

B4X 手册

B4A B4i B4J B4R

B4X 基础语言

目录

1. B4X 平台	6
2. BASIC	7
3. 变量和对象	8
3.1. 变量类型	8
3.2. 变量名称	11
3.3. 声明变量	11
3.3.1. 简单变量	11
3.3.2. 数组变量	13
3.3.3. 常量变量 Const 关键字	15
3.3.4. 视图/节点 (对象) 数组	15
3.3.5. 类型变量 只限 B4A、B4i 和 B4J 专用	17
3.4. 铸件	19
3.5. 范围	20
3.5.1. 过程变量	20
3.5.2 Activity variables B4A only	21
3.5.3 Local variables	21
4.1.1 Program Start	23
4.1.2 Process global variables	24
4.1.3 Activity variables	24
4.1.4 Starter service	24
4.1.5 Program flow	25
4.1.6 Sub Process_Globals / Sub Globals	26
4.1.7 Sub Activity_Create (FirstTime As Boolean)	26
4.1.8 Variable declaration summary	27
4.1.10 Activity.Finish / ExitApplication	29
5.1.1 Mathematical expressions	35
5.1.2 Relational expressions	36
5.1.3 Boolean expressions	36
🌀 Array	40
🌀 CallSubDelayed (Component As Object, Sub As String)	41

Catch.....	41
☒ Continue	42
☒ CreateMap	42
☒ Dim.....	42
☒ Exit	43
☒ If	43
☒ IIf.....	43
☒ Is	44
Syntax: Return [value]	46
☒ RndSeed (Seed As Long)	46
☒ Select.....	47
Sub.....	48
☒ Try	48
Type.....	49
☒ Until.....	49
☒ While	49
5.3.1 If – Then – Else	50
5.3.2 IIf Inline If	51
5.3.3 Select – Case	53
5.4.1 For – Next.....	55
5.4.2 For - Each	56
5.4.3 Do - Loop	57
5.6.1 Declaring	60
5.6.2 Calling a Sub	60
5.6.3 Calling a Sub from another module	60
5.6.4 Naming	60
5.6.5 Parameters	61
5.6.6 Returned value.....	62
5.7.1 Sleep	63
5.7.2 Wait For.....	64
5.7.3 Code Flow	65
5.7.4 Waiting for a resumable sub to complete.....	67
5.7.5 Resumable Sub return value.....	67
5.7.7 Dialogs	71
5.7.8 SQL with Wait For.....	71
5.7.9 Notes & Tips	73

5.8.1 B4A	74
5.8.2 B4i	78
5.8.3 B4J.....	80
5.8.4 B4R.....	85
5.8.5 User interface summary	85
5.9.1 Standard libraries.....	88
5.9.2 Additional libraries folder	88
5.9.3 B4X Libraries *.b4xlib.....	91
5.9.4 Load and update a Library	92
5.9.5 Error message "Are you missing a library reference?"	93
5.10.1 B4A, B4i, B4J String.....	94
5.10.2 String concatenation.....	95
5.10.3 B4A, B4i, B4J StringBuilder.....	96
5.10.4 Smart String Literal	97
5.10.5 B4A, B4i CharSequence CSBuilder.....	100
5.10.6 B4J TextFlow class	107
5.10.7 B4R.....	107
5.11.1 B4A, B4i, B4J	110
5.11.2 B4X NumberFormatter	110
5.11.3 B4R.....	110
5.13.1 File object.....	113
5.13.2 Filenames	119
5.13.3 Subfolders.....	119
5.13.4 B4A, B4J TextWriter	119
5.13.5 B4A, B4J TextReader	120
5.13.6 Text encoding.....	121
5.16.1 Getting started	128
5.16.2 Standard Class module	132
7.12.1 Encoding.....	140

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 11.0

B4i version 7.50

B4J version 9.10

B4R version 3.71

[B4X Booklets:](#)

B4X Getting Started

B4X Basic Language

B4X IDE Integrated Development Environment

B4X Visual Designer

B4X Help tools

B4XPages Cross-platform projects

B4X CustomViews

B4X Graphics

B4X XUI B4X User Interface

B4X SQLite Database

B4X JavaObject NativeObject

B4R Example Projects

You can consult these booklets online in this link [\[B4X\] Documentation Booklets](#).


Be aware that external links don't work in the online display.

1. B4X 平台

B4X 是一个给于不同平台的 BASIC 编程语言软件套件。

B4X 套件比任何其他工具支持更多的平台

ANDROID | IOS | WINDOWS | MAC | LINUX | ARDUINO | RASPBERRY PI | ESP8266 | 和更多...

- **B4A**  **Android 安卓**

B4A 是一款 **100% 免费** 的安卓应用程序开发工具，它包括快速开发任何类型的安卓应用程序所需的所有功能。

- **B4i**  **iOS**

B4i 是原生 iOS 应用程序的开发工具。

B4i 遵循与 B4A 相同的概念，允许您重用大部分代码并为 Android 和 iOS 构建应用程序。

- **B4J**  **Java / Windows / Mac / Linux / Raspberry PI**

B4J 是一款 **100% 免费** 的桌面、服务器和物联网解决方案开发工具。

使用 B4J，您可以轻松创建桌面应用程序 (UI)、控制台程序（非 UI）和服务器解决方案。

编译后的应用程序可以在 Windows、Mac、Linux 和 ARM 板（如树莓派）。

- **B4R**  **ARDUINO** **Arduino / ESP8266**

B4R 是 **100% 免费** 的原生 Arduino 和 ESP8266 程序开发工具。B4R 遵循其他 B4X 工具的相同概念，提供简单而强大的开发工具。

B4R、B4A、B4J 和 B4i 共同构成物联网 (IoT) 的最佳开发解决方案。

- **B4XPages**

B4XPage 是 B4A、B4i 和 B4J 的内部库，允许轻松开发跨平台程序。

B4XPages 在 B4XPages 跨平台项目手册中有详细的解释。即使您只想在一个平台上进行开发，使用 B4XPage 库也很有趣，它使程序流程更简单，尤其是对于 B4A。

2. BASIC

BASIC (对于 **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode 的缩写) 是一系列通用的高级编程语言，其设计理念强调易用性。

在 1964 年，John G. Kemeny 和 Thomas E. Kurtz 在美国新罕布什尔州的达特茅斯学院设计了原始的 BASIC 语言。他们希望让科学和数学以外领域的学生能够使用计算机。当时，几乎所有使用计算机都需要编写定制软件，这是只有科学家和数学家倾向于学习的东西（来源维基百科）。

3. 变量和对象

变量是赋予某些已知或未知数量或信息的符号名称，目的是允许名称独立于它所代表的信息使用。计算机源代码中的变量名称通常与数据存储位置相关联，因此也与它的内容相关联，这些可能会在程序执行过程中发生变化（源维基百科）。

有两种类型的变量：原始类型和非原始类型。

原始包括数字类型：Byte, Short, Int, Long, Float and Double.

