form1 As Form, Args() As String)

   Buttons = Array As Button(b1, b2, b3, b4, b5, b6, b7)
End Sub

Sub Buttons_MouseClicked (EventData As MouseEvent)
   Private btn As Button
   btn = Sender
   Log("Button " & btn.Tag & " clicked")
End Sub
```

2.3.4 Type variables B4A, B4i and B4J only

A Type cannot be private. Once declared it is available everywhere (similar to Class modules).
The best place to declare them is in the Process_Globals routine in the Main module.

Let us reuse the example with the data of a person.
Instead of declaring each parameter separately, we can define a personal type variable with the Type keyword:

```
Public NbUsers = 10 As Int
Type Person(LastName As String, FirstName As String. Address As String, City As String)
Public User(NbUsers) As Person
Public CurrentUser As Person
```

The new personal type is `Person` , then we declare either single variables or arrays of this personal type.
To access a particular item use following code.

```
CurrentUser.FirstName
CurrentUser.LastName

User(1).LastName
User(1).FirstName
```

The variable name, followed by a dot and the desired parameter.
If the variable is an array then the name is followed by the desired index between brackets.

It is possible to assign a typed variable to another variable of the same type, as shown below.

```
CurrentUser = User(1)
```

2.4 Casting

B4x casts types automatically as needed. It also converts numbers to strings and vice versa automatically.
In many cases you need to explicitly cast an Object to a specific type.
This can be done by assigning the Object to a variable of the required type.
For example, Sender keyword references an Object which is the object that raised the event.
The following code changes the color of the pressed button.
Note that there are multiple buttons that share the same event sub.

```
Sub Globals
   Private Btn1, Btn2, Btn3 As Button
End Sub

Sub Activity_Create(FirstTime As Boolean)
   Btn1.Initialize("Btn")
   Btn2.Initialize("Btn")
   Btn3.Initialize("Btn")
   Activity.AddView(Btn1, 10dip, 10dip, 200dip, 50dip)
   Activity.AddView(Btn2, 10dip, 70dip, 200dip, 50dip)
   Activity.AddView(Btn3, 10dip, 130dip, 200dip, 50dip)
End Sub

Sub Btn_Click
   Private btn As Button
   btn = Sender     ' Cast the Object to Button
   btn.Color = Colors.RGB(Rnd(0, 255), Rnd(0, 255), Rnd(0, 255))
End Sub
```

The above code could also be written more elegantly:

```
Sub Globals

End Sub

Sub Activity_Create(FirstTime As Boolean)
   Private i As Int
   For i = 0 To 9   ' create 10 Buttons
      Private Btn As Button
      Btn.Initialize("Btn")
      Activity.AddView(Btn, 10dip, 10dip + 60dip * i, 200dip, 50dip)
   Next
End Sub

Sub Btn_Click
   Private btn As Button
   btn = Sender     ' Cast the Object to Button
   btn.Color = Colors.RGB(Rnd(0, 255), Rnd(0, 255), Rnd(0, 255))
End Sub
```

2.5 Scope

2.5.1 Process variables

These variables live as long as the process lives.
You should declare these variables inside Sub Process_Globals.
This sub is called once when the process starts (this is true for all modules, not just the main module).
These variables are the only "public" variables. Which means that they can be accessed from other modules as well.

However, in B4A, not all types of objects can be declared as process variables.
For example, views / nodes cannot be declared as process variables.
The reason is that we do not want to hold a reference to objects that should be destroyed together with the activity.
In other words, once the activity is being destroyed, all of the views which are contained in the activity are being destroyed as well.
If we hold a reference to a view, the garbage collector would not be able to free the resource and we will have a memory leak. The compiler enforces this requirement.

To access process global variables in other modules than the module where they were declared their names must have the module name they were declared as a prefix.
Example:
Variable defined in a module with the name : *MyModule*

```
Sub Process_Globals
   Public MyVar As String
End Sub
```

Accessing the variable in *MyModule* module:

```
   MyVar = "Text"
```

Accessing the variable in any other module:

```
   MyModule.MyVar = "Text"
```

Variables can be declared with:

```
Dim MyVar As String
```

In this case the variable is public same as Public.

It is good practice to declare the variables like this:

```
Public MyVar As String
```

This variable is public.

It is possible to declare private variables in Sub Process_Globals like this:

```
Private MyVar As String
```

The variable is private to the activity or the module where it is declared.
For Activities it is better to declare them in Sub Globals.

For variables declared in Class modules in Sub Class_Globals the same rules as above are valid.

```
   Public MyVarPublic As String         ' public
   Private MyVarPublic As String        ' private
   Dim MyVar As String                  ' public like Public
```

Using Dim in Sub Class_Globals is not recommended !

2.5.2 Activity variables B4A only

These variables are contained by the activity.
You should declare these variables inside Sub Globals.
These variables are "private" and can only be accessed from the current activity module.
All object types can be declared as activity variables.
Every time the activity is created, Sub Globals is called (before Activity_Create).
These variables exist as long as the activity exists.

2.5.3 Local variables

Variables declared in a subroutine are local to this subroutine.
They are "private" and can only be accessed from within the subroutine where they were declared.
All objects types can be declared as local variables.
At each call of the subroutine the local variables are initialized to their default value or to any other value you have defined in the code and are 'destroyed' when the subroutine is exited.

2.6 Tips

A view / node can be assigned to a variable so you can easily change the common properties of the view.

For example, the following code disables all views that are direct children of a Panel / Pane:

```
For i = 0 To MyPanel.NumberOfViews - 1
   Private v As View
   v = MyPanel.GetView(i)
   v.Enabled = False
Next
```

If we only want to disable buttons:

```
For i = 0 To.NumberOfViews - 1
   Private v As View
   v = MyPanel.GetView(i)
   If v Is Button Then ' check whether it is a Button
      v.Enabled = False
   End If
Next
```

Note: `MyPanel` is a *Panel* in B4A and B4i but it is a *Pane* in B4J.

3 Basic language

3.1 Expressions

An expression in a programming language is a combination of explicit values, constants, variables, operators, and functions that are interpreted according to the particular rules of precedence and of association for a particular programming language, which computes and then produces (returns) another value. This process, like for mathematical expressions, is called evaluation. The value can be of various types, such as numerical, string, and logical (source Wikipedia).

For example, 2 + 3 is an arithmetic and programming expression which evaluates to 5. A variable is an expression because it is a pointer to a value in memory, so y + 6 is an expression. An example of a relational expression is 4 = 4 which evaluates to True (source Wikipedia).

3.1.1 Mathematical expressions

Operator	Example	Precedence level	Operation
+	`x + y`	3	Addition
-	`x - y`	3	Subtraction
*	`x * y`	2	Multiplication
/	`x / y`	2	Division
`Mod`	`x Mod y`	2	Modulo
`Power`	`Power(x,y)` x^y	1	Power of

Precedence level: In an expression, operations with level 1 are evaluated before operations with level 2, which are evaluated before operations with level 3.

Examples:

`4 + 5 * 3 + 2 = 21` > `4 + 15 + 2`

`(4 + 5) * (3 + 2) = 45` > `9 * 5`

$(4 + 5)^2 * (3 + 2) = 405$ > $9^2 * 5$ > `81 * 5`
`Power(4+5,2)*(3+2)`

`11 Mod 4 = 3` > Mod is the remainder of 10 / 4

23^3 `Power(23,3)` > 23 at the power of 3

$- 2^2 = - 4$
$(-2)^2 = 4$

3.1.2 Relational expressions

In computer science in relational expressions an operator tests some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 >= 3).
In B4x these operators return **True** or **False**, depending on whether the conditional relationship between the two operands holds or not.

Operator	Example	Used to test
=	x = y	the equivalence of two values
<>	x <> y	the negated equivalence of two values
>	x > y	if the value of the left expression is greater than that of the right
<	x < y	if the value of the left expression is less than that of the right
>=	x >= y	if the value of the left expression is greater than or equal to that of the right
<=	x <= y	if the value of the left expression is less than or equal to that of the right

3.1.3 Boolean expressions

In computer science, a Boolean expression is an expression that produces a Boolean value when evaluated, i.e. one of **True** or **False**. A Boolean expression may be composed of a combination of the Boolean constants **True** or **False**, Boolean-typed variables, Boolean-valued operators, and Boolean-valued functions (source Wikipedia).

Boolean operators are used in conditional statements such as IF-Then and Select-Case.

Operator	Comment
Or	Boolean Or Z = X Or Y Z = True if X or Y is equal to True or both are True
And	Boolean And Z = X And Y Z = True if X and Y are both equal to True
Not ( )	Boolean Not X = True Y = Not(X) > Y = False

		Or	And
X	Y	Z	Z
False	False	False	False
True	False	True	False
False	True	True	False
True	True	True	True

3.2 Standard keywords

Not all keywords are available in B4R.

- **Abs** (Number As Double) As Double
- **ACos** (Value As Double) As Double
- **ACosD** (Value As Double) As Double
- **Array**
- **Asc** (Char As Char) As Int
- **ASin** (Value As Double) As Double
- **ASinD** (Value As Double) As Double
- **ATan** (Value As Double) As Double
- **ATan2** (Y As Double, X As Double) As Double
- **ATan2D** (Y As Double, X As Double) As Double
- **ATanD** (Value As Double) As Double
- **BytesToString** (Data() As Byte, StartOffset As Int, Length As Int, CharSet As String) As String
- **CallSub** (Component As Object, Sub As String) As Object
- **CallSub2** (Component As Object, Sub As String, Argument As Object) As Object
- **CallSub3** (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object) As Object
- **Catch**
- **cE** As Double
- **Ceil** (Number As Double) As Double
- **CharsToString** (Chars() As Char, StartOffset As Int, Length As Int) As String
- **Chr** (UnicodeValue As Int) As Char
- **Continue**
- **Cos** (Radians As Double) As Double
- **CosD** (Degrees As Double) As Double
- **cPI** As Double
- **CRLF** As String
- **Dim**
- **Exit**
- **False** As Boolean
- **Floor** (Number As Double) As Double
- **For**
- **GetType** (object As Object) As String
- **If**
- **Is**
- **IsNumber** (Text As String) As Boolean
- **LoadBitmap** (Dir As String, FileName As String) As Bitmap
- **LoadBitmapSample** (Dir As String, FileName As String, MaxWidth As Int, MaxHeight As Int) As Bitmap
- **Log** (Message As String)
- **Logarithm** (Number As Double, Base As Double) As Double
- **LogColor** (Message As String, Color As Int)
- **Max** (Number1 As Double, Number2 As Double) As Double
- **Me** As Object
- **Min** (Number1 As Double, Number2 As Double) As Double
- **Not** (Value As Boolean) As Boolean

- **Null** As Object
- **NumberFormat** (Number As Double, MinimumIntegers As Int, MaximumFractions As Int) As String
- **NumberFormat2** (Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean) As String
- **Power** (Base As Double, Exponent As Double) As Double
- **QUOTE** As String
- **Regex** As Regex
- **Return**
- **Rnd** (Min As Int, Max As Int) As Int
- **RndSeed** (Seed As Long)
- **Round** (Number As Double) As Long
- **Round2** (Number As Double, DecimalPlaces As Int) As Double
- **Select**
- **Sender** As Object
- **Sin** (Radians As Double) As Double
- **SinD** (Degrees As Double) As Double
- **Sleep** (Milliseconds As Int)
- **SmartStringFormatter** (Format As String, Value As Object) As String
- **Sqrt** (Value As Double) As Double
- **Sub**
- **SubExists** (Object As Object, Sub As String) As Boolean
- **TAB** As String
- **Tan** (Radians As Double) As Double
- **TanD** (Degrees As Double) As Double
- **True** As Boolean
- **Try**
- **Type**
- **Until**
- **While**

Abs (Number As Double) As Double

Returns the absolute value.

ACos (Value As Double) As Double

Returns the angle measured with radians.

ACosD (Value As Double) As Double

Returns the angle measured with degrees.

Array

Creates a single dimension array of the specified type.
The syntax is: Array [As type] (list of values).
If the type is ommitted then an array of objects will be created.
Example:

```
Dim Days() As String
Days = Array As String("Sunday", "Monday", ...)
```

Asc (Char As Char) As Int

Returns the unicode code point of the given character or first character in string.

ASin (Value As Double) As Double

Returns the angle measured with radians.

ASinD (Value As Double) As Double

Returns the angle measured with degrees.

ATan (Value As Double) As Double

Returns the angle measured with radians.

ATan2 (Y As Double, X As Double) As Double

Returns the angle measured with radians.

ATan2D (Y As Double, X As Double) As Double

Returns the angle measured with degrees.

ATanD (Value As Double) As Double

Returns the angle measured with degrees.

BytesToString (Data() As Byte, StartOffset As Int, Length As Int, CharSet As String) As String

Decodes the given bytes array as a string.
Data - The bytes array.
StartOffset - The first byte to read.
Length - Number of bytes to read.
CharSet - The name of the character set.
Example:
```
Dim s As String
s = BytesToString(Buffer, 0, Buffer.Length, "UTF-8")
```

CallSub (Component As Object, Sub As String) As Object

Calls the given sub. CallSub can be used to call a sub which belongs to a different module.
However the sub will only be called if the other module is not paused. In that case an empty string will be returned.
You can use IsPaused to test whether a module is paused.
This means that one activity cannot call a sub of a different activity. As the other activity will be paused for sure.
CallSub allows an activity to call a service sub or a service to call an activity sub.
Note that it is not possible to call subs of code modules.
CallSub can also be used to call subs in the current module. Pass Me as the component in that case.
Example:
```
CallSub(Main, "RefreshData")
```

CallSub2 (Component As Object, Sub As String, Argument As Object) As Object

Similar to CallSub. Calls a sub with a single argument.

CallSub3 (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object) As Object

Catch

Any exception thrown inside a try block will be caught in the catch block.
Call LastException to get the caught exception.
Syntax:
```
Try
  ...
Catch
  ...
End Try
```

cE As Double

e (natural logarithm base) constant.

Ceil (Number As Double) As Double

Returns the smallest double that is greater or equal to the specified number and is equal to an integer.

CharsToString (Chars() As Char, StartOffset As Int, Length As Int) As String

Creates a new String by copying the characters from the array.
Copying starts from StartOffset and the number of characters copied equals to Length.

Chr (UnicodeValue As Int) As Char

Returns the character that is represented by the given unicode value.

Continue

Stops executing the current iteration and continues with the next one.

Cos (Radians As Double) As Double

Calculates the trigonometric cosine function. Angle measured in radians.

CosD (Degrees As Double) As Double

Calculates the trigonometric cosine function. Angle measured in degrees.

cPI As Double

PI constant.

CRLF As String

New line character. The value of Chr(10).

Dim

Declares a variable.
Syntax:
Declare a single variable:
Dim variable name [As type] [= expression]
The default type is String.

Declare multiple variables. All variables will be of the specified type.
Dim [Const] variable1 [= expression], variable2 [= expression], ..., [As type]
Note that the shorthand syntax only applies to Dim keyword.
Example: `Dim a = 1, b = 2, c = 3 As Int`

Declare an array:
Dim variable(Rank1, Rank2, ...) [As type]
Example: `Dim Days(7) As String`
The actual rank can be omitted for zero length arrays.

Exit

Exits the most inner loop.
Note that Exit inside a Select block will exit the Select block.

False As Boolean

Floor (Number As Double) As Double

Returns the largest double that is smaller or equal to the specified number and is equal to an integer.

For

Syntax:
```
For variable = value1 To value2 [Step interval]
  ...
Next
```
If the iterator variable was not declared before it will be of type Int.

Or:
```
For Each variable As type In collection
  ...
Next
```
Examples:
```
For i = 1 To 10
  Log(i) 'Will print 1 to 10 (inclusive).
Next
For Each n As Int In Numbers 'an array
  Sum = Sum + n
Next
```

Note that the loop limits will only be calculated once before the first iteration.

GetType (object As Object) As String

Returns a string representing the object's java type.

If

Single line:
```
If condition Then true-statement [Else false-statement]
```
Multiline:
```
If condition Then
  statement
Else If condition Then
  statement
  ...
Else
  statement
End If
```

IsNumber (Text As String) As Boolean

Tests whether the specified string can be safely parsed as a number.

LoadBitmap (Dir As String, FileName As String) As Bitmap

Loads the bitmap.
Note that the Android file system is case sensitive.
You should consider using LoadBitmapSample if the image size is large.
The actual file size is not relevant as images are usually stored compressed.
Example:
Activity.SetBackgroundImage(LoadBitmap(File.DirAssets, "SomeFile.jpg"))

LoadBitmapSample (Dir As String, FileName As String, MaxWidth As Int, MaxHeight As Int) As Bitmap

Loads the bitmap.
The decoder will subsample the bitmap if MaxWidth or MaxHeight are smaller than the bitmap dimensions.
This can save a lot of memory when loading large images.
Example:

```
Panel1.SetBackgroundImage(LoadBitmapSample(File.DirAssets, "SomeFile.jpg",
Panel1.Width, Panel1.Height))
```

Log (Message As String)

Logs a message. The log can be viewed in the Logs tab.

Logarithm (Number As Double, Base As Double) As Double

LogColor (Message As String, Color As Int)

Logs a message. The message will be displayed in the IDE with the specified color.

Max (Number1 As Double, Number2 As Double) As Double

Returns the larger number between the two numbers.

Me As Object

For classes: returns a reference to the current instance.
For activities and services: returns a reference to an object that can be used with CallSub, CallSubDelayed and SubExists keywords.
Cannot be used in code modules.

Min (Number1 As Double, Number2 As Double) As Double

Returns the smaller number between the two numbers.

Not (Value As Boolean) As Boolean

Inverts the value of the given boolean.

Null As Object

NumberFormat (Number As Double, MinimumIntegers As Int, MaximumFractions As Int) As String

Converts the specified number to a string.
The string will include at least Minimum Integers and at most Maximum Fractions digits.
Example:
```
Log(NumberFormat(12345.6789, 0, 2)) '"12,345.68"
Log(NumberFormat(1, 3 ,0)) '"001"
```

NumberFormat2 (Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean) As String

Converts the specified number to a string.
The string will include at least Minimum Integers, at most Maximum Fractions digits and at least Minimum Fractions digits.
GroupingUsed - Determines whether to group every three integers.
Example:
```
Log(NumberFormat2(12345.67, 0, 3, 3, false)) '"12345.670"
```

Power (Base As Double, Exponent As Double) As Double

Returns the Base value raised to the Exponent power.

QUOTE As String

Quote character. The value of Chr(34).

Regex As Regex

Regular expressions related methods.

Return

Returns from the current sub and optionally returns the given value.
Syntax: `Return [value]`

Rnd (Min As Int, Max As Int) As Int

Returns a random integer between Min (inclusive) and Max (exclusive).

RndSeed (Seed As Long)

Sets the random seed value.
This method can be used for debugging as it allows you to get the same results each time.

Round (Number As Double) As Long

Returns the closest long number to the given number.

Round2 (Number As Double, DecimalPlaces As Int) As Double

Rounds the given number and leaves up to the specified number of fractional digits.

Select

Compares a single value to multiple values.
Example:
```
Dim value As Int
value = 7
Select value
  Case 1
    Log("One")
  Case 2, 4, 6, 8
    Log("Even")
  Case 3, 5, 7, 9
    Log("Odd larger than one")
  Case Else
    Log("Larger than 9")
End Select
```

Sender As Object

Returns the object that raised the event.
Only valid while inside the event sub.
Example:
```
Sub Button_Click
  Dim b As Button
  b = Sender
  b.Text = "I've been clicked"
End Sub
```

Sin (Radians As Double) As Double

Calculates the trigonometric sine function. Angle measured in radians.

SinD (Degrees As Double) As Double

Calculates the trigonometric sine function. Angle measured in degrees.

Sleep (Value As Double) As Double

Pauses the current sub execution and resumes it after the specified time.

SmartStringFormatter (Format As String, Value As Object) As String
Internal keyword used by the Smart String literal.

Sqrt (Value As Double) As Double

Returns the positive square root.

Sub

Declares a sub with the parameters and return type.
Syntax: Sub name [(list of parameters)] [As return-type]
Parameters include name and type.
The lengths of arrays dimensions should not be included.
Example:

```
Sub MySub (FirstName As String, LastName As String, Age As Int, OtherValues() As
Double) As Boolean
  ...
End Sub
```

In this example OtherValues is a single dimension array.
The return type declaration is different than other declarations as the array parenthesis follow the type and not
the name (which does not exist in this case).

SubExists (Object As Object, Sub As String) As Boolean

Tests whether the object includes the specified method.
Returns false if the object was not initialized or not an instance of a user class.

TAB As String

Tab character.

Tan (Radians As Double) As Double

Calculates the trigonometric tangent function. Angle measured in radians.

TanD (Degrees As Double) As Double

Calculates the trigonometric tangent function. Angle measured in degrees.

True As Boolean

Try

Any exception thrown inside a try block will be caught in the catch block.
Call LastException to get the caught exception.
Syntax:

```
Try
...
Catch
...
End Try
```

Type

Declares a structure.
Can only be used inside sub Globals or sub Process_Globals.
Syntax:
Type type-name (field1, field2, ...)
Fields include name and type.
Example:

```
Type MyType (Name As String, Items(10) As Int)
Dim a, b As MyType
a.Initialize
a.Items(2) = 123
```

Until

Loops until the condition is true.
Syntax:

```
Do Until condition
  ...
Loop
```

While

Loops while the condition is true.
Syntax:
Do While condition
...
Loop

3.3 Conditional statements

Different conditional statements are available in Basic.

3.3.1 If – Then – End If

The **If-Then-Else** structure allows to operate conditional tests and execute different code sections according to the test result.
General case:

```
If test1 Then
   ' code1
Else If test2 Then
   ' code2
Else
   ' code3
End If
```

The **If-Then-Else** structure works as follows:

1. When reaching the line with the **`If`** keyword, **`test1`** is executed.
2. If the test result is **True**, then **`code1`** is executed until the line with the **`Else If`** keyword. And jumps to the line following the **`End If`** keyword and continues.
3. If the result is **False**, then **`test2`** is executed.
4. If the test result is **True**, then **`code2`** is executed until the line with the **`Else`** keyword. And jumps to the line following the **`End If`** keyword and continues.
5. If the result is **False**, then **`code3`** is executed and continues at the line following the **`End If`** keyword.

The tests can be any kind of conditional test with two possibilities **True** or **False**.
Some examples:

```
If b = 0 Then
   a = 0
End If
```

The simplest **If-Then** structure.

```
If b = 0 Then a = 0
```

The same but in one line.

```
If b = 0 Then
   a = 0
Else
   a = 1
End If
```

The simplest **If-Then-Else** structure.

```
If b = 0 Then a = 0 Else a = 1
```

The same but in one line.

Personally, I prefer the structure on several lines, better readable.
An old habit from HP Basic some decades ago, this Basic accepted only one instruction per line.

Note. Difference between:

B4x	VB
Else If	**ElseIf**

In B4x there is a blank character between **Else** and **If**.

Some users try to use this notation:

```
If b = 0 Then a = 0 : c = 1
```

There is a big difference between B4x and VB that gives errors :
The above statements is equivalent to :

B4x
```
If b = 0 Then
   a = 0
End If
c = 1
```

VB
```
If b = 0 Then
   a = 0
   c = 1
End If
```

The colon character ' : ' in the line above is treated in B4x like a CarriageReturn CR character.

This structure throws an error.

```
Sub Plus1 : x = x + 1 : End Sub
```

You cannot have a Sub declaration and End Sub on the same line.

3.3.2 Select – Case

The **Select - Case** structure allows to compare a **TestExpression** with other **Expressions** and to execute different code sections according to the matches between the **TestExpression** and **Expressions**.

General case:

```
Select TestExpression
Case ExpressionList1
   ' code1
Case ExpressionList2
   ' code2
Case Else
   ' code3
End Select
```

TestExpression is the expression to test.

ExpressionList1 is a list of expressions to compare to **TestExpression**

ExpressionList2 is another list of expressions to compare to **TestExpression**

The **Select - Case** structure works as follows:

1. The **TestExpression** is evaluated.
2. If one element in the **ExpressionList1** matches **TestExpression** then executes **code1** and continues at the line following the **End Select** keyword.
3. If one element in the **ExpressionList2** matches **TestExpression** then executes **code2** and continues at the line following the **End Select** keyword.
4. For no expression matches **TestExpression** executes **code3** and continues at the line following the **End Select** keyword.

TestExpression can be any expression or value.
ExpressionList1 is a list of any expressions or values.

Examples:

```
Select Value
Case 1, 2, 3, 4
```

The Value variable is a numeric value.

```
Select a + b
Case 12, 24
```

The **TestExpression** is the sum of a + b

```
Select Txt.CharAt
Case "A", "B", "C"
```

The **TestExpression** is a character at

```
Sub Activity_Touch (Action As Int, X As Float, Y As Float)
   Select Action
   Case Activity.ACTION_DOWN

   Case Activity.ACTION_MOVE

   Case Activity.ACTION_UP

   End Select
End Sub
```

Note. Differences between:

B4x
```
Select Value
Case 1,2,3,4,8,9,10
```

VB
```
Select Case Value
Case 1 To 4 , 8 To 9
```

In VB the keyword `Case` is added after the `Select` keyword.
VB accepts `Case 1 To 4` , this is not implemented in B4x.

3.4 Loop structures

Different loop structures are available in Basic.

3.4.1 For – Next

In a **For–Next** loop a same code will be executed a certain number of times.
Example:

```
For i = n1 To n2 Step n3

   ' Specific code

Next
```

`i` incremental variable
`n1` initial value
`n2` final value
`n3` step

The **For–Next** loop works as below:

1. At the beginning, the incremental variable **i** is equal to the initial value **n1**.
   i = n1
2. The specific code between the **For** and **Next** keywords is executed.
3. When reaching **Next**, the incremental variable **i** is incremented by the step value **n3**.
   i = i + n3.
4. The program jumps back to **For**, compares if the incremental variable **i** is lower or equal to the final value **n2**.
   test if i <= n2
5. If **Yes**, the program continues at step 2, the line following the **For** keyword.
6. If **No**, the program continues at the line following the **Next** keyword.

If the step value is equal to '+1' the step keyword is not needed.

```
For i = 0 To 10
Next
```

is the same as

```
For i = 0 To 10 Step 1
Next
```

The step variable can be negative.

```
For i = n3 To 0 Step -1

Next
```

It is possible to exit a For – Next loop with the `Exit` keyword.

