B4x Booklets





B4x Basic Language

Copyright: © 2017 Anywhere Software Edition 1.1

Last update : 2017.10.12

[1 BASIC 6](#_Toc495585898)

[1.1 B4x 6](#_Toc495585899)

[1.1.1 B4A Android 6](#_Toc495585900)

[1.1.2 B4i iOS 6](#_Toc495585901)

[1.1.3 B4J Java Desktop 6](#_Toc495585902)

[1.1.4 B4R Arduino 6](#_Toc495585903)

[2 Variables and objects 7](#_Toc495585904)

[2.1 Variable Types 7](#_Toc495585905)

[2.2 Names of variables 10](#_Toc495585906)

[2.3 Declaring variables 10](#_Toc495585907)

[2.3.1 Simple variables 10](#_Toc495585908)

[2.3.2 Array variables 11](#_Toc495585909)

[2.3.3 Array of views / nodes (objects) 13](#_Toc495585910)

[2.3.4 Type variables B4A, B4i and B4J only 16](#_Toc495585911)

[2.4 Casting 17](#_Toc495585912)

[2.5 Scope 18](#_Toc495585913)

[2.5.1 Process variables 18](#_Toc495585914)

[2.5.2 Activity variables B4A only 19](#_Toc495585915)

[2.5.3 Local variables 19](#_Toc495585916)

[2.6 Tips 19](#_Toc495585917)

[3 Program flow / Process life cycle 20](#_Toc495585918)

[3.1 B4A 20](#_Toc495585919)

[3.1.1 Program Start 21](#_Toc495585920)

[3.1.2 Process global variables 22](#_Toc495585921)

[3.1.3 Activity variables 22](#_Toc495585922)

[3.1.4 Starter service 23](#_Toc495585923)

[3.1.5 Program flow 24](#_Toc495585924)

[3.1.6 Sub Process\_Globals / Sub Globals 25](#_Toc495585925)

[3.1.7 Sub Activity\_Create (FirstTime As Boolean) 25](#_Toc495585926)

[3.1.8 Variable declaration summary 26](#_Toc495585927)

[3.1.9 Sub Activity\_Resume Sub Activity\_Pause (UserClosed As Boolean) 27](#_Toc495585928)

[3.1.10 Activity.Finish / ExitApplication 28](#_Toc495585929)

[3.2 Program flow B4i 29](#_Toc495585930)

[3.3 Program flow B4J 30](#_Toc495585931)

[3.4 Program flow B4R 31](#_Toc495585932)

[3.5 Program flow comparison B4A / B4i / B4J 32](#_Toc495585933)

[3.5.1 Program start B4A / B4i / B4J 32](#_Toc495585934)

[3.5.2 Rotating device B4A / B4i 32](#_Toc495585935)

[4 Basic language 33](#_Toc495585936)

[4.1 Expressions 33](#_Toc495585937)

[4.1.1 Mathematical expressions 33](#_Toc495585938)

[4.1.2 Relational expressions 34](#_Toc495585939)

[4.1.3 Boolean expressions 34](#_Toc495585940)

[4.2 Standard keywords 35](#_Toc495585941)

[https://www.b4x.com/android/help/images/Method_636.pngAbs (Number As Double) As Double 37](#_Toc495585942)

[https://www.b4x.com/android/help/images/Method_636.pngACos (Value As Double) As Double 37](#_Toc495585943)

[https://www.b4x.com/android/help/images/Method_636.pngACosD (Value As Double) As Double 37](#_Toc495585944)

[https://www.b4x.com/android/help/images/Method_636.pngArray 37](#_Toc495585945)

[https://www.b4x.com/android/help/images/Method_636.pngAsc (Char As Char) As Int 37](#_Toc495585946)

[https://www.b4x.com/android/help/images/Method_636.pngASin (Value As Double) As Double 37](#_Toc495585947)

[https://www.b4x.com/android/help/images/Method_636.pngASinD (Value As Double) As Double 37](#_Toc495585948)

[https://www.b4x.com/android/help/images/Method_636.pngATan (Value As Double) As Double 37](#_Toc495585949)

[https://www.b4x.com/android/help/images/Method_636.pngATan2 (Y As Double, X As Double) As Double 37](#_Toc495585950)

[https://www.b4x.com/android/help/images/Method_636.pngATan2D (Y As Double, X As Double) As Double 37](#_Toc495585951)

[https://www.b4x.com/android/help/images/Method_636.pngATanD (Value As Double) As Double 37](#_Toc495585952)

[https://www.b4x.com/android/help/images/Method_636.pngBytesToString (Data() As Byte, StartOffset As Int, Length As Int, CharSet As String) As String 38](#_Toc495585953)

[https://www.b4x.com/android/help/images/Method_636.pngCallSub (Component As Object, Sub As String) As Object 38](#_Toc495585954)

[https://www.b4x.com/android/help/images/Method_636.pngCallSub2 (Component As Object, Sub As String, Argument As Object) As Object 38](#_Toc495585955)

[https://www.b4x.com/android/help/images/Method_636.pngCallSub3 (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object) As Object 38](#_Toc495585956)

[https://www.b4x.com/android/help/images/Method_636.pngCatch 38](#_Toc495585957)

[https://www.b4x.com/android/help/images/Field.pngcE As Double 38](#_Toc495585958)

[https://www.b4x.com/android/help/images/Method_636.pngCeil (Number As Double) As Double 38](#_Toc495585959)

[https://www.b4x.com/android/help/images/Method_636.pngCharsToString (Chars() As Char, StartOffset As Int, Length As Int) As String 39](#_Toc495585960)

[https://www.b4x.com/android/help/images/Method_636.pngChr (UnicodeValue As Int) As Char 39](#_Toc495585961)

[https://www.b4x.com/android/help/images/Method_636.pngContinue 39](#_Toc495585962)

[https://www.b4x.com/android/help/images/Method_636.pngCos (Radians As Double) As Double 39](#_Toc495585963)

[https://www.b4x.com/android/help/images/Method_636.pngCosD (Degrees As Double) As Double 39](#_Toc495585964)

[https://www.b4x.com/android/help/images/Field.pngcPI As Double 39](#_Toc495585965)

[https://www.b4x.com/android/help/images/Field.pngCRLF As String 39](#_Toc495585966)

[https://www.b4x.com/android/help/images/Method_636.pngDim 39](#_Toc495585967)

[https://www.b4x.com/android/help/images/Method_636.pngExit 39](#_Toc495585968)

[https://www.b4x.com/android/help/images/Field.pngFalse As Boolean 40](#_Toc495585969)

[https://www.b4x.com/android/help/images/Method_636.pngFloor (Number As Double) As Double 40](#_Toc495585970)

[https://www.b4x.com/android/help/images/Method_636.pngFor 40](#_Toc495585971)

[https://www.b4x.com/android/help/images/Method_636.pngGetType (object As Object) As String 40](#_Toc495585972)

[https://www.b4x.com/android/help/images/Method_636.pngIf 40](#_Toc495585973)

[https://www.b4x.com/android/help/images/Method_636.pngIsNumber (Text As String) As Boolean 40](#_Toc495585974)

[https://www.b4x.com/android/help/images/Method_636.pngLoadBitmap (Dir As String, FileName As String) As Bitmap 41](#_Toc495585975)

[https://www.b4x.com/android/help/images/Method_636.pngLoadBitmapSample (Dir As String, FileName As String, MaxWidth As Int, MaxHeight As Int) As Bitmap 41](#_Toc495585976)

[https://www.b4x.com/android/help/images/Method_636.pngLog (Message As String) 41](#_Toc495585977)

[https://www.b4x.com/android/help/images/Method_636.pngLogarithm (Number As Double, Base As Double) As Double 41](#_Toc495585978)

[https://www.b4x.com/android/help/images/Method_636.pngLogColor (Message As String, Color As Int) 41](#_Toc495585979)

[https://www.b4x.com/android/help/images/Method_636.pngMax (Number1 As Double, Number2 As Double) As Double 41](#_Toc495585980)

[https://www.b4x.com/android/help/images/Method_636.pngMe As Object 41](#_Toc495585981)

[https://www.b4x.com/android/help/images/Method_636.pngMin (Number1 As Double, Number2 As Double) As Double 41](#_Toc495585982)

[https://www.b4x.com/android/help/images/Method_636.pngNot (Value As Boolean) As Boolean 41](#_Toc495585983)

[https://www.b4x.com/android/help/images/Field.pngNull As Object 41](#_Toc495585984)

[https://www.b4x.com/android/help/images/Method_636.pngNumberFormat (Number As Double, MinimumIntegers As Int, MaximumFractions As Int) As String 42](#_Toc495585985)

[https://www.b4x.com/android/help/images/Method_636.pngNumberFormat2 (Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean) As String 42](#_Toc495585986)

[https://www.b4x.com/android/help/images/Method_636.pngPower (Base As Double, Exponent As Double) As Double 42](#_Toc495585987)

[https://www.b4x.com/android/help/images/Field.pngQUOTE As String 42](#_Toc495585988)

[https://www.b4x.com/android/help/images/Field.pngRegex As Regex 42](#_Toc495585989)

[https://www.b4x.com/android/help/images/Method_636.pngReturn 42](#_Toc495585990)

[https://www.b4x.com/android/help/images/Method_636.pngRnd (Min As Int, Max As Int) As Int 42](#_Toc495585991)

[https://www.b4x.com/android/help/images/Method_636.pngRndSeed (Seed As Long) 42](#_Toc495585992)

[https://www.b4x.com/android/help/images/Method_636.pngRound (Number As Double) As Long 42](#_Toc495585993)

[https://www.b4x.com/android/help/images/Method_636.pngRound2 (Number As Double, DecimalPlaces As Int) As Double 42](#_Toc495585994)