原始还包括：Boolean and Char.

3.1. 变量类型

B4A, B4i, B4J

类型列表及其范围：

B4X	类型	最小值	最大值
Boolean	boolean	False	True
Byte	integer 8 bits	-2^7 -128	$2^7 - 1$ 127
Short	integer 16 bits	-2^{15} -32768	$2^{15} - 1$ 32767
Int	integer 32 bits	-2^{31} -2147483648	$2^{31} - 1$ 2147483647
Long	long integer 64 bits	-2^{63} -9223372036854775808	$2^{63} - 1$ 9223372036854775807
Float	floating point number 32 bits	-2^{-149} 1.4E-45	$(2^{-2} - 2^{-23}) * 2^{127}$ 3.4028235 E 38
Double	double precision number 64 bits	-2^{-1074} 2.2250738585072014 E 308	$(2^{-2} - 2^{-52}) * 2^{1023}$ 1.7976931348623157 E 308
Char	character 文字		
String	array of characters 字符数组		

B4R

类型列表及其范围：

数字类型：

Byte	0 - 255	
Int (2 bytes)	-32,768 – 32,768	类似于其他 B4X 工具中的 Short 类型。
UInt (2 bytes)	0 – 65,535	B4R 专用。
Long (4 bytes)	-2,147,483,648 - 2,147,483,647	类似于其他 B4X 工具中的 Int 类型。
ULong (4 bytes)	0 - 4,294,967,295	B4R 专用。
Double (4 bytes)	4 bytes floating point.	类似于其他 B4X 工具中的 Float 类型。
Float 与 Double 相同。		
Short 与 Int 相同。		

以上适用于所有电路板，包括 Arduino Due。

其他类型：

Boolean True 或 False。实际上，它被保存为一个值为 1 或 0 的字节。

String 字符串由以空字节结尾的字节数组组成（值为 0 的字节）。

Object 对象可以保存其他类型的值。

原始类型总是按值传递给其他子对象或分配给其他变量。

例如：

```
Sub S1
  Private A As Int
  A = 12          变量 A = 12
  S2(A)          它按值传递给例程 S2
  Log(A)         ' 打印 12  变量 A 仍然等于 12，即使 B 在例程 S2 中改变了。
End Sub

Sub S2(B As Int)  变量 B = 12
  B = 45          其值更改为 B = 45
End Sub
```

所有其他类型，包括原始类型数组和字符串，都归类为非原始类型。

当您将非原始类型传递给 sub 或将其分配给不同的变量时，将传递引用的副本。

这意味着数据本身不会重复。

它与通过引用传递略有不同，因为您无法更改原始变量的引用。

所有类型都可以视为对象。

Lists 和 Maps 之类的 Collections 与 Objects 一起使用，因此可以存储任何值。

下面是一个常见错误的示例，其中开发人员试图将多个数组添加到列表中：

```
Private arr(3) As Int
Private List1 As List
List1.Initialize
For I = 1 To 5
    arr(0) = I * 2
    arr(1) = I * 2
    arr(2) = I * 2
    List1.Add(arr)    '将整个数组添加为单个项目
Next
arr = List1.Get(0)    '获取列表中的第一项
Log(arr(0))           '这里会打印什么 ???
```

您可能预计它打印 2。但是，它会打印 10。

我们创建了一个数组并将该数组的 5 个引用添加到列表中。

单个数组中的值是上次迭代中设置的值。

为了解决这个问题，我们需要在每次迭代时创建一个新数组。

这是通过每次迭代调用 `Private` 来完成的：

```
Private arr(3) As Int '在这种情况下，这个称呼是多余的。
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)    '将整个数组添加为单个物品
Next
arr = List1.Get(0)    '从列表中获取第一个物品
Log(arr(0))           '将打印 2
```

3.2. 变量名称

除了保留字外，您可以为变量指定任何名称。

变量名必须以字母开头，并且必须由以下字符 A-Z、az、0-9 和下划线 “_” 组成，不能有空格，不能有括号等。

变量名不区分大小写，这意味着 Index 和 index 指的是同一个变量。

但是给它们起有意义的名字是一种很好的做法。

例子：

Interest = Capital * Rate / 100	是有意义
n1 = n2 * n3 / 100	没有意义

对于 Views (B4A, B4i), Nodes (B4J), 在名称中添加一个定义其类型的三个字符的前缀很有用。例子：

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

3.3. 声明变量

3.3.1. 简单变量

变量声明为 `Private` 或者 `Public` 关键词后跟变量名和 `As` 关键词然后是变量类型。详情请看 [chapter Scope](#)。存在着 `Dim` 关键词，这是为了兼容性而维护的。

例子：

<code>Private Capital As Double</code>	将三个变量声明为 Double, 双精度数。
<code>Private Interest As Double</code>	
<code>Private Rate As Double</code>	

<code>Private i As Int</code>	声明三个变量为 Int, 整数。
<code>Private j As Int</code>	
<code>Private k As Int</code>	

<code>Private lblCapital As Label</code>	将三个变量声明为标签视图。
<code>Private lblInterest As Label</code>	
<code>Private lblRate As Label</code>	

<code>Private btnNext As Button</code>	将两个变量声明为按钮视图。
<code>Private btnPrev As Button</code>	