```
For i = 0 To 10
   ' code
   If A = 0 Then Exit
   ' code
Next
```

In this example, if the variable a equals 0
Then exit the loop.

Note : Differences between

B4x	VB
`Next`	`Next i`
`Exit`	`Exit For`

In VB :

- The increment variable is added after the **`Next`** Keyword.
- The loop type is specified after the **`Exit`** keyword.

3.4.2 For - Each

It is a variant of the For - Next loop.

Example:

```
For Each n As Type In Array
   ' Specific code
Next
```

n	variable any type or object
Type	type of variable n
Array	Array of values or objects

The **For–Each** loop works as below:

1. At the beginning, **n** gets the value of the first element in the Array.
   n = Array(0)
2. The specific code between the **`For`** and **`Next`** keywords is executed.
3. When reaching **Next**, the program checks if **n** is the last element in the array.
4. If **No**, the variable **n** gets the next value in the Array and continues at step 2, the line following the **`For`** keyword.
   n = Array(next)
5. If **Yes**, the program continues at the line following the **`Next`** keyword.

Example For - Each :

```
Private Numbers() As Int
Private Sum As Int

Numbers = Array As Int(1, 3, 5 , 2, 9)

Sum = 0
For Each n As Int In Numbers
   Sum = Sum + n
Next
```

Same example but with a For - Next loop :

```
Private Numbers() As Int
Private Sum As Int
Private i As Int

Numbers = Array As Int(1, 3, 5 , 2, 9)

Sum = 0
For i = 0 To Numbers.Length - 1
   Sum = Sum + Numbers(i)
Next
```

This example shows the power of the For - Each loop :

```
For Each lbl As Label In Activity
   lbl.TextSize = 20
Next
```

Same example with a For - Next loop :

```
For i = 0 To Activity.NumberOfViews - 1
   Private v As View
   v = Activity.GetView(i)
   If v Is Label Then
      Private lbl As Label
      lbl = v
      lbl.TextSize = 20
   End If
Next
```

3.4.3 Do - Loop

Several configurations exist:

```
Do While test
   ' code
Loop
```

test is any expression
Executes the **code** while **test** is **True**

```
Do Until test
   ' code
Loop
```

test is any expression
Executes the **code** until **test** is **True**

The **Do While -Loop** loop works as below :

1. At the beginning, **test** is evaluated.
2. If **True**, then executes **code**
3. If **False** continues at the line following the **Loop** keyword.

The **Do Until -Loop** loop works as below :

1. At the beginning, **test** is evaluated.
2. If **False**, then executes **code**
3. If **True** continues at the line following the **Loop** keyword.

It is possible to exit a Do-Loop structure with the Exit keyword.

```
Do While test
   ' code
   If a = 0 Then Exit
   ' code
Loop
```

If **a = 0** then exit the loop

Examples :

Do Until Loop :
```
Private i, n As Int

i = 0
Do Until i = 10
   ' code
   i = i + 1
Loop
```

Do While Loop :
```
Private i, n As Int

i = 0
Do While i < 10
   ' code
   i = i + 1
Loop
```

Read a text file and fill a List :
```
Private lstText As List
Private line As String
Private tr As TextReader

tr.Initialize(File.OpenInput(File.DirInternal, "test.txt"))
lstText.Initialize
line = tr.ReadLine
Do While line <> Null
   lstText.Add(line)
   line = tr.ReadLine
Loop

tr.Close
```

Note : Difference between:

B4x	VB
`Exit`	`Exit Loop`

In VB the loop type is specified after the **`Exit`** keyword.

VB accepts also the following loops, which are not supported in B4x.
```
Do                          Do
  ' code                      ' code
Loop While test             Loop Until test
```

3.5 Subs

A Subroutine ("Sub") is a piece of code. It can be any length, and it has a distinctive name and a defined scope (in the means of variables scope discussed earlier). In B4x code, a subroutine is called "Sub", and is equivalent to procedures, functions, methods and subs in other programming languages. The lines of code inside a Sub are executed from first to last, as described in the program flow chapter.
It is not recommended to have Subs with a large amount of code, they get less readable.

3.5.1 Declaring

A Sub is declared in the following way:

```
Sub CalcInterest(Capital As Double, Rate As Double) As Double
   Return Capital * Rate / 100
End Sub
```

It starts with the keyword **Sub**, followed by the Sub's name, followed by a parameter list, followed by the return type and ends with the keywords **End Sub**.
Subs are always declared at the top level of the module, you cannot nest two Subs one inside the other.

3.5.2 Calling a Sub

When you want to execute the lines of code in a Sub, you simply write the Sub's name.

For example:

```
   Interest = CalcInterest(1234, 5.2)
```

Interest	Value returned by the Sub.
`CalcInterest`	Sub name.
`1235`	Capital value transmitted to the Sub.
`5.25`	Rate value transmitted to the Sub.

3.5.3 Calling a Sub from another module

A subroutine declared in a code module can be accessed from any other module but the name of the routine must have the name of the module where it was declared as a prefix.

Example: If the `CalcInterest` routine is declared in module `MyModule` then calling the routine must be :

```
   Interest = MyModule.CalcInterest(1234, 5.2)
```

instead of:

```
   Interest = CalcInterest(1234, 5.2)
```

3.5.4 Naming

Basically, you can name a Sub any name that's legal for a variable. It is recommended to name the Sub with a significant name, like **CalcInterest** in the example, so you can tell what it does from reading the code.
There is no limit on the number of Subs you can add to your program, but it is not allowed to have two Subs with the same name in the same module.

```
Sub CalcInterest(Capital As Double, Rate As Double) As Double
   Return Capital * Rate / 100
End Sub
```

3.5.5 Parameters

Parameters can be transmitted to the Sub. The list follows the sub name. The parameter list is put in brackets.
The parameter types should be declared directly in the list.

```
Sub CalcInterest(Capital As Double, Rate As Double) As Double
   Return Capital * Rate / 100
End Sub
```

In B4x, the parameters are transmitted by value and not by reference.

3.5.6 Returned value

A sub can return a value, this can be any object.
Returning a value is done with the Return keyword.
The type of the return value is added after the parameter list.

```
Sub CalcInterest(Capital As Double, Rate As Double) As Double
   Return Capital * Rate / 100
End Sub
```

3.6 Resumable Subs

Resumable subs is a new feature added in B4A v7.00 / B4i v4.00 / B4J v5.50. It dramatically simplifies the handling of asynchronous tasks.
(This feature is a variant of stackless coroutines.)

You find more examples in the forum.

The special feature of resumable subs is that they can be paused, without pausing the executing thread, and later be resumed.
The program doesn't wait for the resumable sub to be continued. Other events will be raised as usual.

Any sub with one or more calls to Sleep or Wait For is a resumable sub.
The IDE shows this indicator  next to the sub declaration:

```
Private Sub CountDown(Start As Int)
   For i = Start To 0 Step -1
      Label1.Text = i
      Sleep(1000)
   Next
End Sub
```

3.6.1 Sleep

Pauses the current sub execution and resumes it after the specified time.

Sleep (Milliseconds As Int) Milliseconds, time delay in milliseconds.
Example:
Sleep(1000)

Using Sleep is simple:

```
Log(1)
Sleep(1000)
Log(2)
```

The sub will be paused for 1000 milliseconds and then be resumed.

You can call Sleep(0) for the shortest pause. This can be used to allow the UI to be refreshed. It is a good alternative to DoEvents (which doesn't exist in B4J and B4i and should be avoided in B4A).

```
Sub VeryBusySub
   For i = 1 To 10000000
      'do something
      If i Mod 1000 = 0 Then Sleep(0) 'allow the UI to refresh every 1000 iterations.
   Next
   Log("finished!")
End Sub
```

3.6.2 Wait For

B4X programming languages are event driven. Asynchronous tasks run in the background and raise an event when the task completes.
With the new Wait For keyword you can handle the event inside the current sub.

For example, this code will wait for the GoogleMap Ready event (B4J example):

```
Sub AppStart (Form1 As Form, Args() As String)
   MainForm = Form1
   MainForm.RootPane.LoadLayout("1") 'Load the layout file.

   gmap.Initialize("gmap")
   Pane1.AddNode(gmap.AsPane, 0, 0, Pane1.Width, Pane1.Height)
   MainForm.Show
   Wait For gmap_Ready '<----------------
   gmap.AddMarker(10, 10, "Marker")
End Sub
```

A bit more complicated example with FTP:
Listing all files in a remote folder and then downloading all the files:

```
Sub DownloadFolder (ServerFolder As String)
  FTP.List(ServerFolder)
  Wait For FTP_ListCompleted (ServerPath As String, Success As Boolean, Folders() As
    FTPEntry, Files() As FTPEntry) '<----
  If Success Then
    For Each f As FTPEntry In Files
      FTP.DownloadFile(ServerPath & f.Name, False, File.DirApp, f.Name)
      Wait For FTP_DownloadCompleted (ServerPath2 As String, Success As Boolean) '<----
      Log($"File ${ServerPath2} downloaded. Success = ${Success}"$)
     Next
  End If
  Log("Finish")
End Sub
```

When the Wait For keyword is called, the sub is paused and the internal events dispatcher takes care to resume it when the event is raised. If the event is never raised then the sub will never be resumed. The program will still be completely responsive.
If Wait For is later called with the same event then the new sub instance will replace the previous one.

Lets say that we want to create a sub that downloads an image and sets it to an ImageView:

```
'Bad example. Don't use.
Sub DownloadImage(Link As String, iv As ImageView)
   Dim job As HttpJob
   job.Initialize("", Me) 'note that the name parameter is no longer needed.
   job.Download(Link)
   Wait For JobDone(job As HttpJob)
   If job.Success Then
      iv.SetImage (job.GetBitmap) 'replace with iv.Bitmap = job.GetBitmap in B4A / B4i
   End If
   job.Release
End Sub
```

It will work properly if we call it once (more correctly, if we don't call it again before the previous call completes).
If we call it like this:

```
   DownloadImage("https://www.b4x.com/images3/android.png", ImageView1)
   DownloadImage("https://www.b4x.com/images3/apple.png", ImageView2)
```

Then only the second image will show because the second call to Wait For JobDone will overwrite the previous one.
This brings us to the second variant of Wait For.
To solve this issue, Wait For can distinguish between events based on the event sender.
This is done with an optional parameter:

Wait For (<sender>) <event signature>

Example:

```
'Good example. Use.
Sub DownloadImage(Link As String, iv As ImageView)
   Dim job As HttpJob
   job.Initialize("", Me) 'note that the name parameter is no longer needed.
   job.Download(Link)
   Wait For (job) JobDone(job As HttpJob)
   If job.Success Then
     iv.SetImage (job.GetBitmap) 'replace with iv.Bitmap = job.GetBitmap in B4A / B4i
   End If
   job.Release
End Sub
```

With the above code, each resumable sub instance will wait for a different event and will not be affected by other calls.

3.6.3 Code Flow

```
Sub S1
   Log("S1: A")
   S2
   Log("S1: B")
End Sub

Sub S2
   Log("S2: A")
   Sleep(0)
   Log("S2: B")
End Sub
```

The output is:
S1: A
S2: A
S1: B
S2: B

Whenever Sleep or Wait For are called, the current sub is paused. This is equivalent to calling Return.

3.6.4 Waiting for a resumable sub to complete

When one sub calls a second resumable sub, the code in the first sub will continue after the first Sleep or Wait For call (in the second sub).

If you want to wait for the second sub to complete then you can raise an event from the second sub and wait for it in the first:

```
Sub FirstSub
   Log("FirstSub started")
   SecondSub
   Wait For SecondSub_Complete
   Log("FirstSub completed")
End Sub

Sub SecondSub
   Log("SecondSub started")
   Sleep(1000)
   Log("SecondSub completed")
   CallSubDelayed(Me, "SecondSub_Complete")
End Sub
```

Logs:
FirstSub started
SecondSub started
SecondSub completed
FirstSub completed

Notes:
- It is safer to use CallSubDelayed than CallSub. CallSub will fail if the second sub is never paused (for example if the sleep is only called based on some condition).
- There is an assumption here that FirstSub will not be called again until it is completed.

3.6.5 DoEvents

Starting from B4A v7.0 the following warning will appear for DoEvents calls:
DoEvents is deprecated. It can lead to stability issues. Use Sleep(0) instead (if really needed).

The purpose of DoEvents was to allow the UI to be updated while the main thread is busy.
DoEvents which shares the same implementation as the modal dialogs implementation, is a low level implementation. It accesses the process message queue and runs some of the waiting messages.

As Android evolved, the handling of the message queue became more sophisticated and fragile.
The reasons for deprecating DoEvents are:

1. It is a major source for instability issues. It can lead to hard to debug crashes or ANR (application not responding) dialogs. Note that this is also true for the modal dialogs (such as Msgbox and InputList).
2. There are better ways to keep the main thread free. For example use the asynchronous SQL methods instead of the synchronous methods.
3. It doesn't do what many developers expect it to do. As it only handles UI related messages, most events could not be raised from a DoEvents call.
4. It is now possible to call Sleep to pause the current sub and resume it after the waiting messages are processed. Sleep implementation is completely different than DoEvents. It doesn't hold the thread. It instead releases it while preserving the sub state.
Unlike DoEvents which only processed UI related messages, with Sleep all messages will be processed and other events will be raised.
(Note that using Wait For to wait for an event is better than calling Sleep in a loop.)

With that said, DoEvents is still there and existing applications will work exactly as before.

3.6.6 Dialogs

Modal dialogs = dialogs that hold the main thread until the dialog is dismissed.

As written above, modal dialogs share the same implementation as DoEvents. It is therefore recommended to switch to the new async dialogs instead. Using Wait For it is really a simple change:

Instead of:

```
Dim res As Int = Msgbox2("Delete?", "Title", "Yes", "Cancel", "No", Null)
If res = DialogResponse.POSITIVE Then
   '...
End If
```

You should use :