[https://www.b4x.com/android/help/images/Method_636.pngSelect 43](#_Toc495585995)

[https://www.b4x.com/android/help/images/Method_636.pngSender As Object 43](#_Toc495585996)

[Sin (Radians As Double) As Double Calculates the trigonometric sine function. Angle measured in radians. 43](#_Toc495585997)



[SinD (Degrees As Double) As Double 43](#_Toc495585998)



[Calculates the trigonometric sine function. Angle measured in degrees. 43](#_Toc495585999)

[https://www.b4x.com/android/help/images/Method_636.pngSleep (Value As Double) As Double 43](#_Toc495586000)

[https://www.b4x.com/android/help/images/Method_636.pngSqrt (Value As Double) As Double 43](#_Toc495586001)

[https://www.b4x.com/android/help/images/Method_636.pngSub 44](#_Toc495586002)

[https://www.b4x.com/android/help/images/Method_636.pngSubExists (Object As Object, Sub As String) As Boolean 44](#_Toc495586003)

[https://www.b4x.com/android/help/images/Field.pngTAB As String 44](#_Toc495586004)

[https://www.b4x.com/android/help/images/Method_636.pngTan (Radians As Double) As Double 44](#_Toc495586005)

[https://www.b4x.com/android/help/images/Method_636.pngTanD (Degrees As Double) As Double 44](#_Toc495586006)

[https://www.b4x.com/android/help/images/Field.pngTrue As Boolean 44](#_Toc495586007)

[https://www.b4x.com/android/help/images/Method_636.pngTry 44](#_Toc495586008)

[https://www.b4x.com/android/help/images/Method_636.pngType 45](#_Toc495586009)

[https://www.b4x.com/android/help/images/Method_636.pngUntil 45](#_Toc495586010)

[https://www.b4x.com/android/help/images/Method_636.pngWhile 45](#_Toc495586011)

[4.3 Conditional statements 46](#_Toc495586012)

[4.3.1 If – Then – Else 46](#_Toc495586013)

[4.3.2 Select – Case 48](#_Toc495586014)

[4.4 Loop structures 50](#_Toc495586015)

[4.4.1 For – Next 50](#_Toc495586016)

[4.4.2 For - Each 51](#_Toc495586017)

[4.4.3 Do - Loop 52](#_Toc495586018)

[4.5 Subs 54](#_Toc495586019)

[4.5.1 Declaring 54](#_Toc495586020)

[4.5.2 Calling a Sub 54](#_Toc495586021)

[4.5.3 Calling a Sub from another module 54](#_Toc495586022)

[4.5.4 Naming 55](#_Toc495586023)

[4.5.5 Parameters 55](#_Toc495586024)

[4.5.6 Returned value 55](#_Toc495586025)

[4.6 Resumable Subs 56](#_Toc495586026)

[4.6.1 Sleep 56](#_Toc495586027)

[4.6.2 Wait For 57](#_Toc495586028)

[4.6.3 Code Flow 59](#_Toc495586029)

[4.6.4 Waiting for a resumable sub to complete 60](#_Toc495586030)

[4.6.5 DoEvents 61](#_Toc495586031)

[4.6.6 Dialogs 62](#_Toc495586032)

[4.6.7 SQL with Wait For 63](#_Toc495586033)

[4.6.7.1 Queries 63](#_Toc495586034)

[4.6.7.2 B4J 64](#_Toc495586035)

[4.6.8 Notes & Tips 64](#_Toc495586036)

[4.7 Events 65](#_Toc495586037)

[4.7.1 B4A 65](#_Toc495586038)

[4.7.2 B4i 68](#_Toc495586039)

[4.7.3 B4J 70](#_Toc495586040)

[4.7.4 B4R 73](#_Toc495586041)

[4.7.5 User interface summary 74](#_Toc495586042)

[4.8 Libraries 75](#_Toc495586043)

[4.8.1 Standard libraries 76](#_Toc495586044)

[4.8.2 Additional libraries folder 77](#_Toc495586045)

[4.8.3 Load and update a Library 78](#_Toc495586046)

[4.8.4 Error message "Are you missing a library reference?" 78](#_Toc495586047)

[4.9 String manipulation 79](#_Toc495586048)

[4.9.1 B4A, B4i, B4J 79](#_Toc495586049)

[4.9.2 B4R 80](#_Toc495586050)

[4.10 Number formatting 83](#_Toc495586051)

[4.10.1 B4A, B4i, B4J 83](#_Toc495586052)

[4.10.2 B4R 83](#_Toc495586053)

[4.11 Timers 84](#_Toc495586054)

[4.12 Files B4A, B4i, B4J 86](#_Toc495586055)

[4.12.1 File object 86](#_Toc495586056)

[4.12.2 Filenames 88](#_Toc495586057)

[4.12.3 Subfolders 88](#_Toc495586058)

[4.12.4 B4A, B4J TextWriter 89](#_Toc495586059)

[4.12.5 B4A, B4J TextReader 90](#_Toc495586060)

[4.12.6 Text encoding 91](#_Toc495586061)

[4.13 Lists B4A, B4i and B4J only 93](#_Toc495586062)

[4.14 Maps B4A, B4i and B4J only 95](#_Toc495586063)

[5 Help tools 97](#_Toc495586064)

[5.1 Search function in the forum 97](#_Toc495586065)

[5.2 B4x Help Viewer 99](#_Toc495586066)

[5.3 Help documentation - B4A Object Browser 102](#_Toc495586067)

[6 Tips 103](#_Toc495586068)

[6.1 Separate data from code 103](#_Toc495586069)

[6.2 Don't Repeat Yourself (DRY principle). 103](#_Toc495586070)

[6.3 Map collection 103](#_Toc495586071)

[6.4 New technologies and features. 103](#_Toc495586072)

[6.5 Logs 103](#_Toc495586073)

[6.6 B4A Avoid calling DoEvents. 104](#_Toc495586074)

[6.7 Strings are made of characters not bytes. 104](#_Toc495586075)

[6.8 B4A Use services, especially the Starter service 104](#_Toc495586076)

[6.9 UI Layouts 104](#_Toc495586077)

[6.10 B4J as a backend solution. 104](#_Toc495586078)

[6.11 Search. 105](#_Toc495586079)

[6.12 Notepad++. 105](#_Toc495586080)

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 7.30

B4i version 4.30

B4J version 5.90

B4R version 2.20

# BASIC

**BASIC** (an acronym for **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) is a family of [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language), [high-level programming languages](https://en.wikipedia.org/wiki/High-level_programming_language) 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).

## B4x

**B4X** is a suite of rapid application development IDEs that allows the creation of applications for the following platforms: Google [Android](https://en.wikipedia.org/wiki/Android_%28operating_system%29), Apple [iOS](https://en.wikipedia.org/wiki/IOS), [Java](https://en.wikipedia.org/wiki/Java_%28programming_language%29), [Raspberry Pi](https://en.wikipedia.org/wiki/Raspberry_Pi) and [Arduino](https://en.wikipedia.org/wiki/Arduino). B4X uses a Proprietary dialect of [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) (hereinafter called "B4X") and also features a visual designer that simplifies the process of creating user interfaces.

B4R, B4A, B4J and B4i together make the best development solution for the Internet of Things (IoT).

### 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.

### 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.

### 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).

### 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.

# 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.

## 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 | 2 7 - 1 |
| -128 | 127 |
| Short | integer 16 bits | - 2 15 | 2 15 -1 |
| - 32768 | 32767 |
| Int | integer 32 bits | - 2 31 | 2 31 -1 |
| -2147483648 | 2147483647 |
| Long | long integer 64 bits | - 2 63 | 2 63 -1 |
| -9223372036854775808 | 9223372036854775807 |
| Float | floating point number  32 bits | - 2 -149 | (2 -2 -23) \* 2 127 |
| 1.4E-45 | 3.4028235 E 38 |
| Double | double precision number  64 bits | - 2 -1074 | (2 -2 -52) \* 2 1023 |
| 2.2250738585072014 E -308 | 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 are(3) As Int

Private List1 As List

List1.Initialize

For I = 1 To 5

are(0) = I \* 2

are(1) = I \* 2

are(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

## 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

## Declaring variables

### 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](#_Scope).

There exist the Dim keyword, this is maintained for compatibility.

Examples:

Private Capital As Double Declares three variables as Double,

Private Interest As Double double precision numbers.

Private Rate As Double

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

Private j As Int

Private k As Int

Private lblCapital As Label

Private lblInterest As Label Declares three variables as Label views.

Private lblRate As Label

Private btnNext As Button Declares two variables as Button views.

Private btnPrev As Button

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.

### 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](#_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")

### 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

### 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)

## 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

## Scope

### 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 !

### 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.

### 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.

## 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 MyPanel.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.

# Program flow / Process life cycle

## B4A

Let's start simple:

Each B4A program runs in its own process.

A process has one main thread which is also named the UI thread which lives as long as the process lives. A process can also have more threads which are useful for background tasks.

A process starts when the user launches your application, assuming that it is not running already in the background.

The process end is less determinant. It will happen sometime after the user or system has closed all the activities.

If for example you have one activity and the user pressed on the back key, the activity gets closed. Later when the phone gets low on memory (and eventually it will happen) the process will quit.

If the user launches your program again and the process was not killed then the same process will be reused.

A B4A application is made of one or more activities.