也可以用简短的方式声明相同的变量。

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

变量名用逗号分隔，后跟类型声明。

以下变量声明有效：

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

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

如果我们想在代码中使用它们，就必须声明视图名称。

例如，如果我们要在代码中更改 Label 视图中的文本，例如

```
lblCapital.Text = "1200",
```

我们需要通过它的名字 lblCapital 来引用这个 Label view，这是通过 Private 声明完成的。

如果我们从未在代码中的任何地方引用此 Label 视图，则不需要声明。

对该视图使用事件例程也不需要声明。

要将值分配给变量，请写入其名称后跟等号再后跟值，例如：

```
Capital = 1200
```

```
LastName = "SMITH"
```

请注意，对于 Capital，我们只写了 1200，因为 Capital 是一个数字。

但是对于 LastName，我们写了 "SMITH"，因为 LastName 是一个字符串。

字符串必须始终写在双引号之间。

3.3.2. 数组变量

数组是可以通过索引选择的数据或对象的集合。数组可以有多个维度。

声明包含 Private 或 Public 关键字，后跟变量名 LastName、方括号(50)之间的项目数、关键字 As 和变量类型 String。

有关详细信息，请参阅 [chapter Scope](#)。存在 Dim 关键字，这是为了兼容性而维护的。

注意：B4R 只支持一维数组！

例子：

```
Public LastName(50) As String    一维字符串数组，物品总数 50。
```

```
Public Matrix(3, 3) As Double    二维数组 Doubles，物品总数 9。
```

```
Public Data(3, 5, 10) As Int     三维整数数组，物品总数 150。
```

数组中每个维度的第一个索引是 0。

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

最后一个索引等于每个维度中的项目数减 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
```

或者

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

此示例显示如何访问三维数组中的所有项目。

```
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
```

声明数组的一种更通用的方法是使用变量。

```
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
```

我们将变量声明为 `Public NbPers = 10 As Int` 并将其值设置为 10。

然后我们用这个变量来声明数组，而不是像以前那样用数字 10 来声明。

最大的优点是如果在某个时候我们需要更改项目的数量，我们只更改『一个』值。

对于 Data 数组，我们可以使用以下代码。

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

和访问例程。

```
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
```

使用 Array 关键字填充数组：

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

3.3.3. 常量变量 Const 关键字

Const 变量是不能在代码中的任何地方更改的常量变量。

为此，我们在 *Private* 或 *Public* 之后使用 *Const* 关键字，如下所示，

```
Private Const Size As Int = 10
Public Const ItemNumber As Int = 100
```

3.3.4. 视图/节点 (对象) 数组

视图/节点或对象也可以在一个数组中。以下代码显示了一个示例：

在 B4A 和 B4i 中，用户界面对象在 B4J 中称为 *视图*(views)和 *节点*(nodes)。

在下面的示例中，按钮 (Button) 通过代码添加到父视图/节点。

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
    Next
End Sub
```

```

        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

```

按钮也可以添加到布局文件中，在这种情况下，它们既不能被初始化，也不能被添加到父视图/节点，并且文本和标签属性也应该在设计器中设置。
在这种情况下，代码将如下所示：

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

```

3.3.5. 类型变量 只限 B4A、B4i 和 B4J 专用

类型不能是私有的。一旦声明它在任何地方都可用（类似于 **Class** 模块）。
声明它们的最佳位置是在 Main 模块的 Process_Globals 例程中。

让我们用一个人的数据重用这个例子。
我们可以使用 **Type** 关键字定义一个个人类型变量，而不是单独声明每个参数：

```

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

```

新的个人类型是 **Person**，然后我们声明此个人类型的单个变量或数组。

要访问特定项目，请使用以下代码。

```
CurrentUser.FirstName  
CurrentUser.LastName
```

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

变量名称，后跟一个点和所需的参数。

如果变量是一个数组，则名称后跟括号之间的所需索引。

可以将一个类型化变量分配给另一个相同类型的变量，如下所示。

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

3.4. 铸件

B4X 根据需要自动铸件类型。 它还自动将数字转换为字符串，反之亦然。

在许多情况下，您需要将 Object 显式铸件为特定类型。

这可以通过将 Object 分配给所需类型的变量来完成。

例如，Sender 关键字引用一个对象，它是引发事件的对象。

以下代码更改按下按钮的颜色。

请注意，有多个按钮共享相同的事件子。

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 '将对象铸件成按钮
```

```
btn.Color = Colors.RGB(Rnd(0, 255), Rnd(0, 255), Rnd(0, 255))
```

End Sub

上面的代码也可以写得更优雅：

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
```

3.5. 范围

3.5.1. 过程变量

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 !

3.5 Scope

3.5.2 Activity variables B4A only

These variables are contained by the activity.

You should declare these variables inside Sub Globals.

These variables are "private" and can only be accessed from the current activity module.

All object types can be declared as activity variables.

Every time the activity is created, Sub Globals is called (before Activity_Create).

These variables exist as long as the activity exists.

3.5.3 Local variables

Variables declared in a subroutine are local to this subroutine.

They are "private" and can only be accessed from within the subroutine where they were declared.

All objects types can be declared as local variables.

At each call of the subroutine the local variables are initialized to their default value or to any other value you have defined in the code and are 'destroyed' when the subroutine is exited.

3.6 Tips

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

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

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

If we only want to disable buttons:

```
For i = 0 To 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.

4 Program flow / Process life cycle

4 Program flow / Process life cycle

Each platform has its own program flow.

To make cross-platform projects it is now easier to do with B4XPages.

B4XPages is explained in detail in the B4XPages Cross-platform projects booklet.

4.1 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.