```
Msgbox2Async("Delete?", "Title", "Yes", "Cancel", "No", Null, False)
Wait For Msgbox_Result (Result As Int)
If Result = DialogResponse.POSITIVE Then
      '...
End If
```

Wait For doesn't hold the main thread. It instead saves the current sub state and releases it. The code will resume when the user clicks on one of the dialog buttons.
The other similar new methods are: MsgboxAsync, InputListAsync and InputMapAsync.

With the exception of MsgboxAsync, the new methods also add a new *cancelable* parameter. If it is true then the dialog can be dismissed by clicking on the back key or outside the dialog. This is the default behavior of the older methods.

As other code can run while the async dialog is visible, it is possible that multiple dialogs will appear at the same time.
If this case is relevant for your app then you should set the sender filter parameter in the Wait For call:

```
Dim sf As Object = Msgbox2Async("Delete?", "Title", "Yes", "Cancel", "No", Null, False)
Wait For (sf) Msgbox_Result (Result As Int)
If Result = DialogResponse.POSITIVE Then
   '...
End If
```

This allows multiple messages to be displayed and the result events will be handled correctly.

3.6.7 SQL with Wait For

The new resumable subs feature, makes it simpler to work with large data sets with minimum effect on the program responsiveness.

The new standard way to insert data is:

```
For i = 1 To 1000
   SQL1.AddNonQueryToBatch("INSERT INTO table1 VALUES (?)", Array(Rnd(0, 100000)))
Next
Dim SenderFilter As Object = SQL1.ExecNonQueryBatch("SQL")
Wait For (SenderFilter) SQL_NonQueryComplete (Success As Boolean)
Log("NonQuery: " & Success)
```

The steps are:
- Call AddNonQueryToBatch for each commands that should be issued.
- Execute the commands with ExecNonQueryBatch. This is an asynchronous method. The commands will be executed in the background and the NonQueryComplete event will be raised when done.
- This call returns an object that can be used as the sender filter parameter. This is important as there could be multiple background batch executions running. With the filter parameter the event will be caught by the correct Wait For call in all cases.
- Note that SQL1.ExecNonQueryBatch begins and ends a transaction internally.

3.6.7.1 Queries

In most cases the queries will be fast and should therefore be issued synchronously with SQL1.ExecQuery2. However if there is a slow query then you should switch to SQL1.ExecQueryAsync:

```
Dim SenderFilter As Object = SQL1.ExecQueryAsync("SQL", "SELECT * FROM table1", Null)
Wait For (SenderFilter) SQL_QueryComplete (Success As Boolean, rs As ResultSet)
If Success Then
   Do While rs.NextRow
      Log(rs.GetInt2(0))
   Loop
   rs.Close
Else
   Log(LastException)
End If
```

As in the previous case, the ExecQueryAsync method returns an object that is used as the sender filter parameter.

Tips:
1. ResultSet type in B4A extends the Cursor type. You can change it to Cursor if you prefer. The advantage of using ResultSet is that it is compatible with B4J and B4i.
2. If the number of rows returned from the query is large then the Do While loop will be slow in debug mode. You can make it faster by putting it in a different sub and cleaning the project (Ctrl + P):

```
   Wait For (SenderFilter) SQL_QueryComplete (Success As Boolean, rs As ResultSet)
   If Success Then
      WorkWithResultSet(rs)
   Else
      Log(LastException)
   End If
End Sub

Private Sub WorkWithResultSet(rs As ResultSet)
   Do While rs.NextRow
      Log(rs.GetInt2(0))
   Loop
   rs.Close
End Sub
```

This is related to a debugger optimization that is currently disabled in resumable subs.
The performance of both solutions will be the same in release mode.

3.6.7.2 B4J

- Requires jSQL v1.50+ (https://www.b4x.com/android/forum/threads/updates-to-internal-libaries.48274/#post-503552).
- Recommended to set the journal mode to WAL: https://www.b4x.com/android/forum/t...ent-access-to-sqlite-databases.39904/#content

3.6.8 Notes & Tips

- Resumable subs cannot return a value.
- The performance overhead of resumable subs in release mode should be insignificant in most cases. The overhead can be larger in debug mode. (If this becomes an issue then take the slow parts of the code and move them to other subs that are called from the resumable sub.)
- Wait For events handlers precede the regular event handlers.
- Resumable subs do not create additional threads. The code is executed by the main thread, or the handler thread in server solutions.

3.7 Events

In Object-oriented programming we have objects which can react on different user actions called events.
The number and the type of events an object can raise depend on the type of the object.

3.7.1 B4A

User interface objects are called 'Views' in Android.

Summary of the events for different views:

	Events																
Views	Click	LongClick	Touch	Down	Up	KeyPress	KeyUp	ItemClick	ItemLongClick	CheckedChange	EnterPressed	FocusChanged	TextChanged	ScrollChanged	ValueChanged	TabChanged	OverrideUrl	PageFinished
Activity	■	■	■			■	■											
Button	■	■		■	■													
CheckBox										■								
EditText											■	■	■					
HorizontalScrollView														■				
ImageView	■	■																
Label	■	■																
ListView								■	■									
Panel	■	■	■															
RadioButton										■								
ScrollView														■				
SeekBar															■			
Spinner								■										
TabHost	■	■														■		
ToggleButton										■								
WebView																	■	■

The most common events are:

- **Click** Event raised when the user clicks on the view.
  Example:
```
Sub Button1_Click
  ' Your code
End Sub
```

- **LongClick** Event raised when the user clicks on the view and holds it pressed for a while.
  Example:
```
Sub Button1_LongClick
  ' Your code
End Sub
```

- **Touch** `(Action As Int, X As Float, Y As Float)`
  Event raised when the user touches the screen.

  Three different actions are handled:
  - Activity.Action_DOWN, the user touches the screen.
  - Activity.Action_MOVE, the user moves the finger without leaving the screen.
  - Activity.Action_UP, the user leaves the screen.

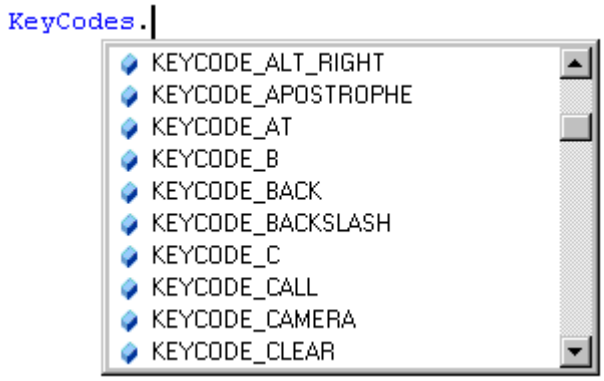
  The X an Y coordinates of the finger position are given.

  Example:
```
Sub Activity_Touch (Action As Int, X As Float, Y As Float)
  Select Action
  Case Activity.ACTION_DOWN
    ' Your code for DOWN action
  Case Activity.ACTION_MOVE
    ' Your code for MOVE action
  Case Activity.ACTION_UP
    ' Your code for UP action
  End Select
End Sub
```

- **CheckChanged** `(Checked As Boolean)`
  Event raised when the user clicks on a CheckBox or a RadioButton
  Checked is equal to True if the view is checked or False if not checked.

  Example:
```
Sub CheckBox1_CheckedChange(Checked As Boolean)
  If Checked = True Then
    ' Your code if checked
  Else
    ' Your code if not checked
  End If
End Sub
```

- **KeyPress** (KeyCode As Int) As Boolean
  Event raised when the user presses a physical or virtual key.
  KeyCode is the code of the pressed key, you can get them with the KeyCodes keyword.



  The event can return either:
  - True, the event is 'consumed', considered by the operating system as already executed and no further action is taken.
  - False, the event is not consumed and transmitted to the system for further actions.

  Example:

```
Sub Activity_KeyPress(KeyCode As Int) As Boolean
  Private Answ As Int
  Private Txt As String