**Activities are somewhat similar to Windows Forms.**

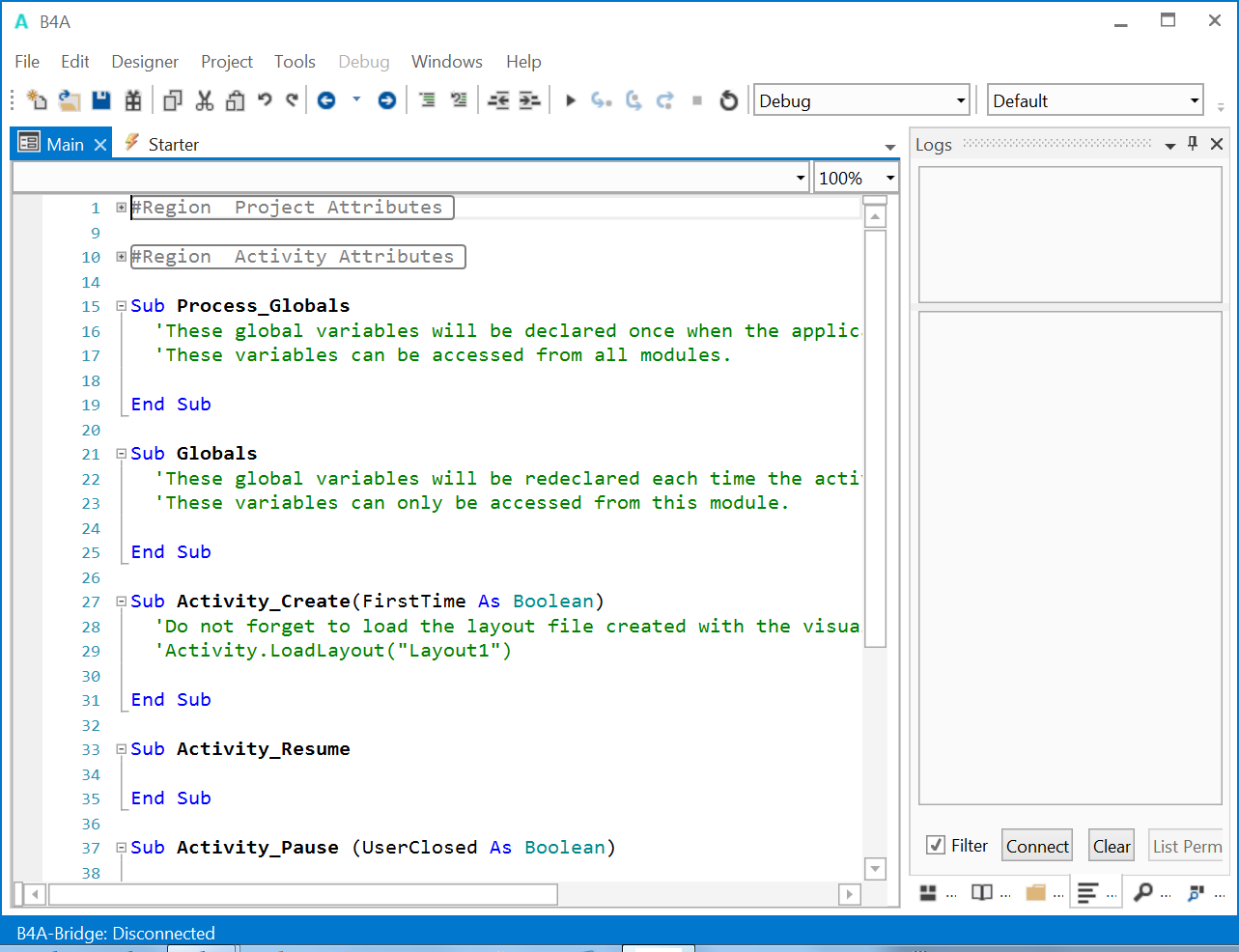
One major difference is that, while an activity is not in the foreground it can be killed in order to preserve memory. Usually you will want to save the state of the activity before it gets lost. Either in a persistent storage or in memory that is associated with the process.

Later this activity will be recreated when needed.

Another delicate point happens when there is a major configuration change in the device. The most common is an orientation change (the user rotates the device). When such a change occurs the current activities are destroyed and then recreated. Now it is possible to create the activity according to the new configuration (for example, we now know the new screen dimensions).

### Program Start

When we start a new program we get following template:



On the top left we see two module Tabs :

Main Activity

[Starter Service](#_Starter_service)

The Starter Service is used to declare all ProcessGlobal variables and these variables are accessible from any module in the project.

The Main Activity is the starting activity, it cannot be removed.

Variables can be either global or local. Local variables are variables that are declared inside a sub other than Process\_Globals or Globals.

Local variables are local to the containing sub or module. Once the sub ends, these variables no longer exist.

Global variables can be accessed from all subs in the containing module.

There are two types of global variables.

Process variables (accessible from all modules) and activity variables (accessible from a single module).

### Process global variables

These variables live as long as the process lives.

You should declare these variables as Public inside Sub Process\_Globals of the Starter Service like.

Sub **Process\_Globals**

'These global variables will be declared once when the application starts.

'These variables can be accessed from all modules.

Public MyVariable = "Test" As String

This sub is called once when the process starts.

These variables are the only "public" variables. Which means that they can be accessed from other modules as well.

There is also a Process\_Globals routines in each Activity module.

If you need variables, valid only in the Activity, which are initialized only once when the program is lauched you should put them in the Activity’s Process\_Globals routine (this is true for all activities, not just the first activity).

However, not all types of objects can be declared as process variables.

All of the views for example 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, when the activity is destroyed, all of the views that are contained in the activity are destroyed as well. If we didn't do this, and kept a reference to a view after the Activity was destroyed, the garbage collector would not be able to free the resource and we would have a memory leak.

The compiler enforces this requirement.

### Activity variables

These variables are owned 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.

### Starter service

One of the challenges that developers of any non-small Android app need to deal with, is the multiple possible entry points.  
  
During development in almost all cases the application will start from the Main activity.  
Many programs start with code similar to:

Sub **Activity\_Create** (FirstTime As Boolean)

If FirstTime Then

SQL.Initialize(...)

SomeBitmap = LoadBitmap(...)

'additional code that loads application-wide resources

End If

End Sub

Everything seems to work fine during development. However the app "strangely" crashes from time to time on the end user device.  
The reason for those crashes is that the OS can start the process from a different activity or service. For example if you use StartServiceAt and the OS kills the process while it is in the background.  
Now the SQL object and the other resources will not be initialized.  
  
Starting from B4A v5.20 there is a new feature named Starter service that provides a single and consistent entry point. If the Starter service exists then the process will always start from this service.  
  
The Starter service will be created and started, and only then, the activity or service that were supposed to be started will start.  
This means that the Starter service is the best place to initialize all the application-wide resources.  
Other modules can safely access these resources.  
The Starter service should be the default location for all the public process global variables. SQL objects, data read from files and bitmaps used by multiple activities should all be initialized in the Service\_Create sub of the Starter service.  
  
Notes

* The Starter service is identified by its name. You can add a new service named Starter to an existing project and it will be the program entry point.  
  This is done by selecting Project > Add New Module > Service Module.
* This is an optional feature. You can remove the Starter service.
* You can call StopService(Me) in Service\_Start if you don't want the service to keep on running. However this means that the service will not be able to handle events (for example you will not be able to use the asynchronous SQL methods).
* The starter service should be excluded from compiled libraries. Its #ExcludeFromLibrary attribute is set to True by default in the Service Attributes region.

### Program flow

The program flow is the following:

* **Main Process\_Globals** Process\_Globals routines of the Main modules   
  Here we declare all Private variables and objects for the Main module.
* **Starter Sevice Process\_Globals** If the service exists, it is run.  
  Here we declare all Public Process Global variables and objects like SQL, Bitmaps etc.
* **Other Activity Main Process\_Globals** Process\_Globals routines of other modules   
  Here we declare all Private variables and objects for the given module.
* **Starter Service Service\_Create** If the service exists, it is run.  
  Here we initialize all Public Process Global variables and objects like SQL, Bitmaps etc.
* **Starter Sevice Service\_Start** If the service exists, it is run.  
  We can leave this routine empty.
* [Globals](#_Globals_versus_FirstTime)  
  Here we declare all Private variables for the given Activity.
* [Sub Activity\_Create](#_Activity_Create_(FirstTime_As)  
  Here we load layouts and initialize activity objects added by code
* [Activity\_Resume](#_Sub_Activity_Resume_Sub)This routine is run every time the activity changes its state.
* [Activity\_Pause](#_Sub_Activity_Resume_Sub)  
  This routine is run when the Activity is paused, like orientation change, lauch of another activity etc.

### Sub Process\_Globals / Sub Globals

In any Activity, Process\_Globals and Globals should be used to declare variables.

You can also set the values of "simple" variables (numeric, strings and booleans).  
  
You should not put any other code there.  
You should instead put the code in Activity\_Create.

### Sub Activity\_Create (FirstTime As Boolean)

This sub is called when the activity is created.

The activity is created

* when the user first launches the application
* the device configuration has changed (user rotated the device) and the activity was destroyed
* when the activity was in the background and the OS decided to destroy it in order to free memory.

The primary purpose of this sub is to load or create the layout (among other uses).

The FirstTime parameter tells us if this is the first time that this activity is created. First time relates to the current process.

You can use FirstTime to run all kinds of initializations related to the process variables.

For example if you have a file with a list of values that you need to read, you can read it if FirstTime is True and store the list as a process variable by declaring the list in Sub Process\_Globals

Now we know that this list will be available as long as the process lives and there is no need to reload it even when the activity is recreated.

To summarize, you can test whether FirstTime is True and then initialize the process variables that are declared in the Activity’s Sub Process\_Globals.