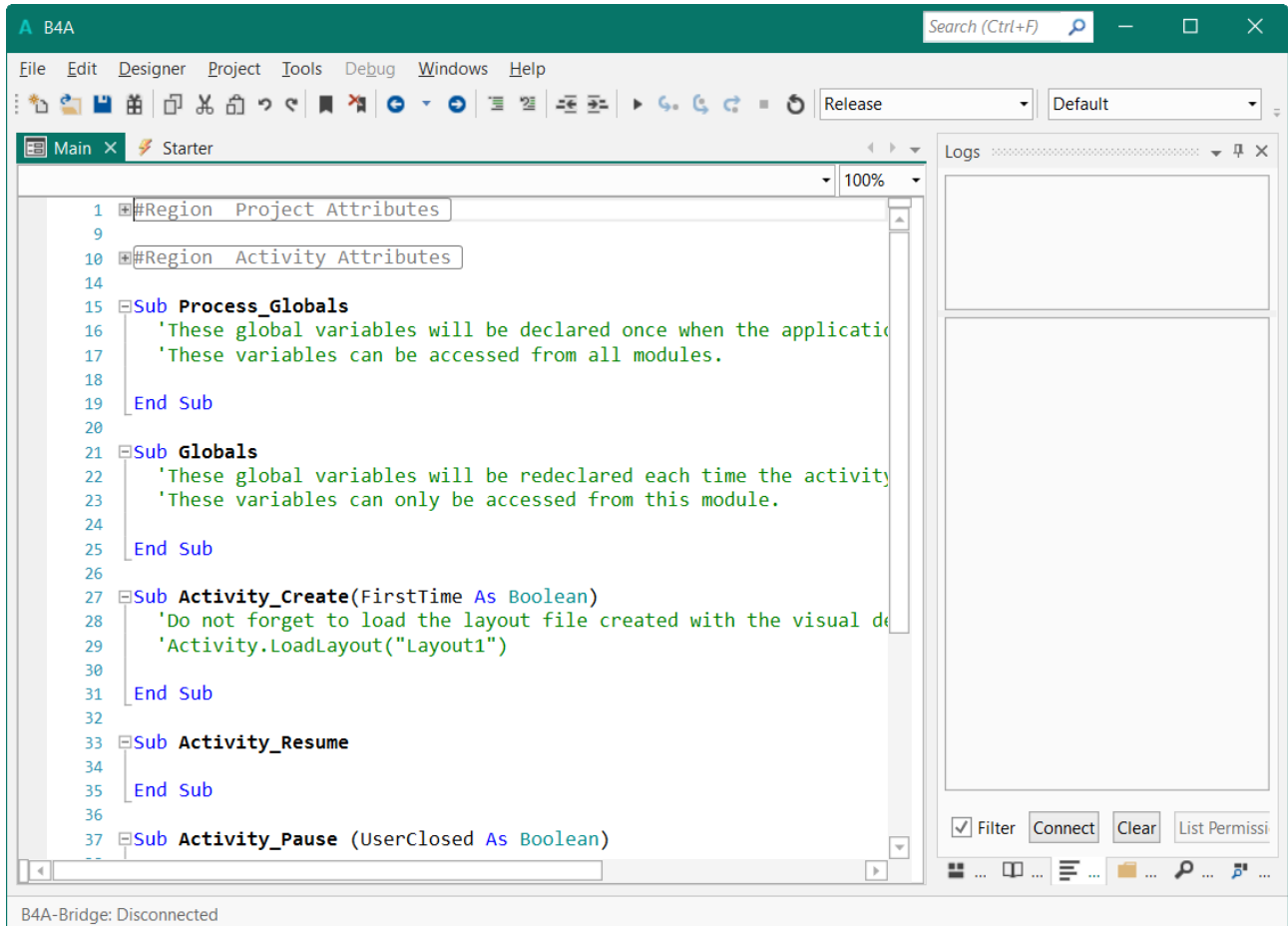
Activities are somewhat similar to Windows Forms.


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).

4.1.1 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](#)

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).

4.1.2 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 launched 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.

4.1.3 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.

4.1.4 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.

4.1.5 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](#)
Here we declare all Private variables for the given Activity.
- [Sub Activity_Create](#)
Here we load layouts and initialize activity objects added by code
- [Activity Resume](#)
This routine is run every time the activity changes its state.
- [Activity Pause](#)
This routine is run when the Activity is paused, like orientation change, lauch of another activity etc.

4.1.6 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.

4.1.7 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.

4.1.8 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 Activity_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 outside the Class or Code module.

Class modules are explained in detail in the [B4X Booklet CustomViews Booklet](#).

- 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 !

4.1.9 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).

4.1.10 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](#).

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.

4.2 Program flow B4i

4.2 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, they 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.