  If KeyCode = KeyCodes.KEYCODE_BACK Then    ' Checks if KeyCode is BackKey
    Txt = "Do you really want to quit the program ?"
    Answ = Msgbox2(Txt,"A T T E N T I O N","Yes","","No",Null)' MessageBox
    If Answ = DialogResponse.POSITIVE Then  ' If return value is Yes then
      Return False     ' Return = False  the Event will not be consumed
    Else               '                  we leave the program
      Return True      ' Return = True   the Event will be consumed to avoid
    End If             '                 leaving the program
  End If
End Sub
```

3.7.2 B4i

User interface objects are called 'Views' in iOS.

Summary of the events for different views:

	Events													
Views	Click	LongClick	BeginEdit	EndEdit	EnterPressed	TextChanged	Touch	Resize	ScrollChanged	ValueChanged	ItemSelected	IndexChanged	OverrideUrl	PageFinished
Button	■	■												
TextField	■	■	■	■	■	■								
TextView	■	■												
ImageView	■	■												
Label	■	■												
Panel	■	■					■	■						
ScrollView	■	■							■					
Slider	■	■								■				
Picker	■	■									■			
Stepper	■	■								■				
Switch	■	■								■				
SegmentedControl	■	■										■		
Slider	■	■								■				
Stepper	■	■								■				
WebView	■	■											■	■

The most common events are:

- **Click** Event raised when the user clicks on the view.
  Example:
```
Private Sub Button1_Click
  ' Your code
End Sub
```

- **LongClick** Event raised when the user clicks on the view and holds it pressed for a while.
  Example:
```
Private Sub Button1_LongClick
  ' Your code
End Sub
```

- **Touch** (Action As Int, X As Float, Y As Float)
  Event raised when the user touches a Panel on the screen.

  Three different actions are handled:
  - Panel.ACTION_DOWN, the user touches the screen.
  - Panel.ACTION_MOVE, the user moves the finger without leaving the screen.
  - Panel.ACTION_UP, the user leaves the screen.

  The X and Y coordinates of the finger positions are given in Points not in Pixels.

  Example:
```
Private Sub Panel_Touch (Action As Int, X As Float, Y As Float)
  Select Action
  Case Panel.ACTION_DOWN
    ' Your code for DOWN action
  Case Panel.ACTION_MOVE
    ' Your code for MOVE action
  Case Panel.ACTION_UP
    ' Your code for UP action
  End Select
End Sub
```

3.7.3 B4J

User interface objects are called 'Nodes' in Java.

Summary of the events for different nodes:

			Events													
Nodes	Action	FocusChanged	MouseClicked	MouseDragged	MouseMoved	MousePressed	MouseReleased	Resize	CheckedChange	SelectedIndexChanged	ValueChange	SelectedChange	H / VScrollChanged	TabChanged	TextChanged	PageFinished
Button	■	■	■	■	■	■	■	■								
Canvas		■	■	■	■	■	■									
CheckBox									■							
ChoiceBox										■						
ComboBox										■	■					
ImageView		■	■	■	■	■	■									
Label		■	■	■	■	■	■	■								
ListView		■	■	■	■	■	■	■		■						
Pane		■	■	■	■	■	■	■								
RadioButton												■				
ScrollPane		■	■	■	■	■	■	■					■			
Slider											■					
Spinner											■					
TabPane		■	■	■	■	■	■	■						■		
TextArea		■	■												■	
TextField	■	■	■												■	
ToggleButton												■				
WebView		■	■	■	■	■	■									■

The most common events are:

- **Action** Event raised when the user clicks on the node (Button or TextField).
  Example:
```
Private Sub Button1_Action
  ' Your code
End Sub
```

- **FocusChanged** (HasFocus As Boolean) Event raised when the node gets or looses focus.
  Example:
```
Private Sub Button1_FocusChanged (HasFocus As Boolean)
  ' Your code
End Sub
```

- **MouseClicked** (EventData As MouseEvent)
  Event raised when the user clicks on the node.
  Example:
```
Private Sub Pane1_MouseClicked (EventData As MouseEvent)
  ' Your code
End Sub
```

- **MouseDragged** (EventData As MouseEvent)
  Event raised when the user draggs over the node (moves with a button pressed).
  Similar to ACTION_MOVE in B4A Touch events.
- Example:
```
Private Sub Pane1_MouseDragged (EventData As MouseEvent)
  ' Your code
End Sub
```

- **MouseMoved** (EventData As MouseEvent)
  Event raised when the user moves over the node (without a button pressed).
  Example:
```
Private Sub Pane1_MouseMoved (EventData As MouseEvent)
  ' Your code
End Sub
```

- **MousePressed** (EventData As MouseEvent)
  Event raised when the user presses on the node.
  Similar to ACTION_DOWN in B4A Touch events.
  Example:
```
Private Sub Pane1_MousePressed (EventData As MouseEvent)
  ' Your code
End Sub
```

- **MouseReleased** (EventData As MouseEvent)
  Event raised when the user releases the node.
  Similar to ACTION_UP in B4A Touch events.
  Example:
```
Private Sub Pane1_MouseReleased (EventData As MouseEvent)
  ' Your code
End Sub
```

- **MouseEvent**
  Data returned by the Mouse events:

  - **ClickCount** Returns the number of clicks associated with this event.
  - **Consume** Consumes the current event and prevent it from being handled by the nodes parent.
  - **MiddleButtonDown** Returns true if the middle button is currently down.
  - **MiddleButtonPressed** Returns true if the middle button was responsible for raising the current click event.
  - **PrimaryButtonDown** Returns true if the primary button is currently down.
  - **PrimaryButtonPressed** Returns true if the primary button was responsible for raising the current click event.
  - **SecondaryButtonDown** Returns true if the secondary button is currently down.
  - **SecondaryButtonPressed** Returns true if the secondary button was responsible for raising the current click event.
  - **X** Returns the X coordinate related to the node bounds.
  - **Y** Returns the Y coordinate related to the node bounds.

3.7.4 B4R

In B4R, the Pin and Timer objects are the only ones raising an event:

- Pin
  StateChanged (State As Boolean) Event raised when the pin changes its state.

  Example:
  ```
  Sub Pin1_StateChanged(State As Boolean)
    ' Your code
  End Sub
  ```

- Timer
  Tick Event raised at every given interval

  Example:
  ```
  Sub Timer1_Tick
    ' Your code
  End Sub
  ```

3.7.5 User interface summary

The ‘standard’ user interface objects.
This shows the difference between the three operating systems.
Some views / nodes which don’t exist as standard objects can exis as CustomViews in other operating systems. You should look in the forums.

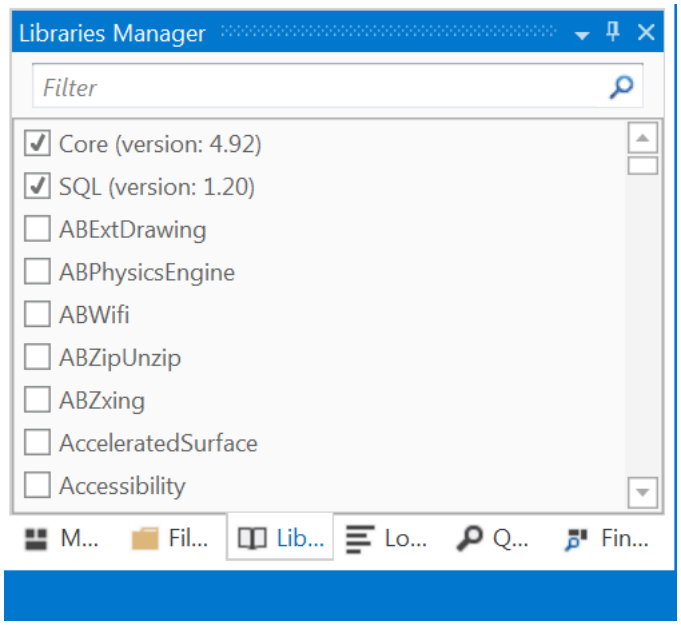
View / node	B4A	B4i	B4J
Activity	■		
Button	■	■	■
CheckBox	■		■
EditText	■		
HorizontalScrollView	■		
ImageView	■	■	■
Label	■	■	■
ListView	■		■
Panel	■	■	
RadioButton	■		■
ScrollView	■		
SeekBar	■		
Spinner	■		■
TabHost	■		
ToggleButton	■		■
WebView	■	■	■
TextField		■	■
TextView		■	
ScrollView different from B4A 2D		■	
Slider		■	■
Picker		■	
Stepper		■	
Switch		■	
SegmentedControl		■	
Canvas a node on its own			■
ChoiceBox			■
ComboBox			■
Pane similar to Panel in B4A and B4i			■
ScrollPane similar to ScrollView			■
TabPane			■
TextArea			■

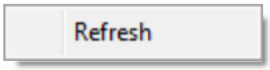
3.8 Libraries

Libraries add more objects and functionalities to B4x.
Some of these libraries are shipped with the B4x products and are part of the standard development system.
Other, often developed by users, can be downloaded (by registered users only) to add supplementary functionalities to the B4x development environments.

When you need a library, you have to:

- Check it in the Libs Tab, if you already have the library.
- For additional libraries, check if it's the latest version.
  You can check the versions in the documentation page
  B4A, B4i, B4J, B4R
  To find the library files use a query like
  http://www.b4x.com/search?query=betterdialogs+library
  in your internet browser.
- If **yes**, then check the library in the list to select it.



- If **no**, download the library, unzip it and copy the <LibraryName>.jar and <LibraryName>.xml files to the additional libraries folder.
- Right click in the Lib area and click on 

and check the library in the list to select it.

3.8.1 Standard libraries

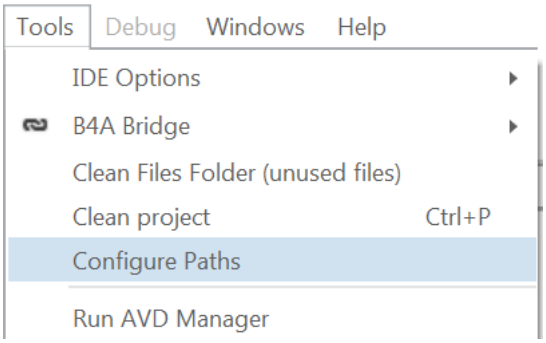
The standard B4x libraries are saved in the Libraries folder in the B4x program folder.
Normally in:
C:\Program Files\Anywhere Software\B4A\Libraries
C:\Program Files\Anywhere Software\B4i\Libraries
C:\Program Files\Anywhere Software\B4J\Libraries
C:\Program Files\Anywhere Software\B4R\Libraries

3.8.2 Additional libraries folder

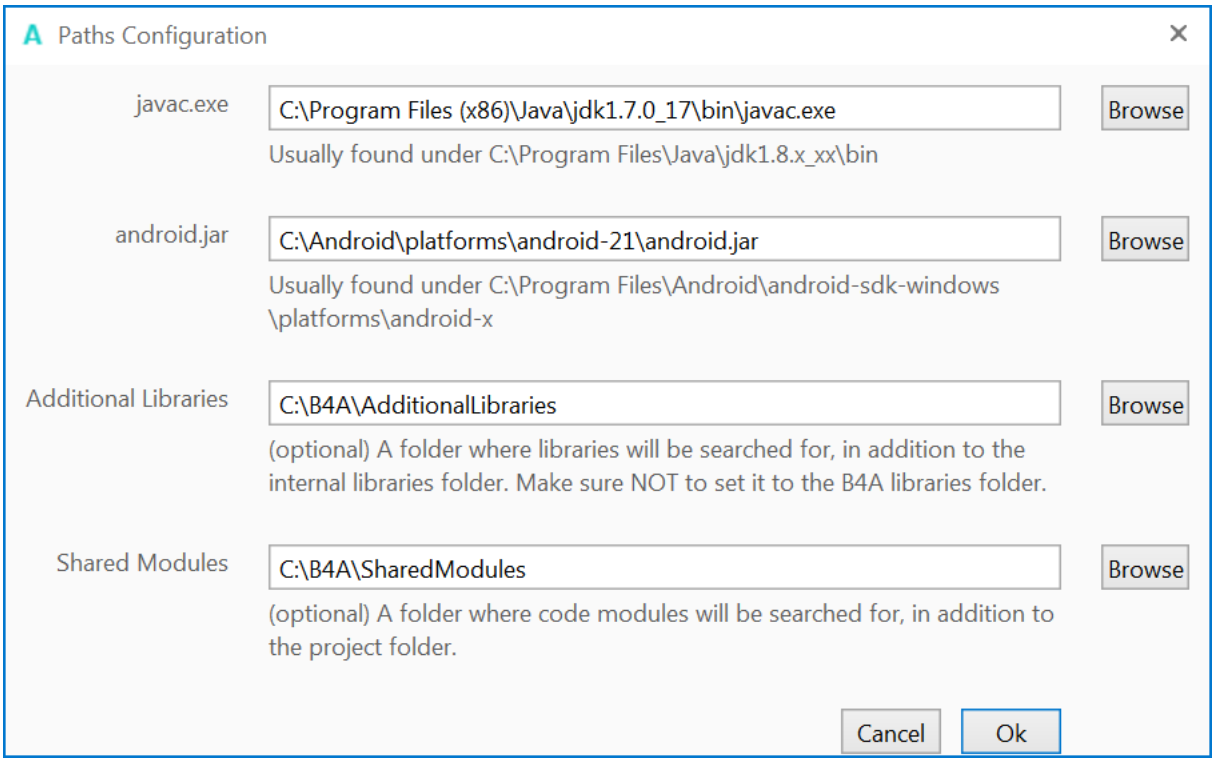
For the additional libraries it is useful to setup a special folder to save them somewhere else.
For example:
C:\B4A\AddLibraries, C:\B4i\AddLibraries, C:\B4J\AddLibraries, C:\B4R\AddLibraries

When you install a new version of B4x, all standard libraries are automatically updated, but the additional libraries are not included. The advantage of the special folder is that you don't need to care about them because this folder is not affected when you install the new version of B4x.
The additional libraries are not systematically updated with new version of B4x.

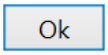
When the IDE starts, it looks first for the available libraries in the Libraries folder of B4x and then in the folder for the additional libraries.



To setup the special additional libraries folder click in the IDE menu on Tools / Configure Paths.



Example for B4A. It is similar for the other B4x products.

Enter the folder name and click on 

.

3.8.3 Load and update a Library

A list of the official and additional libraries with links t.o the relevant help documentation can be found on the B4x site in the:
B4A Documentation page: List of Libraries.
B4i Documentation page: List of Libraries.
B4J Documentation page: List of Libraries.
B4RA Documentation page: List of Libraries.

To find the library files use a query like http://www.b4x.com/search?query=betterdialogs+library in your internet browser.

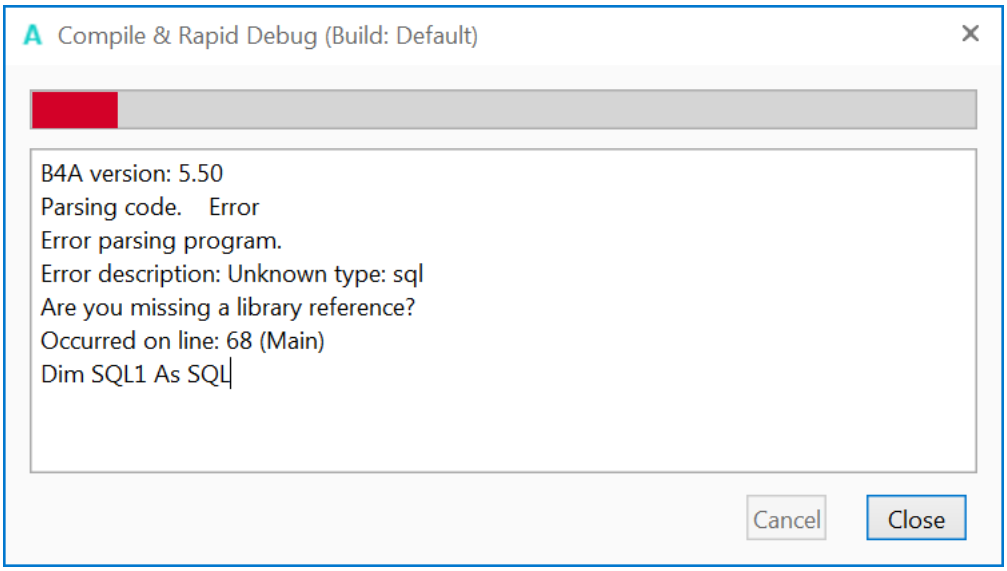
To load or update a library follow the steps below:

- Download the library zip file somewhere.
- Unzip it.
- Copy the xxx.jar and xxx.xml files to the
  - B4x Library folder for a standard B4x library
  - Additional libraries folder for an additional library.
- Right click in the libraries list in the Lib Tab and click on 

and select the library.

3.8.4 Error message "Are you missing a library reference?"

If you get a message similar to this, means that you forgot to check the specified library in the Lib Tab list !



3.9 String manipulation

3.9.1 B4A, B4i, B4J

B4A, B4i and B4J allow string manipulations like other Basic languages but with some differences.

These manipulations can be done directly on a string.
Example:

```
txt = "123,234,45,23"
txt = txt.Replace(",", ";")
```

Result: `123;234;45;23`

The different functions are:

- **CharAt(Index)** Returns the character at the given index.
- **CompareTo(Other)** Lexicographically compares the string with the Other string.
- **Contains(SearchFor)** Tests whether the string contains the given SearchFor string.
- **EndsWith(Suffix)** Returns True if the string ends with the given Suffix substring.
- **EqualsIgnoreCase(Other)** Returns True if both strings are equal ignoring their case.
- **GetBytes(Charset)** Encodes the Charset string into a new array of bytes.
- **IndexOf(SearchFor)** Returns the index of the first occurrence of SearchFor in the string. The index is 0 based. Returns -1 if no occurrence is found.
- **IndexOf2(SearchFor, Index)** Returns the index of the first occurrence of SearchFor in the string. Starts searching from the given index.
  The index is 0 based. Returns -1 if no occurrence is found.
- **LastIndexOf(SearchFor)** Returns the index of the first occurrence of SearchFor in the string. The search starts at the end of the string and advances to the beginning.
  The index is 0 based. Returns -1 if no occurrence is found.
- **LastIndexOf2(SearchFor)** Returns the index of the first occurrence of SearchFor in the string. The search starts at the given index and advances to the beginning.
  The index is 0 based. Returns -1 if no occurrence is found.
- **Length** Returns the length, number of characters, of the string.
- **Replace(Target, Replacement)** Returns a new string resulting from the replacement of all the occurrences of Target with Replacement.
- **StartsWith(Prefix)** Returns True if this string starts with the given Prefix.
- **Substring(BeginIndex)** Returns a new string which is a substring of the original string. The new string will include the character at BeginIndex and will extend to the end of the string.
- **Substring2(BeginIndex, EndIndex)** Returns a new string which is a substring of the original string. The new string will include the character at BeginIndex and will extend to the character at EndIndex, not including the last character.
  Note that EndIndex is the end index and not the length like in other languages.
- **ToLowerCase** Returns a new string which is the result of lower casing this string.
- **ToUpperCase** Returns a new string which is the result of upper casing this string.
- **Trim** Returns a copy of the original string without any leading or trailing white spaces.

Note: The string functions are case sensitive.
If you want to use case insensitive functions you should use either ToLowerCase or ToUpperCase.

Example: `NewString = OriginalString.ToLowerCase.StartsWith("pre")`

3.9.2 B4R

B4R doesn’t support string manipulations like other Basic languages.

These kind of manipulations can be done with the ByteConverter object in the rRandomAccesFile library.

B4R strings are different than in other B4x tools. The reasons for these differences are:
- Very limited memory.
- Lack of Unicode encoders.

A String object in B4R is the same as a C language char* string. It is an array of bytes with an additional zero byte at the end.
The requirement of the last zero byte makes it impossible to create a substring without copying the memory to a new address.
For that reason, arrays of bytes are preferable over Strings.
The various string related methods work with arrays of bytes.

Converting a string to an array of bytes is very simple and doesn't involve any memory copying. The compiler will do it automatically when needed:

```
Private b() As Byte = "abc" 'equivalent to Private b() As Byte = "abc".GetBytes
```

Only two functions are supported:

These functions are:

- **GetBytes(Charset)** Returns the string content as an array of bytes.
  Note that the array and string share the same memory
- **Length** Returns the length, number of characters, of the string.

String Methods

The standard string methods are available in ByteConverter type (rRandomAccessFile library).

They are similar to the string methods in other B4x tools:

```
Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")
   Dim bc As ByteConverter
   Log("IndexOf: ", bc.IndexOf("0123456", "3")) 'IndexOf: 3
   Dim b() As Byte = " abc,def,ghijkl "
   Log("Substring: ", bc.SubString(b, 3)) 'Substring: c,def,ghijkl
   Log("Trim: ", bc.Trim(b)) 'Trim: abc,def,ghijkl
   For Each s() As Byte In bc.Split(b, ",")
     Log("Split: ", s)
     'Split: abc
     'Split: def
     'Split: ghijkl
   Next
   Dim c As String = JoinStrings(Array As String("Number of millis: ", Millis, CRLF, "N
umber of micros: ", Micros))
   Log("c = ", c)
   Dim b() As Byte = bc.SubString2(c, 0, 5)
   b(0) = Asc("X")
   Log("b = ", b)
   Log("c = ", c) 'first character will be X
End Sub
```

Note how both strings and array of bytes can be used as the compiler converts strings to arrays of bytes automatically.

With the exception of JoinStrings, none of the above methods make a copy of the original string / bytes.
This means that modifying the returned array as in the last three lines will also modify the original array.

It will also happen with string literals that all share the same memory block:

```
Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")
   Dim bc As ByteConverter
   Dim b() As Byte = bc.Trim("abcdef ")
   b(0) = Asc("M") 'this line will change the value of the literal string
   Dim s as String = "abcdef "
   Log(s) 'Mbcdef
End Sub
```

String manipulations in the ByteConverte object:

- **EndsWith(Source As Byte(), Suffix As Byte())**
  Returns True if the string ends with the given Suffix substring.
- **IndexOf(Source As Byte(), SearchFor As Byte())**
  Returns the index of the first occurrence of SearchFor in the string.
- **IndexOf2(Source As Byte(), SearchFor As Byte(), Index As UInt)**
  Returns the index of the first occurrence of SearchFor in the string. Starts searching from the given index.
- **LastIndexOf(Source As Byte(), SearchFor As Byte())**
  Returns the index of the first occurrence of SearchFor in the string. Starts searching from the end of the string.
- **LastIndexOf2(Source As Byte(), SearchFor As Byte(), Index As UInt)**
  Returns the index of the first occurrence of SearchFor in the string. Starts searching from the given index and advances to the beginning.
- **StartsWith(Source As Byte(), Prefix As Byte())**
  Returns True if this string starts with the given Prefix.
- **Substring(Source As Byte(), BeginIndex As UInt)**
  Returns a new string which is a substring of the original string.
  The new string will include the character at BeginIndex and will extend to the end of the string.
- **Substring2(Source As Byte(), BeginIndex As UInt, EndIndex As UInt)**
  Returns a new string which is a substring of the original string. The new string will include the character at BeginIndex and will extend to the character at EndIndex, not including the last character.
- **Trim(Source As Byte())**
  Returns a copy of the original string without any leading or trailing white spaces.

Number formatting, display numbers as strings with different formats:

- **NumberFormat**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int)
  `NumberFormat(12345.6789, 0, 2)` = 12,345.68
  `NumberFormat(1, 3 ,0)` = 001
  `NumberFormat(Value, 3 ,0)` variables can be used.
  `NumberFormat(Value + 10, 3 ,0)` arithmetic operations can be used.
  `NumberFormat((lblscore.Text + 10), 0, 0)` if one variable is a string add parentheses.

3.10 Number formatting

3.10.1 B4A, B4i, B4J

Number formatting, display numbers as strings with different formats, there are two keywords:

- **NumberFormat**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int)
  `NumberFormat(12345.6789, 0, 2)` = 12,345.68
  `NumberFormat(1, 3 ,0)` = 001
  `NumberFormat(Value, 3 ,0)` variables can be used.
  `NumberFormat(Value + 10, 3 ,0)` arithmetic operations can be used.
  `NumberFormat((lblscore.Text + 10), 0, 0)` if one variable is a string add parentheses.

- **NumberFormat2**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean)
  `NumberFormat2(12345.67, 0, 3, 3, True)` = 12,345.670

3.10.2 B4R

Number formatting, display numbers as strings with different formats:

- **NumberFormat**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int)
  `NumberFormat(12345.6789, 0, 2)` = 12,345.68
  `NumberFormat(1, 3 ,0)` = 001
  `NumberFormat(Value, 3 ,0)` variables can be used.
  `NumberFormat(Value + 10, 3 ,0)` arithmetic operations can be used.
  `NumberFormat((lblscore.Text + 10), 0, 0)` if one variable is a string add parentheses.

3.11 Timers

A Timer object generates Tick events at specified intervals. Using a timer is a good alternative to a long loop, as it allows the UI thread to handle other events and messages.
Note that the timer events will not fire while the UI thread is busy running other code (unless you call DoEvents keyword).
Timer events will not fire when the activity is paused, or if a blocking dialog (like Msgbox) is visible.
It is also important to disable the timer when the activity is pausing and then enable it when it resumes. This will save CPU and battery.

A timer has:

- Three parameters.
  - **Initialize** Initializes the timer with two parameters, the EventName and the interval.
    Timer1.Initialize(EventName As String, Interval As Long)
    Ex: `Timer1.Initialize("Timer1", 1000)`

  - **Interval** Sets the timer interval in milli-seconds.
    Timer1. Interval = Interval
    Ex: `Timer1.Interval = 1000`, 1 second

  - **Enabled** Enables or disables the timer. **It is False by default.**
    Ex: `Timer1.Enabled = True`

- One Event
  - **Tick** The Tick routine is called every time interval.
    Ex: `Sub Timer1_Tick`

The Timer must be declared in a Process_Global routine.

```
Sub Process_Globals
   Public Timer1 As Timer
```

But it must be initialized in one of the following routines in the module where the timer tick event routine is used.

B4A: Activity_Create routine

```
Sub Activity_Create(FirstTime As Boolean)
   If FirstTime = True Then
      Timer1.Initialize("Timer1", 1000)
   End If
```

B4i: Application_Start routine

```
Private Sub Application_Start (Nav As NavigationController)
   Timer1.Initialize("Timer1", 1000)
```

B4J: Activity_Create routine

```
Sub AppStart (Form1 As Form, Args() As String)
   Timer1.Initialize("Timer1", 1000)
```

B4R: Activity_Create routine

```
Private Sub AppStart
   Timer1.Initialize("Timer1", 1000)
```

And the Timer Tick event routine.
This routine will be called every second (1000 milli-seconds) by the operating system.

```
Sub Timer1_Tick
   ' Do something
End Sub
```

3.12 Files B4A, B4i, B4J

Many applications require access to a persistent storage. The two most common storage types are files and databases.

Android and iOS have their own file system. The B4A nor the B4i program has access to files in the Windows system,

To add files to your project you must add those in the IDE in the Files Tab. These files will be added to the project Files folder.

3.12.1 File object

The predefined object `File` has a number of functions for working with files.

Files locations - There are several important locations where you can read or write files.

File.DirAssets
The assets folder includes the files that were added with the file manager in the IDE.
It's the Files folder in the project folder.
These files are read-only.
You can not create new files in this folder (which is actually located inside the apk file).
If you have a database file in the Dir.Assets folder you need to copy it to another folder before you can use it.

File.DirInternal / File.DirInternalCache
These two folders are stored in the main memory of the device and are private to your application.
Other applications cannot access these files.
The cache folder may get deleted by the OS if it needs more space.

File.DirRootExternal
The storage card root folder.

File.DirDefaultExternal
The default folder for your application in the SD card.
The folder is: <storage card>/Android/data/<package>/files/
It will be created if required.

Note that calling any of the two above properties will add the EXTERNAL_STORAGE permission to your application.

Tip: You can check if there is a storage card and whether it is available with
File.ExternalReadable and **File.ExternalWritable**.

To check if a file already exists use:
File.Exists (Dir As String, FileName As String)
Returns True if the file exists and False if not.

The File object includes several methods for writing to files and reading from files.
To be able to write to a file or to read from a file, it must be opened.

File.OpenOutput (Dir As String, FileName As String, Append As Boolean)
- Opens the given file for output, the Append parameter tells whether the text will be added at the end of the existing file or not. If the file doesn't exist it will be created.

File.OpenInput (Dir As String, FileName As String)
- Opens the file for reading.

File.WriteString (Dir As String, FileName As String, Text As String)
- Writes the given text to a new file.

File.ReadString (Dir As String, FileName As String) As String
- Reads a file and returns its content as a string.

File.WriteList (Dir As String, FileName As String, List As List)
- Writes all values stored in a list to a file. All values are converted to string type if required. Each value will be stored in a separare line.
Note that if a value contains the new line character it will saved over more than one line and when you read it, it will be read as multiple items.

File.ReadList (Dir As String, FileName As String) As List
- Reads a file and stores each line as an item in a list.

File.WriteMap (Dir As String, FileName As String, Map As Map)
- Takes a map object which holds pairs of key and value elements and stores it in a text file. The file format is known as Java Properties file: .properties - Wikipedia, the free encyclopedia
The file format is not too important unless the file is supposed to be edited manually. This format makes it easy to edit it manually.
One common usage of File.WriteMap is to save a map of "settings" to a file.

File.ReadMap (Dir As String, FileName As String) As Map
- Reads a properties file and returns its key/value pairs as a Map object. Note that the order of entries returned might be different than the original order.

Some other useful functions:

File.Copy (DirSource As String, FileSource As String, DirTarget As String, FileTarget As String)
- Copies the source file from the source directory to the target file in the target directory.
Note that it is not possible to copy files to the Assets folder.

File.Delete (Dir As String, FileName As String)
- Deletes the given file from the given directory.

File.ListFiles (Dir As String) As List
- Lists the files and subdirectories in the diven directory.
Example:
```
Private List1 As List
List1 = File.ListFiles(File.DirRootExternal)
```
List1 can be declared in Sub Globals

File.Size (Dir As String, FileName As String)
- Returns the size in bytes of the specified file.
This method does not support files in the assets folder.

3.12.2 Filenames

B4x file names allow following characters :
a to **z**, **A** to **Z**, **0** to **9** dot **.** underscore **_** and even following characters **+ - % &**
Spaces and following characters *** ?** are not allowed.

Example : MyFile.txt

Note that B4x file names are case sensitive !
`MyFile.txt` is different from `myfile.txt`

3.12.3 Subfolders

You can define subfolders in B4x with.

```
File.MakeDir(File.DirInternal, "Pictures")
```

To access the subfolder you should add the subfoldername to the foldername with "/" inbetween.

```
ImageView1.Bitmap = LoadBitmap(File.DirInternal & "/Pictures", "test1.png")
```

Or add the subfoldername before the filename with "/" inbetween.