### Variable declaration summary

Which variable should we declare where and where do we initialize our variables:

* Variables and none user interface objects you want to access from several modules.  
  Like SQL, Maps, Lists, Bitmaps etc.  
  These must be declared as Public in Starter Process\_Globals like:  
    
  Sub **Process\_Globals** Public SQL1 As SQL  
   Public Origin = 0 As Int  
   Public MyBitmap As Bitmap  
  End Sub  
    
  And initialized in Starter Service\_Create like:  
    
  Sub **Service\_Create** SQL1.Initialize(...)  
   MyBitmap.Initialize(...)  
  End Sub
* Variables accessible from all Subs in an Activity which should be initialized only once.  
  These must be declared as Private in Activity Process\_Globals like:  
    
  Sub **Process\_Globals** Private MyList As List  
   Private MyMap As Map  
  End Sub  
    
  And initialized in Activty\_Create like:  
    
  Sub **Activity\_Create** MyList.Initialize  
   MyMap.Initialize  
  End Sub
* Variables in a Class or Code module  
  These are mostly declared as Private, you can declare them as Public if you want them being accessible from outsides the Class or Code module.   
  Class modules are explained in detail in the *B4x Booklet CustomViews*.
* User interface objects  
  These must be declared in the Activity module where they are used in Globals like:  
    
  Sub **Globals** Private btnGoToAct2, btnChangeValues As Button  
   Private lblCapital, lblInterest, lblRate As Label  
  End Sub

Simple variables like Int, Double String and Boolean can be initialized directly in the declaration line, even in Process\_Globals routines.

Example:

Public Origin = 0 as Int

**No code should be written in Process\_Globals routines !**

### Sub Activity\_Resume Sub Activity\_Pause (UserClosed As Boolean)

Activity\_Resume is called right after Activity\_Create finishes or after resuming a paused activity (activity moved to the background and now it returns to the foreground).

Note that when you open a different activity (by calling StartActivity), the current activity is first paused and then the other activity will be created if needed and (always) resumed.

Each time the activity moves from the foreground to the background Activity\_Pause is called.

Activity\_Pause is also called when the activity is in the foreground and a configuration change occurs (which leads to the activity getting paused and then destroyed).

Activity\_Pause is the last place to save important information.

Generally there are two types of mechanisms that allow you to save the activity state.

Information that is only relevant to the current application instance can be stored in one or more process variables.

Other information should be stored in a persistent storage (file or database).

For example, if the user changed some settings you should save the changes to a persistent storage at this point. Otherwise the changes may be lost.

Activity\_Pause is called every time the activity moves from the foreground to the background. This can happen because:

1. A different activity was started.
2. The Home button was pressed.
3. A configuration changed event was raised (orientation changed for example).
4. The Back button was pressed.

In scenarios 1 and 2, the activity will be paused and for now kept in memory as it is expected to be reused later.

In scenario 3 the activity will be paused, destroyed and then created (and resumed) again.

In scenario 4 the activity will be paused and destroyed. **Pressing on the Back button is similar to closing the activity**. In this case you do **not** need to save any instance specific information (the position of pacman in a PacMan game for example).

The UserClosed parameter will be true in this scenario and false in all other. Note that it will also be true when you call Activity.Finish. This method pauses and destroys the current activity, similar to the Back button.

You can use UserClosed parameter to decide which data to save and also whether to reset any related process variables to their initial state (move pacman position to the center if the position is a process variable).

### Activity.Finish / ExitApplication

Some explanations on how and when to use Activity.Finish and ExitApplication.

An interesting article about the functioning of Android can be found here: [Multitasking the Android way](http://android-developers.blogspot.com/2010/04/multitasking-android-way.html).

**Most applications should not use ExitApplication but prefer Activity.Finish which lets the OS decide when the process is killed.**

**You should use it only if you really need to fully kill the process.**

When should we use Activity.Finish and when not ?

Let us consider following example without any Activity.Finish:

* **Main activity**
  + StartActivity(SecondActivity)
* **SecondActivity activity**
  + StartActivity(ThirdActivity)
* **ThirdActivity activity**
  + Click on Back button
  + The OS goes back to previous activity, SecondActivity
* **SecondActivity activity**
  + Click on Back button
  + The OS goes back to previous activity, Main
* **Main activity**
  + Click on Back button
  + The OS leaves the program

Let us now consider following example with Activity.Finish before each StartActivity:

* **Main activity**
  + Activity.Finish
  + StartActivity(SecondActivity)
* **SecondActivity activity**
  + Activity.Finish
  + StartActivity(ThirdActivity)
* **ThirdActivity activity**
  + Click on Back button
  + The OS leaves the program

We should use Activity.Finish before starting another activity only if we don't want to go back to this activity with the Back button.

## Program flow B4i

The program flow in B4i is much more simple than the B4A program flow.

When we run a new project we get the template below:

Sub **Process\_Globals**

'These global variables will be declared once when the application starts.

'Public variables can be accessed from all modules.

Public App As Application

Public NavControl As NavigationController

Private Page1 As Page

End Sub

Private Sub **Application\_Start** (Nav As NavigationController)

'SetDebugAutoFlushLogs(True) 'Uncomment if program crashes before all logs are printed.

NavControl = Nav

Page1.Initialize("Page1")

Page1.Title = "Page 1"

Page1.RootPanel.Color = Colors.White

NavControl.ShowPage(Page1)

End Sub

Private Sub **Page1\_Resize**(Width As Int, Height As Int)

End Sub

Private Sub **Application\_Background**

End Sub

When you start the program, the routines are executed in the order above.

Be aware that the dimensions of Page1 are not known in Application\_Start, the are only known in the Page1\_Resize routine in the Width and Height parameters.

If you want to adjust views you must do it here.

## Program flow B4J

The program flow in B4i is much more simple than the B4A program flow, similar to B4i.

When we run a new project we get the template below:

Sub **Process\_Globals**

Private fx As JFX

Private MainForm As Form

End Sub

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

MainForm = Form1

'MainForm.RootPane.LoadLayout("Layout1") 'Load the layout file.

MainForm.Show

End Sub

'Return true to allow the default exceptions handler to handle the uncaught exception.

Sub **Application\_Error** (Error As Exception, StackTrace As String) As Boolean

Return True

End Sub

When you start the program, the routines are executed in the order above.

If you want to adjust Nodes when the user resizes a form you must add a Resize routine for this form, like:

Private Sub **MainForm\_Resize** (Width As Double, Height As Double)

' Your code

End Sub

## Program flow B4R

The program flow in B4R is straight forward.

When we run a new project we find this code template:

Sub **Process\_Globals**

'These global variables will be declared once when the application starts.

'Public variables can be accessed from all modules.

Public Serial1 As Serial

End Sub

Private Sub **AppStart**

Serial1.Initialize(115200)

Log("AppStart")

End Sub

When you run the program, Process\_Globals and then AppStart are executed.

Serial1.Initialize(115200) Initializes the bit rate.

Log("AppStart") Writes “AppStart” in the Logs.

## Program flow comparison B4A / B4i / B4J

### Program start B4A / B4i / B4J

**B4A B4i B4J**

Main Process\_Globals Main Process\_Globals Main Process\_Globals

Starter Process\_Globals

Other modues Process\_Globals Other modues Process\_Globals Other modues Process\_Globals

Starter Service\_Create Main Application\_Start Main AppStart

Starter Service\_Start Main Page1\_Resize Main MainForm\_Resize

Main Globals

Main Activity\_Create

FirstTime = True

Main Activity\_Resume

### Rotating device B4A / B4i

**B4A B4i**

Main Activity\_Pause

Main Globals Main Page1\_Resize

Main Activity\_Create

FirstTime = False

Main Activity\_Resume

# Basic language

## Expressions

An [expression](http://en.wikipedia.org/wiki/Expression_(programming)) 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).

### 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) xy | 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 > 92 \* 5 > 81 \* 5

Power(4+5,2)\*(3+2)

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

233 Power(23,3) > 23 at the power of 3

- 22 = - 4

(-2)2 = 4

### 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 |

### 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 |

## Standard keywords

Not all keywords are available in B4R.