4.3 Program flow B4J

4.3 Program flow B4J

The program flow in B4J 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
```

If you use anchors in the Designer, the Resize event will not be necessary in most cases.

4.4 Program flow B4R

4.4 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.

4.5 Program flow comparison

4.5 Program flow comparison B4A / B4i / B4J**4.5.1 Program start B4A / B4i / B4J****B4A**

Main Process_Globals

Starter Process_Globals

Other modules Process_Globals

Starter Service_Create

Starter Service_Start

Main Globals

Main Activity_Create

FirstTime = True

Main Activity_Resume

B4i

Main Process_Globals

Other modules Process_Globals

Main Application_Start

Main Page1_Resize

B4J

Main Process_Globals

Other modules Process_Globals

Main AppStart

Main MainForm_Resize

4.5.2 Rotating device B4A / B4i**B4A**

Main Activity_Pause

Main Globals

Main Activity_Create

FirstTime = False

Main Activity_Resume

B4i

Main Page1_Resize

4.6 B4XPages program flow

4.6 B4XPages program flow

For cross-platform projects with the B4XPages library the program flow is the same for all three platforms. All the platform specific code is hidden in the B4XPages library and transparent to the programmer.

The B4XPagesThreePages project in the B4XPages Cross-platform projects booklet shows the program flow when navigating between Pages.

Examples:

Start of the project, the routines below are executed:

- MainPage Create
- MainPage Foreground
- MainPage Appear
- MainPage Resize

Opening a Page, Page2 in the example:

- Page2 Create
- Page2 Foreground
- Page2 Appear

Closing a Page, Page2 in the example:

- Page2 Disappear

5 Basic language

5 Basic language

5.1 Expressions

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

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

5.1.1 Mathematical expressions

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

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

Examples:

$$4 + 5 * 3 + 2 = 21 \quad > \quad 4 + 15 + 2$$

$$(4 + 5) * (3 + 2) = 45 \quad > \quad 9 * 5$$

$$(4 + 5)^2 * (3 + 2) = 405 \quad > \quad 9^2 * 5 > 81 * 5$$

$$\text{Power}(4 + 5, 2) * (3 + 2)$$

$$11 \text{ Mod } 4 = 3 \quad > \quad \text{Mod is the remainder of } 11 / 4$$

$$23^3 \text{ Power}(23, 3) \quad > \quad 23 \text{ at the power of } 3$$

$$- 2^2 = - 4$$

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

5.1 Expressions

5.1.2 Relational expressions

In computer science in relational expressions an operator tests some kind of relation between two entities. These include numerical equality (e.g., $5 = 5$) and inequalities (e.g., $4 \geq 3$).

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
>=	$x \geq y$	if the value of the left expression is greater than or equal to that of the right
<=	$x \leq y$	if the value of the left expression is less than or equal to that of the right

5.1.3 Boolean expressions

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

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

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

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

5.2 Standard keywords

Not all keywords are available in B4R.

-  [**Abs**](#) (Number [As Double](#)) [As Double](#)
-  [**ACos**](#) (Value [As Double](#)) [As Double](#)
-  [**ACosD**](#) (Value [As Double](#)) [As Double](#)
-  [**Array**](#)
-  [**Asc**](#) (Char [As Char](#)) [As Int](#)
-  [**ASin**](#) (Value [As Double](#)) [As Double](#)
-  [**ASinD**](#) (Value [As Double](#)) [As Double](#)
-  [**ATan**](#) (Value [As Double](#)) [As Double](#)
-  [**ATan2**](#) (Y [As Double](#), X [As Double](#)) [As Double](#)
-  [**ATan2D**](#) (Y [As Double](#), X [As Double](#)) [As Double](#)
-  [**ATanD**](#) (Value [As Double](#)) [As Double](#)
-  [**BytesToString**](#) (Data() [As Byte](#), StartOffset [As Int](#), Length [As Int](#), CharSet [As String](#)) [As String](#)
-  [**CallSub**](#) (Component [As Object](#), Sub [As String](#)) [As Object](#)
-  [**CallSub2**](#) (Component [As Object](#), Sub [As String](#), Argument [As Object](#)) [As Object](#)
- [**CallSub3**](#) (Component [As Object](#), Sub [As String](#), Argument1 [As Object](#), Argument2 [As Object](#)) [As Object](#)
-  [**CallSubDelayed**](#) (Component [As Object](#), Sub [As String](#))
-  [**CallSubDelayed 2**](#) (Component [As Object](#), Sub [As String](#), Argument [As Object](#))
-  [**CallSubDelayed 3**](#) (Component [As Object](#), Sub [As String](#), Argument1 [As Object](#), Argument2 [As Object](#)) [**Catch**](#) [**cE**](#) [As Double](#)
-  [**Ceil**](#) (Number [As Double](#)) [As Double](#)
-  [**CharsToString**](#) (Chars() [As Char](#), StartOffset [As Int](#), Length [As Int](#)) [As String](#)
-  [**Chr**](#) (UnicodeValue [As Int](#)) [As Char](#)
-  [**Continue**](#)
-  [**Cos**](#) (Radians [As Double](#)) [As Double](#)
-  [**CosD**](#) (Degrees [As Double](#)) [As Double](#)
-  [**Double cPI**](#) [As Double](#) [**CreateMap**](#)
-  [**CRLF**](#) [As String](#)
-  [**Dim**](#)
-  [**Exit**](#)
-  [**False**](#) [As Boolean](#)
-  [**Floor**](#) (Number [As Double](#)) [As Double](#)
-  [**For**](#)
-  [**GetType**](#) (object [As Object](#)) [As String](#)
-  [**If**](#)
-  [**Is**](#)
-  [**IsNumber**](#) (Text [As String](#)) [As Boolean](#)
-  [**LoadBitmap**](#) (Dir [As String](#), FileName [As String](#)) [As Bitmap](#)
-  [**LoadBitmapResize**](#) (Dir [As String](#), FileName [As String](#), Width [As Int](#), Height [As Int](#), KeepAspectRatio [As Boolean](#)) [As Bitmap](#)
-   [**LoadBitmapSample**](#) (Dir [As String](#), FileName [As String](#), MaxWidth [As Int](#), MaxHeight [As Int](#)) [As Bitmap](#)

 **Log** (Message **As String**)
 **Logarithm** (Number **As Double**, Base **As Double**) **As Double**
 **LogColor** (Message **As String**, Color **As Int**)
 **Max** (Number1 **As Double**, Number2 **As Double**) **As Double**
 **Me** **As Object**
 **Min** (Number1 **As Double**, Number2 **As Double**) **As Double**
 **Not** (Value **As Boolean**) **As Boolean**  **Null** **As Object**
 **NumberFormat** (Number **As Double**, MinimumIntegers **As Int**, MaximumFractions **As**
Int) **As**
String
 **NumberFormat2** (Number **As Double**, MinimumIntegers **As Int**, MaximumFractions **As Int**,
MinimumFractions **As Int**, GroupingUsed **As Boolean**) **As String**
 **Power** (Base **As Double**, Exponent **As Double**) **As Double**
 **QUOTE** **As String**
 **Regex** **As Regex**
 **Return**
 **Rnd** (Min **As Int**, Max **As Int**) **As Int**
 **RndSeed** (Seed **As Long**)
 **Round** (Number **As Double**) **As Long**
 **Round2** (Number **As Double**, DecimalPlaces **As Int**) **As Double**
 **Select**
 **Sender** **As Object**
 **Sin** (Radians **As Double**) **As Double**
 **SinD** (Degrees **As Double**) **As Double**
 **Sleep** (Milliseconds **As Int**)
 **SmartStringFormatter** (Format **As String**, Value **As Object**) **As String**
 **Sqrt** (Value **As Double**) **As Double**
 **Sub**
 **SubExists** (Object **As Object**, Sub **As String**) **As Boolean**
 **TAB** **As String**
 **Tan** (Radians **As Double**) **As Double**
 **TanD** (Degrees **As Double**) **As Double**
 **True** **As Boolean**
 **Try**
 **Type**
 **Until**

While

Abs

(Number
As Double)
As Double
Returns
the
absolute
value.

 **ACos** (Value **As Double**) **As Double**

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

ACosD (Value As Double) As Double

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

Array

Creates a single dimension array of the specified type.

The syntax is: Array [As type] (list of values).

If the type is omitted then an array of objects will be created.

Example:

```
Dim Days() As String
```

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

Asc (Char As Char) As Int

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

ASin (Value As Double) As Double

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

ASinD (Value As Double) As Double

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

ATan (Value As Double) As Double

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

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

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

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

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

ATanD (Value As Double) As Double

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

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

Decodes the given bytes array as a string.

Data - The bytes array.

StartOffset - The first byte to read.

Length - Number of bytes to read.

CharSet - The name of the character set.

Example:

```
Dim s As String s = BytesToString(Buffer, 0,
Buffer.Length, "UTF-8")
```

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

☞ Calls the given sub. CallSub can be used to call a sub which belongs to a different module. However the sub will only be called if the other module is not paused. In that case an empty string will be returned.

You can use IsPaused to test whether a module is paused.

This means that one activity cannot call a sub of a different activity. As the other activity will be paused for sure.

CallSub allows an activity to call a service sub or a service to call an activity sub.

Note that it is not possible to call subs of code modules.

CallSub can also be used to call subs in the current module. Pass Me as the component in that case.

Example:

```
CallSub(Main, "RefreshData")
```

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

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

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

☞ **CallSubDelayed** (Component As Object, Sub As String)

CallSubDelayed is a combination of StartActivity, StartService and CallSub.

Unlike CallSub which only works with currently running components, CallSubDelayed will first start the target component if needed.

CallSubDelayed can also be used to call subs in the current module. Instead of calling these subs directly, a message will be sent to the message queue.

The sub will be called when the message is processed. This is useful in cases where you want to do something "right after" the current sub (usually related to UI events).

Note that if you call an Activity while the whole application is in the background (no visible activities), the sub will be executed once the target activity is resumed.

☞ **CallSubDelayed2** (Component As Object, Sub As String, Argument As Object)

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

☞ **CallSubDelayed3** (Component As Object, Sub As String, Argument1 As Object, Argument2 As Object)

Similar to CallSubDelayed. Calls a sub with two arguments.

Catch

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

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

As Double

e (natural logarithm base) constant.

☞ **Ceil** (Number As Double) As Double

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

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

Creates a new String by copying the characters from the array.

Copying starts from StartOffset and the number of characters copied equals to Length.

🔓 **Chr** (UnicodeValue **As Int**) **As Char**

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

🔓 **Continue**

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

🔓 **Cos** (Radians **As Double**) **As Double**

Calculates the trigonometric cosine function. Angle measured in radians.

🔓 **CosD** (Degrees **As Double**) **As Double**

Calculates the trigonometric cosine function. Angle measured in degrees.

🔓 **cPI** **As Double** PI constant.

🔓 **CreateMap**

Creates a Map with the given key / value pairs.

The syntax is: CreateMap (key1: value1, key2: value2, ...)

Example:

```
Dim m As Map = CreateMap("January": 1, "February": 2)
```

🔓 **CRLF** **As String**

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

🔓 **Dim**

Declares a variable.

Syntax:

Declare a single variable:

```
Dim variable name [As type] [= expression]
```

The default type is String.

Declare multiple variables. All variables will be of the specified type. Dim

```
[Const] variable1 [= expression], variable2 [= expression], ..., [As type]
```

Note that the shorthand syntax only applies to Dim keyword.

Example: `Dim a = 1, b = 2, c = 3 As Int`

Declare an array:

```
Dim variable(Rank1, Rank2, ...) [As type]
```

Example: `Dim Days(7) As String`

☞ The actual rank can be omitted for zero length arrays.

☞ Exit

Exits the most inner loop.

Note that Exit inside a Select block will exit the Select block.

☞ False As Boolean

☞ Floor (Number As Double) As Double

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

☞ For

Syntax:

```
For variable = value1 To value2 [Step
interval] ...
```

Next

If the iterator variable was not declared before it will be of type Int.

Or:

```
For Each variable As type In
collection ... Next
```

Examples:

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

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

GetType (object As Object) As String

Returns a string representing the object's java type.

☞ If

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

```
If condition Then
statement
Else If condition Then
  statement
... Else
statement
End If
```

☞ IIf

Inline If - returns TrueValue if Condition is True and False otherwise. Only the relevant expression is evaluated.

IIf (Condition As BOOL, TrueValue As Object, FalseValue As Object)

Is

Tests whether the object is of the given type.

Note that when a number is converted to object it might change its type to a different type of number

(for example a Byte might be converted to an Int).

Example:

```
For Each v As View in Page1.RootPanel.GetAllViewsRecursive
  If v Is Button Then
    Dim b As Button = v
    b.Color = Colors.Blue
  End If
Next
```

IsNumber (Text As String) As Boolean

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

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

Loads the bitmap.

Note that the Android file system is case sensitive.

You should consider using LoadBitmapSample if the image size is large.

The actual file size is not relevant as images are usually stored compressed.

Example:

```
Activity.SetBackgroundImage(LoadBitmap(File.DirAssets, "SomeFile.jpg"))
```



LoadBitmapResize (Dir As String, FileName As String, Width As Int, Height As Int, KeepAspectRatio As Boolean) As Bitmap

Loads the bitmap and sets its size.

The bitmap scale will be the same as the device scale.

Unlike LoadBitmapSample which requires the container Gravity to be set to FILL,

LoadBitmapResize provides better results when the Gravity is set to CENTER.

Example:

```
Dim bd As BitmapDrawable = Activity.SetBackgroundImage(LoadBitmapResize(File.DirAssets,
"SomeFile.jpg", 100%x, 100%y, True))
```

```
bd.Gravity = Gravity.CENTER
```

Or:

```
Activity.SetBackgroundImage(LoadBitmapResize(File.DirAssets, "SomeFile.jpg", 100%x, 100%y,
True)).Gravity = Gravity.CENTER
```

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

Loads the bitmap.

☞ The decoder will subsample the bitmap if MaxWidth or MaxHeight are smaller than the bitmap dimensions.

This can save a lot of memory when loading large images.

Example:

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

☞ **Log** (Message As String)

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

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

☞ **LogColor** (Message As String, Color As Int)

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

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

Returns the larger number between the two numbers.

☞ **Me As Object**

For classes: returns a reference to the current instance.

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

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

Returns the smaller number between the two numbers.

Not (Value As Boolean) As Boolean

Inverts the value of the given boolean.

☞ **Null As Object**

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

Converts the specified number to a string.

The string will include at least Minimum Integers and at most Maximum Fractions digits. Example:

```
Log(NumberFormat(12345.6789, 0, 2)) '"12,345.68"
```

```
Log(NumberFormat(1, 3, 0)) '"001"
```

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

Converts the specified number to a string.

The string will include at least Minimum Integers, at most Maximum Fractions digits and at least Minimum Fractions digits.

GroupingUsed - Determines whether to group every three integers.

Example:

```
Log(NumberFormat2(12345.67, 0, 3, 3, false)) '"12345.670"
```

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

Returns the Base value raised to the Exponent power.

 **QUOTE** As String

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

 **Regex** As Regex

Regular expressions related methods.

 **Return**

Returns from the current sub and optionally returns the given value.

Syntax: Return [value]

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

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

 **RndSeed** (Seed As Long)

Sets the random seed value.

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

 **Round** (Number As Double) As Long

Returns the closest long number to the given number.

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

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

Select

Compares a single value to multiple values.

Example:

```
Dim value As Int
value = 7
Select
    value
    Case 1
        Log("One")
    Case 2, 4, 6, 8
        Log("Even")
    Case 3, 5, 7, 9
        Log("Odd larger than one")
    Case Else
        Log("Larger than 9") End
Select
```

Sender As Object

Returns the object that raised the event.

Only valid while inside the event sub.

Example:

```
Sub Button_Click
Dim b As Button    b
= Sender
    b.Text = "I've been clicked"
End Sub
```

Sin (Radians As Double) As Double

Calculates the trigonometric sine function. Angle measured in radians.

SinD (Degrees As Double) As Double

Calculates the trigonometric sine function. Angle measured in degrees.

Sleep (Value As Double) As Double

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

SmartStringFormatter (Format As String, Value As Object) As String

Internal keyword used by the Smart String literal.

Sqrt (Value As Double) As Double

Returns the positive square root.

Sub

Declares a sub with the parameters and return type.

Syntax: Sub name [(list of parameters)] [As return-type]

Parameters include name and type.

The lengths of arrays dimensions should not be included.

Example:

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

In this example OtherValues is a single dimension array.

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

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

Tests whether the object includes the specified method.

Returns false if the object was not initialized or not an instance of a user class.

TAB As String

Tab character.

Tan (Radians As Double) As Double

Calculates the trigonometric tangent function. Angle measured in radians.

TanD (Degrees As Double) As Double

Calculates the trigonometric tangent function. Angle measured in degrees.

True As Boolean

Try

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

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


Type

Declares a structure.

Can only be used inside sub Globals or sub Process_Globals.

Syntax:

Type type-name (field1, field2, ...)

Fields include name and type.

Example:

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

Until

Loops until the condition is true.

Syntax:

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

While

Loops while the condition is true.

Syntax:

```
Do While condition ...
Loop
```

5.3 Conditional statements

Different conditional statements are available in Basic.

5.3.1 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
End If
```

The simplest **If-Then** structure.

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

The same but in one line.

```
If b = 0 Then
```

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

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

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

The same but in one line.

Personally, I prefer the structure on several lines, better readable.

An old habit from HP Basic some decades ago, this Basic accepted only one instruction per line.

Note. Difference between:

B4X
Else If

VB
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
<pre>If b = 0 Then</pre>	<pre>If b = 0 Then</pre>
<pre> a = 0</pre>	<pre> c = 1</pre>
<pre> a = 0 End If</pre>	
<pre>c = 1 End If</pre>	

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.

5.3.1.1 Boolean evaluation order

In this example:

```
If InitVar2(Var1) and Var1 > Var2 then ....
```

If InitVar2(Var1) returns false does it stops evaluation or there is no rule ?

It goes from left to right and stops immediately when the result is determined (short circuit evaluation).

This is very important.

It allows writing code such as:

```
If i < List.Size And List.Get(i) = "abc" Then
```

5.3.2 IIf Inline If

IIf - Inline If, also called *ternary if* as it is an operator with three arguments.

```
Label1.Text = IIf(EditText1.Text <> "", EditText1.Text, "Please enter value")
```

IIf is mostly equivalent to this sub:

```
Sub PseudoIIf (Condition As Boolean, TrueValue As Object, FalseValue As Object) As  
Object  
  If Condition = True Then Return TrueValue Else Return FalseValue  
End Sub
```

Unlike this sub, the IIf keyword will only evaluate the relevant expression. This means that this code will work properly:

```
Return IIf(List1.Size > 0, List1.Get(0), "List is empty")
```

(There is another minor difference related to the return type. If it is set explicitly with the new As method, the compiler will avoid casting the values to Object and back to the target type. This is only significant in very tight and long loops).

5.3.3 Select – Case

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

General case:

```
Select TestExpression  Case  TestExpression is the expression to test.
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
Case 12, 24
```

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

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

The **TestExpression** is a character at

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

Case Activity.ACTION_MOVE
```

```
Case Activity.ACTION_UP
```

```
End Select
```

```
End Sub
```

Note. Differences between:

B4X

```
Select Value
```

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

VB

```
Select Case Value
```

```
Case 1 To 4 , 8 To 9
```

In VB the keyword `Case` is added after the `Select` keyword. VB accepts `Case 1 To 4`, this is not implemented in B4X.

5.4 Loop structures

Different loop structures are available in Basic.

5.4.1 For – Next

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

Example:

```
For i = n1 To n2 Step n3      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 $i \leq n2$
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
' code
  If A = 0 Then Exit
' code
Next
```

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

Note : Differences between

B4X	VB
<code>Next</code>	<code>Next i</code>
<code>Exit</code>	<code>Exit For</code>

In VB :

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

5.4.2 For - Each

It is a variant of the For - Next loop.

Example:

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

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

The **For-Each** loop works as below:

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

Example For - Each :

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

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

```
Sum = 0
```

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

Same example but with a For - Next loop :

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

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

```
Sum = 0
```



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

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

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

Same example with a For - Next loop :

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

5.4.3 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
Exit

VB
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

5.5 .As inline casting

5.5 Inline casting As

As - Inline casting. Allows inline casting from one type to another. Some examples:

```
Dim Buttons As List = Array(Button1, Button2, Button3, Button4, Button5)
Dim s As String = Buttons.Get(2).As(B4XView).Text
Buttons.Get(2).As(B4XView).Text = "abc"
Dim j As String = "${
data: { key1:
value1,
complex_key2: {key: value2}
},
items: [0, 1, 2]
} "$

Dim parser As JSONParser parser.Initialize(j)
Dim m As Map = parser.NextObject
Dim value1 As String = m.Get("data").As(Map).Get("key1")
Dim value2 As String = m.Get("data").As(Map).Get("complex_key2").As(Map).Get("key")
```

And, for B4J:

```
Button1.As(JavaObject).RunMethod("setMouseTransparent", Array(True))
```

It can also be used with numbers, which is especially useful when calling external APIs with `JavaObject`, as the types need to be exact (for B4J):