```
ImageView1.Bitmap = LoadBitmap(File.DirInternal, "Pictures/test1.png")
```

Both possibilities work.

3.12.4 TextWriter

There are two other useful functions for text files: **TextWriter** and TextReader:

TextWriter.Initialize (OutputStream As OutputStream)
- Initializes a TextWriter object as an output stream.

Example:
```
Private Writer As TextWriter
Writer.Initialize(File.OpenOutput(File.DirRootExternal, "Test.txt" , False))
```

Writer could be declared in Sub Globals.

TextWriter.Initialize2 (OutputStream As OutputStream , Encoding As String)
- Initializes a TextWriter object as as output stream.
- Encoding indicates the CodePage (also called CharacterSet), the text encoding (see next chapter).

Example:
```
Private Writer As TextWriter
Writer.Initialize2(File.OpenOutput(File.DirRootExternal,"Test.txt" ,False)," ISO-8859-1")
```

Writer could be declared in Sub Globals.
See : Text encoding

TextWriter.Write (Text As String)
- Writes the given Text to the stream.

TextWriter.WriteLine (Text As String)
- Writes the given Text to the stream followed by a new line character LF Chr(10).

TextWriter.WriteList (List As List)
- Writes each item in the list as a single line.
Note that a value containing CRLF will be saved as two lines (which will return two items when reading with ReadList).
All values will be converted to strings.

TextWriter.Close
- Closes the stream.

Example:

```
Private Writer As TextWriter
Writer.Initialize(File.OpenOutput(File.DirDefaultExternal, "Text.txt", False))
Writer.WriteLine("This is the first line")
Writer.WriteLine("This is the second line")
Writer.Close
```

3.12.5 TextReader

There are two other useful functions for text files: TextWriter and **TextReader**:

TextReader.Initialize (InputStream As InputStream)
- Initializes a TextReader as an input stream.

Example:
```
Private Reader  TextReader
Reader.Initialize(File.InputOutput(File.DirRootExternal, "Test.txt"))
```

Reader could be declared in Sub Globals.

TextReader.Initialize2 (InputStream As InputStream, Encoding As String)
- Initializes a TextReader as an input stream.
- Encoding indicates the CodePage (also called CharacterSet), the text encoding.

Example:
```
Private Reader  TextReader
Reader.Initialize2(File.OpenInput(File.DirRootExternal, "Test.txt", "ISO-8859-1")
```

Reader could be declared in Sub Globals.
See : Text encoding

TextReader.ReadAll As String
- Reads all of the remaining text and closes the stream.

Example:
```
txt = Reader.ReadAll
```

TextReader.ReadLine As String
- Reads the next line from the stream.
The new line characters are not returned.
Returns Null if there are no more characters to read.

Example:

```
Private Reader As TextReader
Reader.Initialize(File.OpenInput(File.DirDefaultExternal, "Text.txt"))
Private line As String
line = Reader.ReadLine
Do While line <> Null
   Log(line)
   line = Reader.ReadLine
Loop
Reader.Close
```

TextReader.ReadList As List
- Reads the remaining text and returns a List object filled with the lines.
Closes the stream when done.

Example:
```
List1 = Reader.ReadList
```

3.12.6 Text encoding

Text encoding or character encoding consists of a code that pairs each character from a given repertoire with something else. Other terms like character set (charset), and sometimes character map or code page are used almost interchangeably (source Wikipedia).

The default character set in Android is Unicode UTF-8.

In Windows the most common character sets are ASCII and ANSI.

- ASCII includes definitions for 128 characters, 33 are non-printing control characters (now mostly obsolete) that affect how text and space is processed.
- ANSI, Windows-1252 or CP-1252 is a character encoding of the Latin alphabet, used by default in the legacy components of Microsoft Windows in English and some other Western languages with 256 definitions (one byte). The first 128 characters are the same as in the ASCII encoding.

Many files generated by Windows programs are encoded with the ANSI character-set in western countries. For example: Excel csv files, Notepad files by default.
But with Notepad, files can be saved with *UTF-8* encoding.

Android can use following character sets:

- UTF-8 default character-set
- UTF -16
- UTF - 16 BE
- UTF - LE
- US-ASCII ASCII character set
- ISO-8859-1 almost equivalent to the ANSI character-set
- Windows-1251 cyrillic characters
- Windows-1252 latin alphabet

To read Windows files encoded with ANSI you should use the *Windows-1252* character-set.
If you need to write files for use with Windows you should also use the *Windows-1252* character-set.

Another difference between Windows and Android is the end of line character:

- Android, only the LF (Line Feed) character Chr(10) is added at the end of a line.
- Windows, two characters CR (Carriage Return Chr(13)) and LF Chr(10) are added at the end of a line. If you need to write files for Windows you must add CR yourself.

The symbol for the end of line is :

- B4x CRLF Chr(10)
- Basic4PPC CRLF Chr(13) & Chr(10)

To read or write files with a different encoding you must use the TextReader or TextWriter objects with the Initialize2 methods. Even for reading csv files.

Tip for reading Excel csv files:
You can either:

- On the desktop, load the csv file in a text editor like *NotePad* or *Notepad++*
- Save the file with *UTF-8* encoding
  With *Notepad++* use Encode in UTF-8 without BOM, see below.

Or

- Read the whole file with TextReader.Initialize2 and "Windows-1252" encoding.
- Save it back with TextWriter.Initialize with the standard Android encoding.
- Read the file with LoadCSV or LoadCSV2 from the StringUtils library.

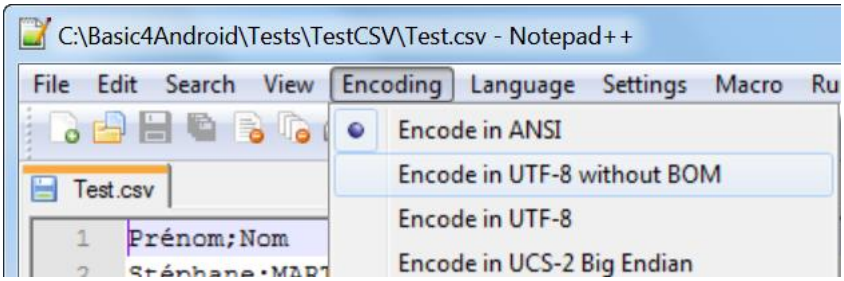
```
Private txt As String
Private tr As TextReader
tr.Initialize2(File.OpenInput(File.DirAssets, "TestCSV1_W.csv"), "Windows-1252")
txt = tr.ReadAll
tr.Close

Private tw As TextWriter
tw.Initialize(File.OpenOutput(File.DirInternal, "TestCSV1_W.csv", False))
tw.Write(txt)
tw.Close

lstTest = StrUtil.LoadCSV2(File.DirInternal, "TestCSV1_W.csv", ";", lstHead)
```

When you save a file with NotePad three additional bytes are added .
These bytes are called BOM characters (Byte Order Mark).
In *UTF-8* they are represented by this byte sequence : `0xEF,0xBB,0xBF`.
A text editor or web browser interpreting the text as *Windows-1252* will display the characters `ï»¿`.

To avoid this you can use *Notepad++* instead of *NotePad* and use Encode in *UTF-8* without BOM.



Another possibility to change a text from *Windows-1252* to *UTF-8* is to use the code below.

```
Private var, result As String
var = "Gestió"
Private arrByte() As Byte
arrByte = var.GetBytes("Windows-1252")
result = BytesToString(arrByte, 0, arrByte.Length, "UTF8")
```

3.13 Lists B4A, B4i and B4J only

Lists are similar to dynamic arrays.

A List must be initialized before it can be used.

- Initialize Initializes an empty List.
```
Private List1 As List
List1.Initialize
List1.AddAll(Array As Int(1, 2, 3, 4, 5))
```

- Initialize2 (SomeArray)
Initializes a list with the given values. This method should be used to convert arrays to lists. Note that if you pass a list to this method then both objects will share the same list, and if you pass an array the list will be of a fixed size.
Meaning that you cannot later add or remove items.
Example 1:
```
Private List1 As List
List1.Initialize2(Array As Int(1, 2, 3, 4, 5))
```
Example 2:
```
Private List1 As List
Private SomeArray(10) As String
' Fill the array
List1.Initialize2(SomeArray)
```

You can add and remove items from a list and it will change its size accordingly.
With either:

- Add (item As Object)
Adds a value at the end of the list.
```
List1.Add(Value)
```

- AddAll (Array As String("value1", "value2"))
Adds all elements of an array at the end of the list.
```
List1.AddAll(List2)
List1.AddAll(Array As Int(1, 2, 3, 4, 5))
```

- AddAllAt (Index As Int, List As List)
Inserts all elements of an array in the list starting at the given position.
```
List1.AddAll(12, List2)
List1.AddAllAt(12, Array As Int(1, 2, 3, 4, 5))
```

- InsertAt (Index As Int, Item As Object)
Inserts the specified element in the specified index.
As a result all items with index larger than or equal to the specified index are shifted.
```
List1.InsertAt(12, Value)
```

- RemoveAt (Index As Int)
Removes the specified element at the given position from the list.
```
List1.RemoveAt(12)
```

A list can hold any type of object. However if a list is declared as a process global object it cannot hold activity objects (like views).
B4x automatically converts regular arrays to lists. So when a List parameter is expected you can pass an array instead.

Get the size of a List:

- `List1.Size`

Use the Get method to get an item from the list with (List indexes are 0 based):
To get the first item use Get(0).
To get the last item use Get(List1.Size - 1).

- Get(Index As Int)

```
number = List1.Get(i)
```

You can use a For loop to iterate over all the values:

```
For i = 0 To List1.Size - 1
   Private number As Int
   number = List1.Get(i)
   ...
Next
```

Lists can be saved and loaded from files with:

- File.WriteList(Dir As String, FileName As String, List As List)

```
File.WriteList(File.DirRootExternal, "Test.txt", List1)
```

- File.ReadList (Dir As String, FileName As String)

```
List1 = File.ReadList(File.DirRootExternal, "Test.txt")
```

A single item can be changed with :

- List1. Set(Index As Int, Item As Object)

```
List1.Set(12, Value)
```

A List can be sorted (the items must all be numbers or strings) with :

- Sort(Ascending As Boolean)

```
List1.Sort(True)          sort ascending
List1.Sort(False)         sort descending
```

- SortCaseInsensitive(Ascending As Boolean)

Clear a List with :

- `List1.Clear`

3.14 Maps B4A, B4i and B4J only

A Map is a collection that holds pairs of keys and values.

The keys are unique. Which means that if you add a key/value pair (entry) and the collection already holds an entry with the same key, the previous entry will be removed from the map.

The key should be a string or a number. The value can be any type of object.

Similar to a list, a map can hold any object, however if it is a process global variable then it cannot hold activity objects (like views).

Maps are very useful for storing applications settings.

Maps are used in this example:

- DBUtils module
  used for database entries, keys are the column names and values the column values.

A Map must be initialized before it can be used.

- Initialize Initializes an empty Map.

```
Private Map1 As Map
Map1.Initialize
```

Add a new entry :

- Put(Key As Object, Value As Object)

```
Map1.Put("Language", "English")
```

Get an entry :

- Get(Key As Object)

```
Language = Map1.Get("Language")
```

Get a key or a value at a given index (only B4A and B4J):

Returns the value of the item at the given index.
GetKeyAt and GetValueAt should be used to iterate over all the items.
These methods are optimized for iterating over the items in ascending order.

- GetKeyAt(Index As Int)

```
Key = Map1.GetKeyAt(12)
```

Get a value at a given index (only B4A and B4J):

- GetValueAt(Index As Int)

```
Value = Map1.GetValueAt(12)
```

Check if a Map contains an entry, tests whether there is an entry with the given key :

- ContainsKey(Key As Object)