https://www.b4x.com/android/help/images/Method_636.png  [**Abs**](#keywords_abs) (Number As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ACos**](#keywords_acos) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ACosD**](#keywords_acosd) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Array**](#keywords_array)  
https://www.b4x.com/android/help/images/Method_636.png  [**Asc**](#keywords_asc) (Char As Char) As Int  
https://www.b4x.com/android/help/images/Method_636.png  [**ASin**](#keywords_asin) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ASinD**](#keywords_asind) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ATan**](#keywords_atan) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ATan2**](#keywords_atan2) (Y As Double, X As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ATan2D**](#keywords_atan2d) (Y As Double, X As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**ATanD**](#keywords_atand) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**BytesToString**](#keywords_bytestostring) (Data() As Byte, StartOffset As Int, Length As Int, CharSet As String) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**CallSub**](#keywords_callsub) (Component As Object, Sub As String) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**CallSub2**](#keywords_callsub2) (Component As Object, Sub As String, Argument As Object) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**CallSub3**](#keywords_callsub3) (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**Catch**](#keywords_catch)  
https://www.b4x.com/android/help/images/Field.png  [**cE**](#keywords_ce) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Ceil**](#keywords_ceil) (Number As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**CharsToString**](#keywords_charstostring) (Chars() As Char, StartOffset As Int, Length As Int) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**Chr**](#keywords_chr) (UnicodeValue As Int) As Char  
https://www.b4x.com/android/help/images/Method_636.png  [**Continue**](#keywords_continue)  
https://www.b4x.com/android/help/images/Method_636.png  [**Cos**](#keywords_cos) (Radians As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**CosD**](#keywords_cosd) (Degrees As Double) As Double  
https://www.b4x.com/android/help/images/Field.png  [**cPI**](#keywords_cpi) As Double  
https://www.b4x.com/android/help/images/Field.png  [**CRLF**](#keywords_crlf) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**Dim**](#keywords_dim)  
https://www.b4x.com/android/help/images/Method_636.png  [**Exit**](#keywords_exit)  
https://www.b4x.com/android/help/images/Field.png  [**False**](#keywords_false) As Boolean  
https://www.b4x.com/android/help/images/Method_636.png  [**Floor**](#keywords_floor) (Number As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**For**](#keywords_for)  
https://www.b4x.com/android/help/images/Method_636.png  [**GetType**](#keywords_gettype) (object As Object) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**If**](#keywords_if)  
https://www.b4x.com/android/help/images/Method_636.png  [**Is**](#keywords_is)  
https://www.b4x.com/android/help/images/Method_636.png  [**IsNumber**](#keywords_isnumber) (Text As String) As Boolean  
https://www.b4x.com/android/help/images/Method_636.png  [**LoadBitmap**](#keywords_loadbitmap) (Dir As String, FileName As String) As Bitmap  
https://www.b4x.com/android/help/images/Method_636.png  [**LoadBitmapSample**](https://www.b4x.com/android/help/core.html#keywords_loadbitmapsample) (Dir As String, FileName As String, MaxWidth As Int, MaxHeight As Int) As Bitmap  
https://www.b4x.com/android/help/images/Method_636.png  [**Log**](#keywords_log) (Message As String)  
https://www.b4x.com/android/help/images/Method_636.png  [**Logarithm**](#keywords_logarithm) (Number As Double, Base As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**LogColor**](https://www.b4x.com/android/help/core.html#keywords_logcolor) (Message As String, Color As Int)  
https://www.b4x.com/android/help/images/Method_636.png  [**Max**](#keywords_max) (Number1 As Double, Number2 As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Me**](#keywords_me) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**Min**](#keywords_min) (Number1 As Double, Number2 As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Not**](#keywords_not) (Value As Boolean) As Boolean

[**Null**](#keywords_null) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**NumberFormat**](#keywords_numberformat) (Number As Double, MinimumIntegers As Int, MaximumFractions As Int) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**NumberFormat2**](#keywords_numberformat2) (Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**Power**](#keywords_power) (Base As Double, Exponent As Double) As Double  
https://www.b4x.com/android/help/images/Field.png  [**QUOTE**](#keywords_quote) As String  
https://www.b4x.com/android/help/images/Field.png  [**Regex**](#keywords_regex) As Regex  
https://www.b4x.com/android/help/images/Method_636.png  [**Return**](#keywords_return)  
https://www.b4x.com/android/help/images/Method_636.png  [**Rnd**](#keywords_rnd) (Min As Int, Max As Int) As Int  
https://www.b4x.com/android/help/images/Method_636.png  [**RndSeed**](#keywords_rndseed) (Seed As Long)  
https://www.b4x.com/android/help/images/Method_636.png  [**Round**](#keywords_round) (Number As Double) As Long  
https://www.b4x.com/android/help/images/Method_636.png  [**Round2**](#keywords_round2) (Number As Double, DecimalPlaces As Int) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Select**](#keywords_select)  
https://www.b4x.com/android/help/images/Method_636.png  [**Sender**](#keywords_sender) As Object  
https://www.b4x.com/android/help/images/Method_636.png  [**Sin**](#keywords_sin) (Radians As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**SinD**](#keywords_sind) (Degrees As Double) As Double



https://www.b4x.com/android/help/images/Method_636.png  [**Sleep**](#keywords_sleep) (Milliseconds As Int)  
https://www.b4x.com/android/help/images/Method_636.png  [**SmartStringFormatter**](#keywords_smartstringformatter) (Format As String, Value As Object) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**Sqrt**](#keywords_sqrt) (Value As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**Sub**](#keywords_sub)  
https://www.b4x.com/android/help/images/Method_636.png  [**SubExists**](#keywords_subexists) (Object As Object, Sub As String) As Boolean  
https://www.b4x.com/android/help/images/Field.png  [**TAB**](#keywords_tab) As String  
https://www.b4x.com/android/help/images/Method_636.png  [**Tan**](#keywords_tan) (Radians As Double) As Double  
https://www.b4x.com/android/help/images/Method_636.png  [**TanD**](#keywords_tand) (Degrees As Double) As Double  
https://www.b4x.com/android/help/images/Field.png  [**True**](#keywords_true) As Boolean  
https://www.b4x.com/android/help/images/Method_636.png  [**Try**](#keywords_try)  
https://www.b4x.com/android/help/images/Method_636.png  [**Type**](#keywords_type)  
https://www.b4x.com/android/help/images/Method_636.png  [**Until**](#keywords_until)  
https://www.b4x.com/android/help/images/Method_636.png  [**While**](#keywords_while)

**https://www.b4x.com/android/help/images/Method_636.pngAbs (Number As Double) As Double**

Returns the absolute value.

**https://www.b4x.com/android/help/images/Method_636.pngACos (Value As Double) As Double**

Calculates the trigonometric arccosine function. Returns the angle measured with radians.

**https://www.b4x.com/android/help/images/Method_636.pngACosD (Value As Double) As Double**

Calculates the trigonometric arccosine function. Returns the angle measured with degrees.

**https://www.b4x.com/android/help/images/Method_636.pngArray**

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", ...)

**https://www.b4x.com/android/help/images/Method_636.pngAsc (Char As Char) As Int**

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

**https://www.b4x.com/android/help/images/Method_636.pngASin (Value As Double) As Double**

Calculates the trigonometric arcsine function. Returns the angle measured with radians.

**https://www.b4x.com/android/help/images/Method_636.pngASinD (Value As Double) As Double**

Calculates the trigonometric arcsine function. Returns the angle measured with degrees.

**https://www.b4x.com/android/help/images/Method_636.pngATan (Value As Double) As Double**

Calculates the trigonometric arctangent function. Returns the angle measured with radians.

**https://www.b4x.com/android/help/images/Method_636.pngATan2 (Y As Double, X As Double) As Double**

Calculates the trigonometric arctangent function. Returns the angle measured with radians.

**https://www.b4x.com/android/help/images/Method_636.pngATan2D (Y As Double, X As Double) As Double**

Calculates the trigonometric arctangent function. Returns the angle measured with degrees.

**https://www.b4x.com/android/help/images/Method_636.pngATanD (Value As Double) As Double**

Calculates the trigonometric arctangent function. Returns the angle measured with degrees.

**https://www.b4x.com/android/help/images/Method_636.pngBytesToString (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")

**https://www.b4x.com/android/help/images/Method_636.pngCallSub (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")

**https://www.b4x.com/android/help/images/Method_636.pngCallSub2 (Component As Object, Sub As String, Argument As Object) As Object**

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

**https://www.b4x.com/android/help/images/Method_636.pngCallSub3 (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object) As Object**

**https://www.b4x.com/android/help/images/Method_636.pngCatch**

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

**https://www.b4x.com/android/help/images/Field.pngcE As Double**

e (natural logarithm base) constant.

**https://www.b4x.com/android/help/images/Method_636.pngCeil (Number As Double) As Double**

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

**https://www.b4x.com/android/help/images/Method_636.pngCharsToString (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.

**https://www.b4x.com/android/help/images/Method_636.pngChr (UnicodeValue As Int) As Char**

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

**https://www.b4x.com/android/help/images/Method_636.pngContinue**

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

**https://www.b4x.com/android/help/images/Method_636.pngCos (Radians As Double) As Double**

Calculates the trigonometric cosine function. Angle measured in radians.

**https://www.b4x.com/android/help/images/Method_636.pngCosD (Degrees As Double) As Double**

Calculates the trigonometric cosine function. Angle measured in degrees.

**https://www.b4x.com/android/help/images/Field.pngcPI As Double**

PI constant.

**https://www.b4x.com/android/help/images/Field.pngCRLF As String**

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

**https://www.b4x.com/android/help/images/Method_636.pngDim**

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.

**https://www.b4x.com/android/help/images/Method_636.pngExit**

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

**https://www.b4x.com/android/help/images/Field.pngFalse As Boolean**

**https://www.b4x.com/android/help/images/Method_636.pngFloor (Number As Double) As Double**

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

**https://www.b4x.com/android/help/images/Method_636.pngFor**

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.

**https://www.b4x.com/android/help/images/Method_636.pngGetType (object As Object) As String**

Returns a string representing the object's java type.

**https://www.b4x.com/android/help/images/Method_636.pngIf**

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

**https://www.b4x.com/android/help/images/Method_636.pngIsNumber (Text As String) As Boolean**

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

**https://www.b4x.com/android/help/images/Method_636.pngLoadBitmap (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"))

**https://www.b4x.com/android/help/images/Method_636.pngLoadBitmapSample (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))

**https://www.b4x.com/android/help/images/Method_636.pngLog (Message As String)**

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

**https://www.b4x.com/android/help/images/Method_636.pngLogarithm (Number As Double, Base As Double) As Double**

**https://www.b4x.com/android/help/images/Method_636.pngLogColor (Message As String, Color As Int)**

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

**https://www.b4x.com/android/help/images/Method_636.pngMax (Number1 As Double, Number2 As Double) As Double**

Returns the larger number between the two numbers.

**https://www.b4x.com/android/help/images/Method_636.pngMe 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.

**https://www.b4x.com/android/help/images/Method_636.pngMin (Number1 As Double, Number2 As Double) As Double**

Returns the smaller number between the two numbers.

**https://www.b4x.com/android/help/images/Method_636.pngNot (Value As Boolean) As Boolean**

Inverts the value of the given boolean.

**https://www.b4x.com/android/help/images/Field.pngNull As Object**

**https://www.b4x.com/android/help/images/Method_636.pngNumberFormat (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"

**https://www.b4x.com/android/help/images/Method_636.pngNumberFormat2 (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"

**https://www.b4x.com/android/help/images/Method_636.pngPower (Base As Double, Exponent As Double) As Double**

Returns the Base value raised to the Exponent power.

**https://www.b4x.com/android/help/images/Field.pngQUOTE As String**

Quote character ". The value of Chr(34).

**https://www.b4x.com/android/help/images/Field.pngRegex As Regex**

Regular expressions related methods.

**https://www.b4x.com/android/help/images/Method_636.pngReturn**

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

**https://www.b4x.com/android/help/images/Method_636.pngRnd (Min As Int, Max As Int) As Int**

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

**https://www.b4x.com/android/help/images/Method_636.pngRndSeed (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.

**https://www.b4x.com/android/help/images/Method_636.pngRound (Number As Double) As Long**

Returns the closest long number to the given number.

**https://www.b4x.com/android/help/images/Method_636.pngRound2 (Number As Double, DecimalPlaces As Int) As Double**

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

**https://www.b4x.com/android/help/images/Method_636.pngSelect**

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

**https://www.b4x.com/android/help/images/Method_636.pngSender 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.

**https://www.b4x.com/android/help/images/Method_636.pngSleep (Value As Double) As Double**

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

**https://www.b4x.com/android/help/images/Method_636.pngSmartStringFormatter (Format As String, Value As Object) As String**

Internal keyword used by the Smart String literal.

**https://www.b4x.com/android/help/images/Method_636.pngSqrt (Value As Double) As Double**

Returns the positive square root.

**https://www.b4x.com/android/help/images/Method_636.pngSub**

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).

**https://www.b4x.com/android/help/images/Method_636.pngSubExists (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.

**https://www.b4x.com/android/help/images/Field.pngTAB As String**

Tab character.

**https://www.b4x.com/android/help/images/Method_636.pngTan (Radians As Double) As Double**

Calculates the trigonometric tangent function. Angle measured in radians.

**https://www.b4x.com/android/help/images/Method_636.pngTanD (Degrees As Double) As Double**

Calculates the trigonometric tangent function. Angle measured in degrees.

**https://www.b4x.com/android/help/images/Field.pngTrue As Boolean**

**https://www.b4x.com/android/help/images/Method_636.pngTry**

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

**https://www.b4x.com/android/help/images/Method_636.pngType**

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

**https://www.b4x.com/android/help/images/Method_636.pngUntil**

Loops until the condition is true.  
Syntax:  
Do Until condition  
 ...  
Loop

**https://www.b4x.com/android/help/images/Method_636.pngWhile**

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

## Conditional statements

Different conditional statements are available in Basic.

### If – Then – Else

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 The simplest **If-Then** structure.

End If

If b = 0 Then a = 0 The same but in one line.

If b = 0 Then

a = 0 The simplest **If-Then-Else** structure.

Else

a = 1

End If

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 VB

If b = 0 Then If b = 0 Then

a = 0 a = 0

End If c = 1

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.

### 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 **TestExpression** is the expression to test.

Case ExpressionList1

' code1 **ExpressionList1** is a list of expressions to compare

Case ExpressionList2 to **TestExpression**

' code2 **ExpressionList2** is another list of expressions to compare

Case Else to **TestExpression**

' code3

End Select

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

Case 12, 24

Select Txt.CharAt The **TestExpression**  is a character at

Case "A", "B", "C"

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 VB

Select Value Select Case Value

Case 1,2,3,4,8,9,10 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.

## Loop structures

Different loop structures are available in Basic.

### 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 i incremental variable

n1 initial value

' Specific code n2 final value

n3 step

Next

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 For i = 0 To 10 Step 1

is the same as

Next 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 In this example, if the variable a equals 0

' code

If A = 0 Then Exit Then exit the loop.

' code

Next

**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.

### For - Each

It is a variant of the For - Next loop.

Example:

For Each n As Type In Array n variable any type or object

Type type of variable n

' Specific code Array Array of values or objects

Next

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

### Do - Loop

Several configurations exist:

Do While test **test**  is any expression

' code Executes the **code** while **test** is **True**

Loop

Do Until test **test**  is any expression

' code Executes the **code** until **test** is **True**

Loop

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

' code

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

## 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.

### 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.

### 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.

### 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)

### 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.

### 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.

### 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

## 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](https://en.wikipedia.org/wiki/Coroutine).)

You find more examples in the [forum](https://www.b4x.com/android/forum/threads/b4x-resumable-subs-sleep-wait-for.78601/#content).

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

### 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

### 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.

### 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.

### 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.

### 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](https://www.b4x.com/android/forum/threads/79532/#content) 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](https://www.b4x.com/android/forum/threads/78601/#content) 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.

### 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](https://www.b4x.com/android/forum/threads/78601/#content) 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.

### 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.

#### 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.

#### 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](https://www.b4x.com/android/forum/threads/webapp-concurrent-access-to-sqlite-databases.39904/#content)

### 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.

## 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.

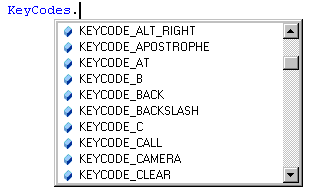
### B4A

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

### 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

### 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 | SelectedIndexChangedd | 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 **TextField1\_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 includes in the MouseEvent object:
* **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.

### B4R

In B4R, the Pin and [Timer](#_Timers) 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

### 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 |  |  |  |

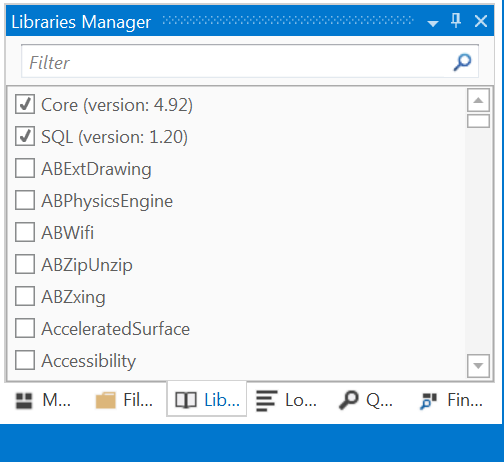
## Libraries

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](https://www.b4x.com/android/documentation.html), [B4i](https://www.b4x.com/b4i/documentation.html), [B4J](https://www.b4x.com/b4j/documentation.html), [B4R](https://www.b4x.com/b4r/documentation.html)  
  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.  
    
  

### 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

### Additional libraries folder

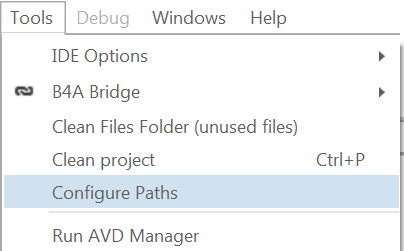
For the additional libraries it is useful to setup a special folder to save them somewhere else.

For example:

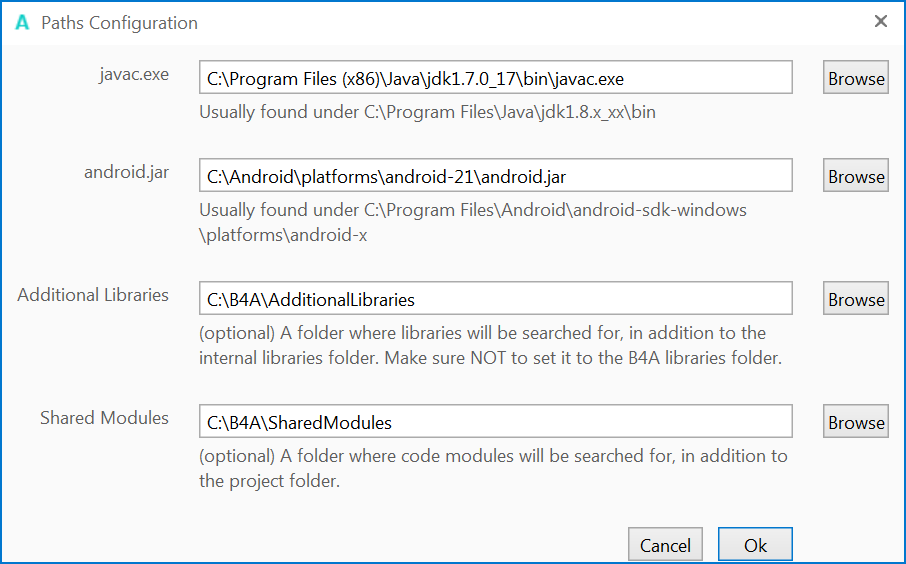
C:\B4A\AddLibraries, C:\B4i\AddLibraries, C:\B4J\AddLibraries, C:\B4R\AddLibraries

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 .

### Load and update a Library

A list of the official and additional libraries with links to the relevant help documentation can be found on the B4x site in the:

B4A Documentation page: [List of Libraries](https://www.b4x.com/android/documentation.html).

B4i Documentation page: [List of Libraries](https://www.b4x.com/android/documentation.html).

B4J Documentation page: [List of Libraries](https://www.b4x.com/android/documentation.html).

B4RA Documentation page: [List of Libraries](https://www.b4x.com/android/documentation.html).