```
If Map1.ContainsKey("Language") Then
   Msgbox("There is already an entry with this key !", "ATTENTION")
   Return
End If
```

Remove an entry :

- Remove(Key As Object)
```
Map1.Remove("Language")
```

Clear an entry, clears all items from the map :

- Clear
```
Map1.Clear
```

Maps can be saved and loaded with :

- File.WriteMap(Dir As String, FileName As String, Map As Map)
```
File.WriteMap(File.DirInternal, "settings.txt", mapSettings)
```

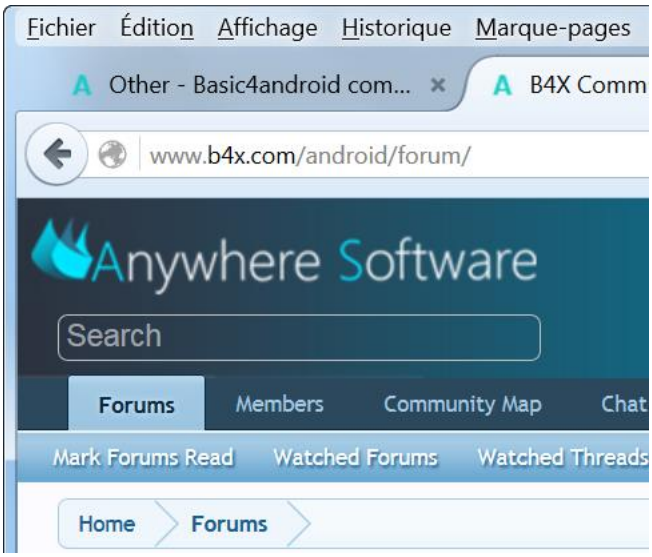
- ReadMap(Dir As String, FileName As String)
  Reads the file and parses each line as a key-value pair (of strings).
  Note that the order of items in the map may not be the same as the order in the file.
```
mapSettings = File.ReadMap(File.DirInternal, "settings.txt")
```

- File.ReadMap2(Dir As String, FileName As String, Map As Map)
  Similar to ReadMap. ReadMap2 adds the items to the given Map.
  By using ReadMap2 with a populated map you can force the items order as needed.
```
mapSettings = File.ReadMap2(File.DirInternal, "settings1.txt", mapSettings)
```

4 Help tools

To find answers to many questions about B4x the following tools are very useful.

4.1 Search function in the forum

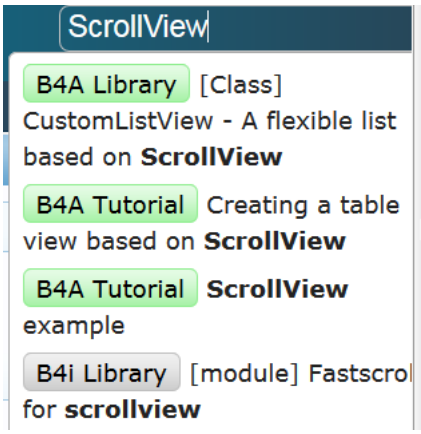


In the upper left corner you find the searchbox for the forum. Depending on the window size it can be on the top right side.

Enter a question or any keywords and press 'Return'.

The function shows you the posts that match your request.

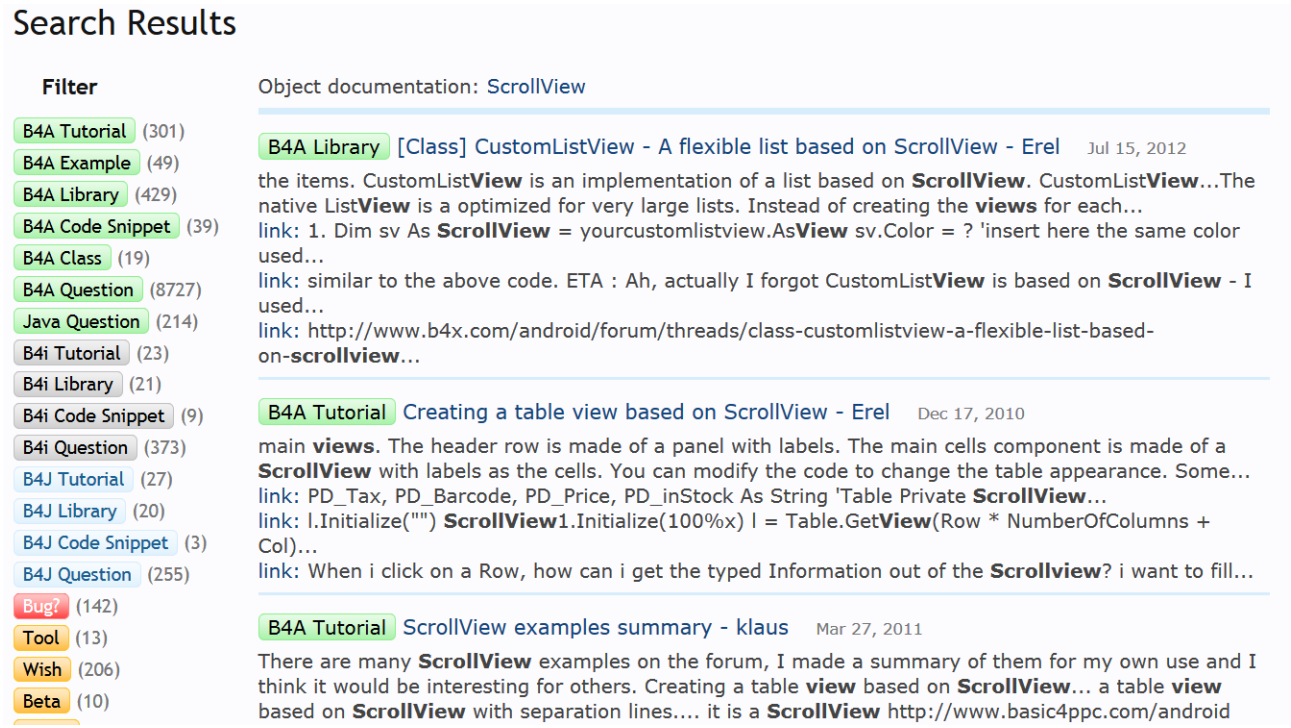
Example: Enter the keyword ScrollView : 



A list of the result is displayed below the search box.

Click on an item to show the thread.

And the result:



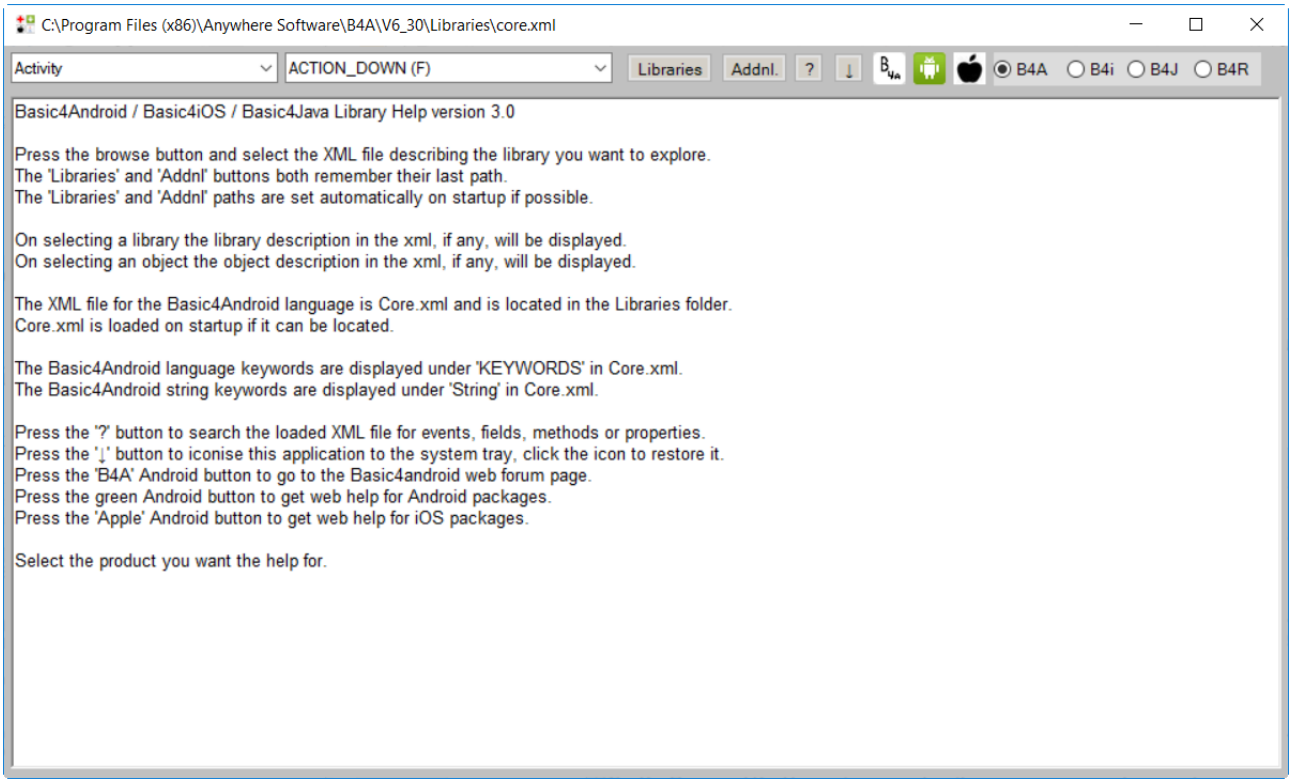
On the left you have a list of forums which you can filter.

Click on the title to show the selected post.

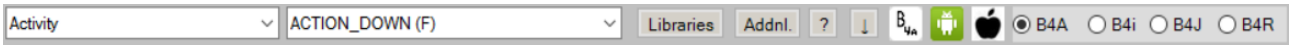
4.2 B4x Help Viewer

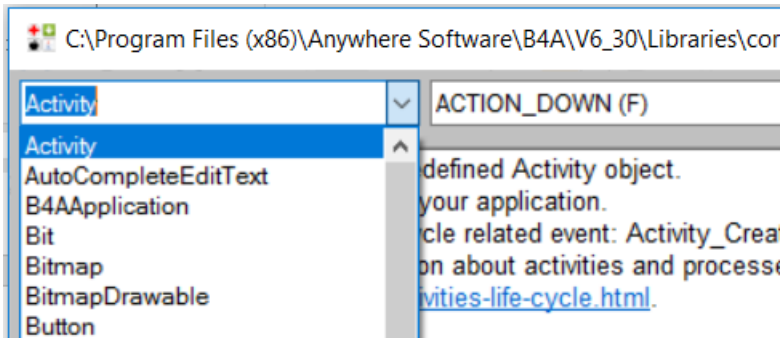
This program shows xml help files. It was originally written by Andrew Graham (agrham) for B4A. I modified it, with Andrews' agreement, to show B4A, B4J, B4i and B4R xml help files.

The program can be downloaded from the forum.

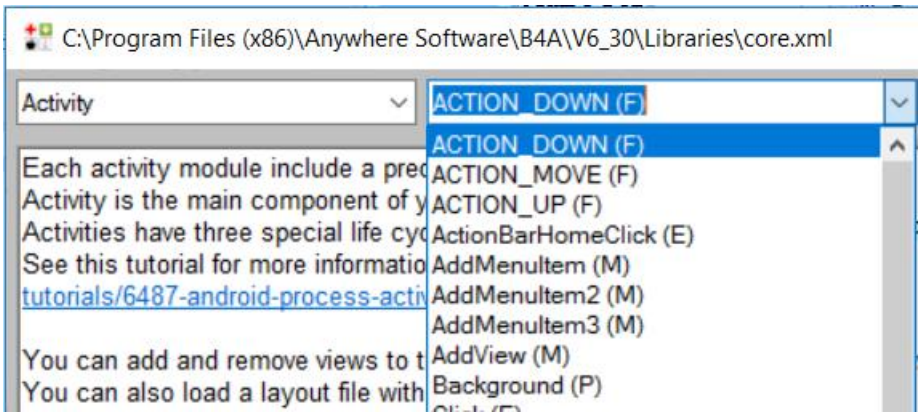


On top we find:





In the upper left corner a drop down list shows the different objects included in the selected library.

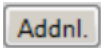


Besides the objects list you find another drop down list with the

- methods(M)
- events(E)
- properties(P)
- fields(F) constants

for the selected object.

Select the standard library to display.



Select the additional library to display.

Search engine to find the object for a given keyword.

Closes B4AHelp

Launches the forum 'Online Community'.

Launches the Android Developers site.

Launches the iOS developer's site.

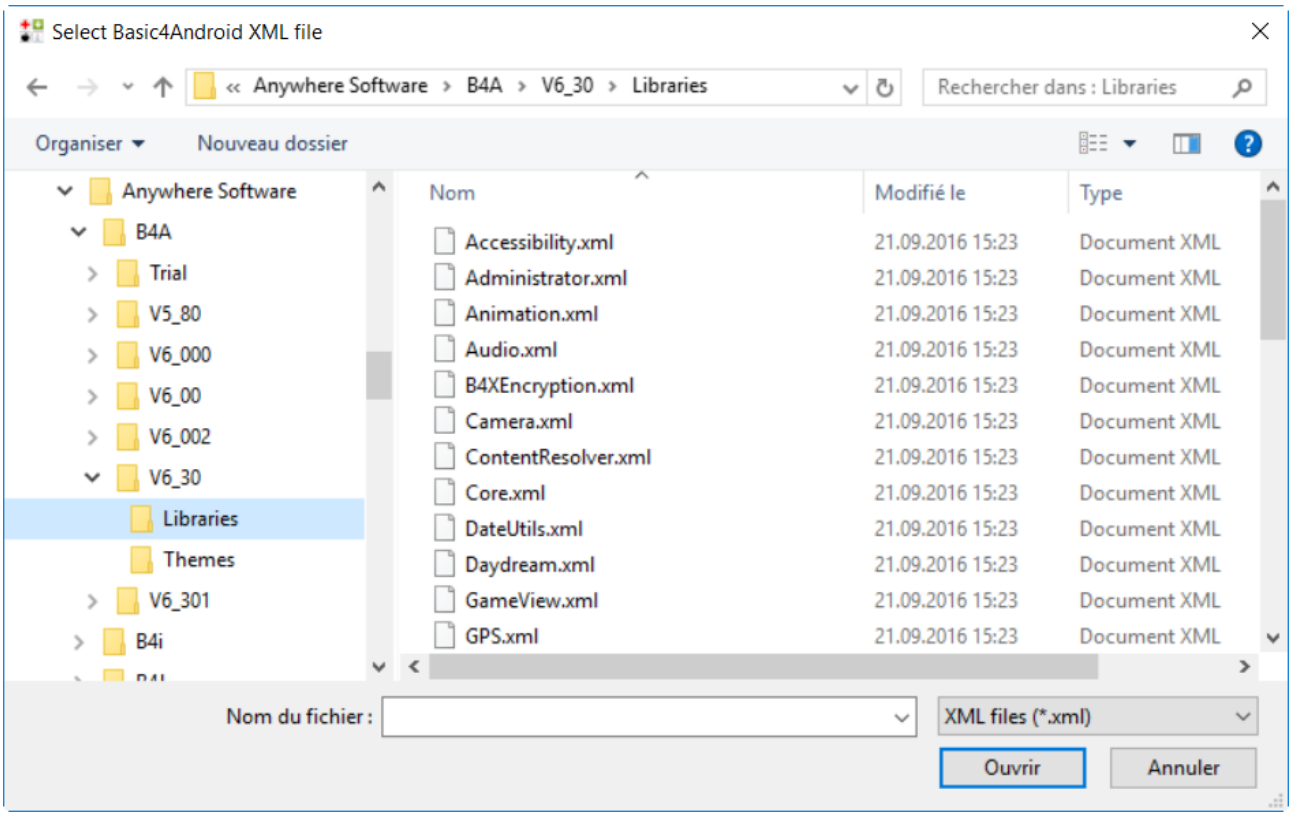
B4A help files.

B4i help files.

B4J help files.

B4R help files.

Libraries Standard libraries



Select the library to display and click on Ouvrir (Open).

Here « Anywhere Software › B4A › V6_30 › Libraries you can select the directory where the standard libraries are saved.

Once selected the directory is saved for the next start of the program.

4.3 Help documentation - B4A Object Browser

This is also a standalone Windows program showing the help files of libraries.

It has been written by Vader and can be downloaded here.

A pdf documentation on how to use the program is part of the download.