To find the library files use a query like <http://www.b4x.com/search?query=betterdialogs+library>

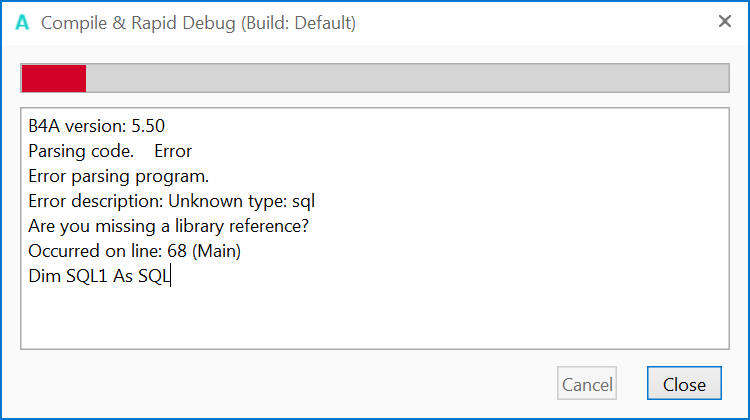
in your internet browser.

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](#_Additional_libraries_folder) for an additional library.
* Right click in the libraries list in the [Lib Tab](#_Libs) and click on  and select the library.  
    
  

### 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 !



## String manipulation

### 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")

### 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, "Number 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 Source 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 Source 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

### 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  
  NumberFormat2(12345.67, 0, 3, 3, False) = 12345.670

### 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.

## 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.  
Timer events will not fire when the activity is paused, or if a blocking dialog (like Msgbox) is visible.  
It is also important, in B4A, 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\_Startroutine

Private Sub **Application\_Start** (Nav As NavigationController)

Timer1.Initialize("Timer1", 1000)

**B4J:** AppStart routine

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

Timer1.Initialize("Timer1", 1000)

**B4R:** AppStart 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.

Private Sub **Timer1\_Tick**

' Do something

End Sub

## 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. B4A nor B4i programs have 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.

### 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](http://en.wikipedia.org/wiki/.properties)  
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.

### 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

### 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.

### B4A, B4J 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) for 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](#_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

### B4A, B4J 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.OpenInput(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](#_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

### 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.

B4x 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 B4x is the end of line character:

* B4x, 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)

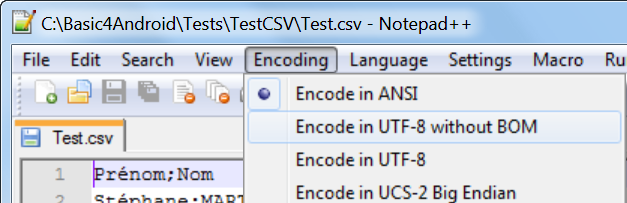
When you save a file with NotePad three additional bytes are added .

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")

## 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

## 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, clears all items from the map :

* Clear  
  Map1.Clear

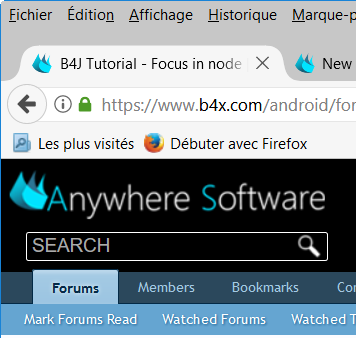
Maps can be saved and loaded with :

* 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)

# Help tools

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

## Search function in the forum

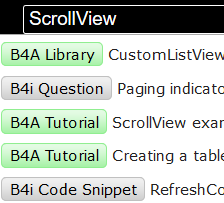


In the upper left corner you find the searchbox for the forum. Depending on the window size it can be on the top right side.

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

The function shows you the posts that match your request.

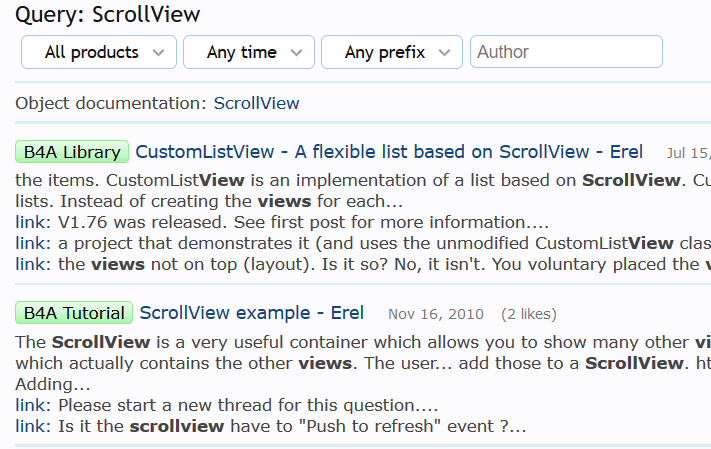
 Example: Enter the keyword ScrollView.



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

Click on an item to show the thread.

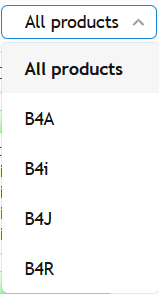
And the result:



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

Click on the title to show the selected post.

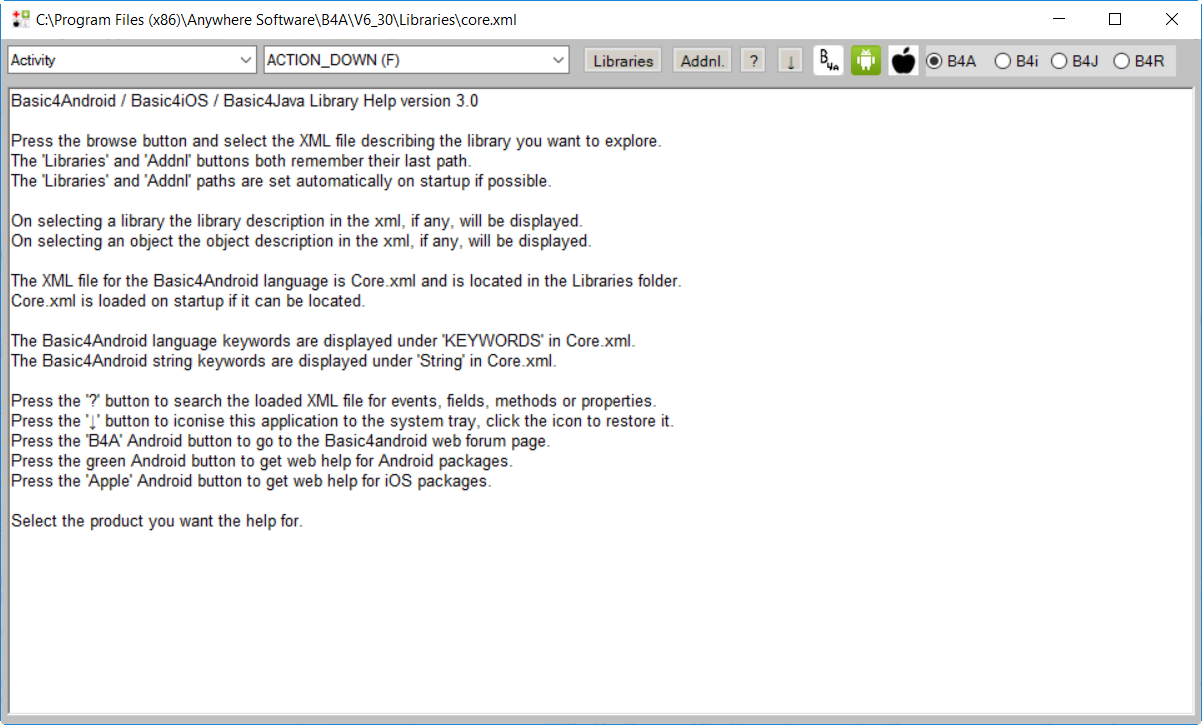
Example of filtering on a product:



## B4x Help Viewer

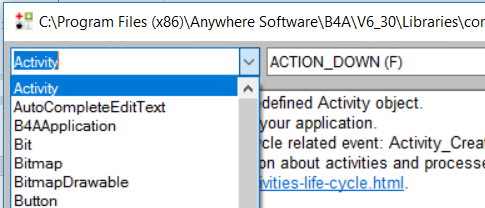
This program shows xml help files. It was originally written by Andrew Graham (agrham) for B4A. I modified it, with Andrews’ agreement, to show B4A, B4J, B4i and B4R xml help files.

The program can be [downloaded](http://www.basic4ppc.com/android/forum/threads/b4i-help-viewer.46969/) from the forum.

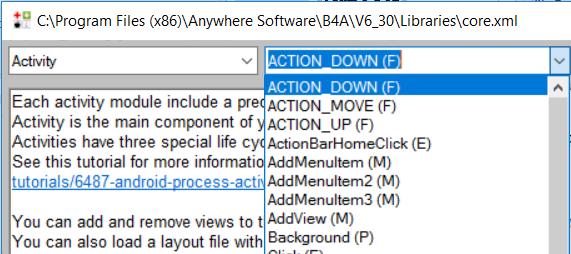


On top we find:





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.

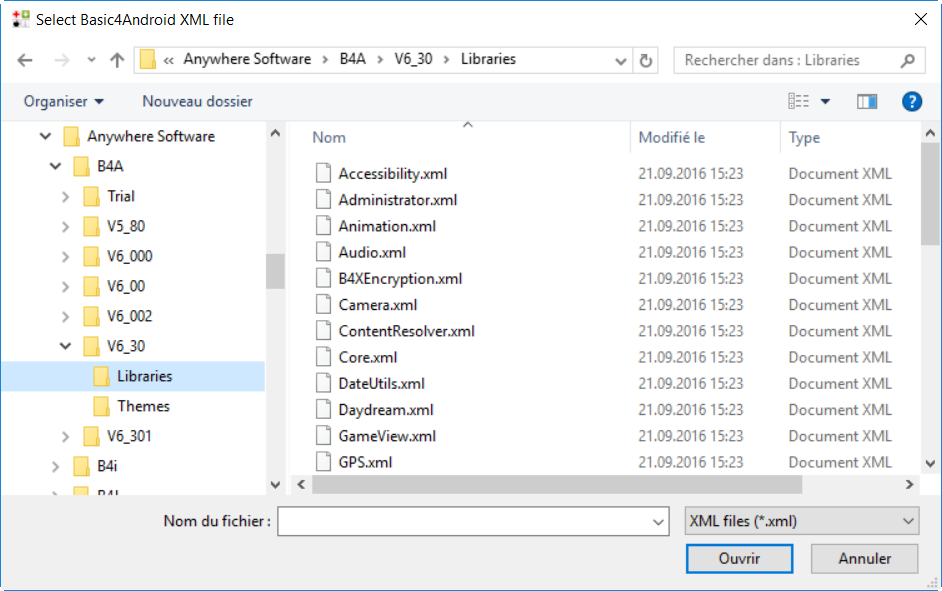
 B4A help files.

 B4i help files.

 B4J help files.

 B4R help files.

 Standard libraries



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

Here  you can select the directory where the standard libraries are saved.

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

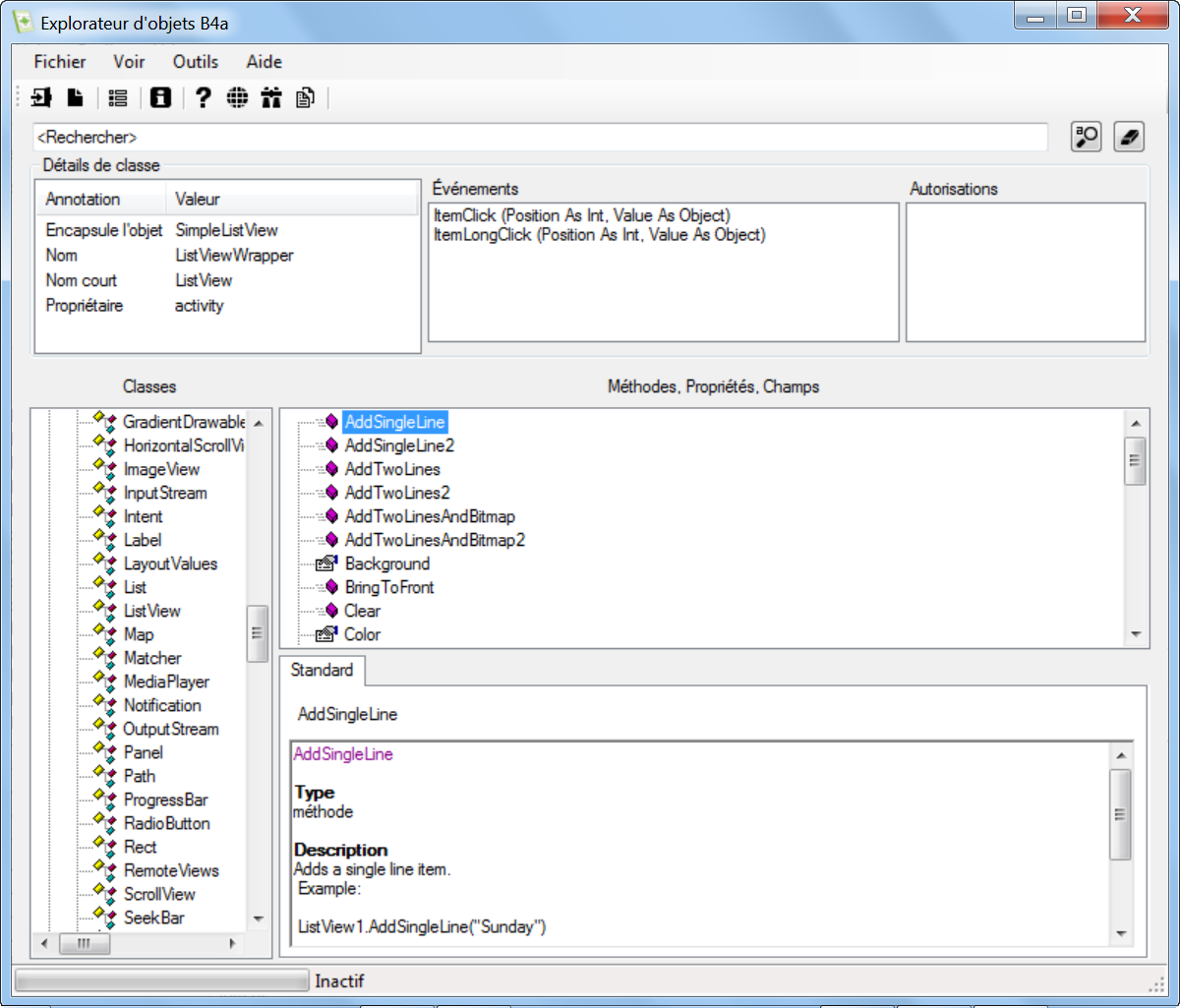
## 

## 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](http://www.basic4ppc.com/forum/basic4android-getting-started-tutorials/25682-help-documentation-docloader-16.html#post148746).

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



# Tips

These are Erels’ tips for B4x developers ([[B4X] Tips for B4X developers](https://www.b4x.com/android/forum/threads/b4x-tips-for-b4x-developers.62121/#post-510240)).

## Separate data from code

Putting the data directly into the code makes your program unreadable and less maintainable.   
There are many simple ways to deal with data. For example you can add a text file to the Files tab and read it to a List with:

Dim data As List = File.ReadList(File.DirAssets, "SomeFile.txt")

## Don't Repeat Yourself (DRY principle).

If you find yourself coping and pasting the same code snippet multiple times and then making a small change then it is a good idea to stop and try to find a more elegant solution.  
Repeated code is difficult to maintain and update. The Sender keyword can help in many cases (old and still relevant tutorial: [Tick-Tack-Toe: working with arrays of views](https://www.b4x.com/android/forum/threads/8506/#content)).

## Map collection

All developers should know how to use a Map collection. This is by far the most useful collection. Tutorial: <https://www.b4x.com/android/forum/threads/map-collection-the-most-useful-collection.60304/>

## New technologies and features.

Don't be afraid to learn new things. As developers we always need to learn new things. Everything is evolving whether we want it or not. I will give [MQTT](https://www.b4x.com/android/forum/threads/59471/#content) as a good example. I wasn't familiar with this technology. When I started learning about it I was a amazed to see how easy and powerful this solution is.  
B4X specific features that all developers should be aware of:  
- Smart strings literal: <https://www.b4x.com/android/forum/threads/50135/#content>  
- For Each iterator: <https://www.b4x.com/android/forum/threads/loops.57877/>  
- Classes: <https://www.b4x.com/android/forum/threads/18626/#content>

## Logs

You should monitor the logs while your app is running. Especially if there is any error. If you are unable to see the logs for some reason then take the time to solve it. Specifically with B4A-Bridge the logs will only appear in Debug mode. If you encounter an issue that only happens in release mode then you need to switch to usb debug mode.

## B4A Avoid calling DoEvents.

DoEvents interferes with the internal message queue. It can cause unexpected issues. There are very few cases where it is required. This was not the case when B4A v1.0 was released. Since then the libraries have evolved and now offer better solutions. For example if the database operations are too slow (and you are correctly using transactions) then you should switch to the asynchronous methods. Or you should use [Sleep](#_Sleep) or [Wait For](#_Wait_For).

## Strings are made of characters not bytes.

Don't try to store raw bytes as strings. It doesn't work. Use arrays of bytes instead. The proper way to convert bytes to strings is with base 64 encoding or ByteConverter.HexFromBytes.

## B4A Use services, especially the Starter service

Services are simpler than Activities. They are not paused and are almost always accessible.   
**Three general rules about global variables:**  
1. All non-UI related variables should be declared in Process\_Globals.   
2. Public (process\_global) variables should be declared and set / initialized in Service\_Create of the Starter service.  
3. Activity process globals should only be initialized if FirstTime is true.  
  
This is only relevant to B4A. It is simpler in B4J and B4i as there is no special life cycle and the modules are never paused.

## UI Layouts

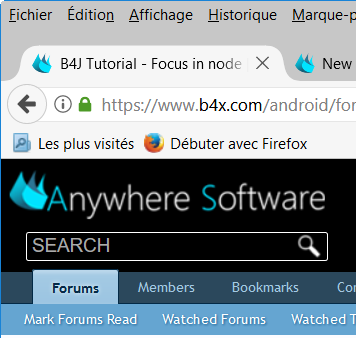
B4X provides several tools to help you implement flexible layouts that adapt to all screen sizes. The main tools are: anchors and designer script. Avoid adding multiple variants (two are fine). Variants were introduced in v1.00, before the other features. Variants are difficult to maintain and can be replaced with scripts.  
Anchors are very simple and powerful.  
Don't overuse percentage units (unless you are building a game).  
[http://www.basic4ppc.com/forum/basi...ing-multiple-screens-tips-best-practices.html](http://www.basic4ppc.com/forum/basic4android-getting-started-tutorials/17647-supporting-multiple-screens-tips-best-practices.html)

## B4J as a backend solution.

B4A, B4i, B4J share the same language, same concepts and mostly the same APIs. It is also simple to exchange data between the different platforms with B4XSerializator.   
It is easy to implement powerful server solutions with B4J. Especially when the clients are implemented with B4A, B4i or B4J.

## Search.

Use the forum search feature. You can filter results by adding the platform(b4a for example) to the query or by clicking on one of the filters in the results page.  
Most of the questions asked in the forum can be solved with a few searches.



## Notepad++.

At one point or another we need to work with text files. I highly recommend all developers to use a good text editor that shows the encoding, the end of line characters and other important features. <https://notepad-plus-plus.org/>