```
Log(Me.As(JavaObject).RunMethod("sum", Array((10).As(Float), (20).As(Double))))
'equivalent to:
Dim jme As JavaObject = Me
Dim f As Float = 10
Dim d As Double = 20
Log(jme.RunMethod("sum", Array(f, d)))
#if
Java
public double sum(float n1, double n2) { return
n1 + n2;
}
#End If
```

5.6 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.

5.6.1 Declaring

A Sub is declared in the following way:

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

It starts with the keyword **Sub**, followed by the Sub's name, followed by a parameter list, followed by the return type and ends with the keywords **End Sub**.

Subs are always declared at the top level of the module, you cannot nest two Subs one inside the other.

5.6.2 Calling a Sub

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

For example:

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

Interest Value returned by the Sub. CalcInterest Sub name.

1235 Capital value transmitted to the Sub.

5.25 Rate value transmitted to the Sub.

5.6.3 Calling a Sub from another module

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

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

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

instead of:

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

5.6.4 Naming

Basically, you can name a Sub any name that's legal for a variable. It is recommended to name the Sub with a significant name, like **CalcInterest** in the example, so you can tell what it does from reading the code.

There is no limit on the number of Subs you can add to your program, but it is not allowed to have two Subs with the same name in the same module.

5.6.5 Parameters

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

The parameter types should be declared directly in the list.

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

In B4X, the parameters are transmitted by value and not by reference.

5.6.6 Returned value

A sub can return a value, this can be any object.

Returning a value is done with the `Return` keyword.

The type of the return value is added after the parameter list.

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

You can return any object.

```
Sub InitList As List
    Private MyList As List
    MyList.Initialize

    For i = 0 To 10
        MyList.Add("Test" & i)
    Next
    Return MyList End
Sub
```

If you want to return an array then you need to add a parenthesis at the end of the object type.

```
Sub StringArray As String ()
    Public strArr(2) As String
    strArr(0) = "Hello" strArr(1)
    = "world!"
    Return strArr End
Sub
```

If you want to return a multidimensional array you need to add comma for supplementary dimension. One comma for a two dimensional array.

```
Sub StringMatrix As String (,)
    Public strMatrix(2,2) As String
    strMatrix(1,1) = "Hello world!"
    Return strMatrix
End Sub
```

5.7 Resumable Subs

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


You find more examples in the [forum](#).

The special feature of resumable subs is that they can be paused, without pausing the executing thread, and later be resumed.

The program doesn't wait for the resumable sub to be continued. Other events will be raised as usual.

Any sub with one or more calls to Sleep or Wait For is a resumable sub.

The IDE shows this indicator  next to the sub declaration:

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

5.7.1 Sleep

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

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

Example:


Sleep(1000)

Using Sleep is simple:

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

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

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

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

```
Log("finished!")
End Sub
```

5.7.2 Wait For

B4X programming languages are event driven. Asynchronous tasks run in the background and raise an event when the task completes.

With the new Wait For keyword you can handle the event inside the current sub.

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

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

    gmap.Initialize("gmap")
    Panel1.AddNode(gmap.AsPane, 0, 0, Panel1.Width, Panel1.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) '<----
        Next
    End If
    Log($"File ${ServerPath2} downloaded. Success = ${Success}")
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.

The difference is in the Wait For lines:

Bad: `Wait For JobDone(job As HttpJob)`

Good: `Wait For (job) JobDone(job As HttpJob)`

5.7.3 Code Flow

```

Sub S1
    Log("S1: A")
S2

```

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

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

The output is:

S1: A

S2: A

S1: B

S2: B

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

5.7.4 Waiting for a resumable sub to complete

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

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

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

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

Logs:

```
FirstSub started
SecondSub started
SecondSub completed
FirstSub completed
```

Notes:

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

5.7.5 Resumable Sub return value

Resumable subs can return a *ResumableSub* value.

Example:

```
Sub Button1_Click
    Sum(1, 2)
    Log("after sum")
End Sub

Sub Sum(a As Int, b As Int)
    Sleep(100) 'this will cause the code flow to return to the parent
    Log(a + b)
```

```
End Sub
Output:
after sum
3
```

This is the reason why it is not possible to simply return a value.

Solution.

Resumable subs can return a new type named ResumableSub. Other subs can use this value to wait for the sub to complete and get the desired return value.

```
Sub Button1_Click
    Wait For(Sum(1, 2)) Complete (Result As Int)
    Log("result: " & Result)
    Log("after sum")
End Sub

Sub Sum(a As Int, b As Int) As ResumableSub
    Sleep(100)
    Log(a + b)
    Return a + b End
Sub
```

```
Output:
3 result:
3 after
sum
```

The above Button1_Click code is equivalent to:

```
Sub Button1_Click
    Dim rs As ResumableSub = Sum(1, 2)
    Wait For(rs) Complete (Result As Int)
    Log("result: " & Result)
    Log("after sum")
End Sub
```

The steps required are:

1. Add *As ResumableSub* to the resumable sub signature.
2. Call Return with the value you like to return.
3. In the calling sub, call the resumable sub with Wait For (<sub here>) Complete (Result As <matching type>)

Notes & Tips:

- If you don't need to return a value but still want to wait for the resumable sub to complete then return Null from the resumable sub and set the type in the calling sub to Object.
- Multiple subs can safely call the resumable sub. The complete event will reach the correct parent.

- You can wait for resumable subs in other modules (in B4A it is relevant for classes only).
- The Result parameter name can be changed.

5.7.6 DoEvents deprecated !

Starting from B4A v7.0 the following warning will appear for DoEvents calls:

DoEvents is deprecated. It can lead to stability issues. Use Sleep(0) instead (if really needed).

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

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

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

Unlike DoEvents which only processed UI related messages, with Sleep all messages will be processed and other events will be raised.

(Note that using Wait For to wait for an event is better than calling Sleep in a loop.)

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

5.7.7 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](#), is really a simple change:

Instead of:

```
Dim res As Int = MsgBox2("Delete?", "Title", "Yes", "Cancel", "No", Null)
If res = DialogResult.POSITIVE Then '...
End If
```

You should use :

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

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

5.7.8 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) SQLNonQueryComplete (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.

5.7.8.1 Queries

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

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

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

Tips:

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

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



```
    Log(LastException)
End If
End Sub

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

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

5.7.8.2 B4J

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

5.7.9 Notes & Tips

- 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.

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 and 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
```

- **CheckBox1_CheckedChanged** (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.

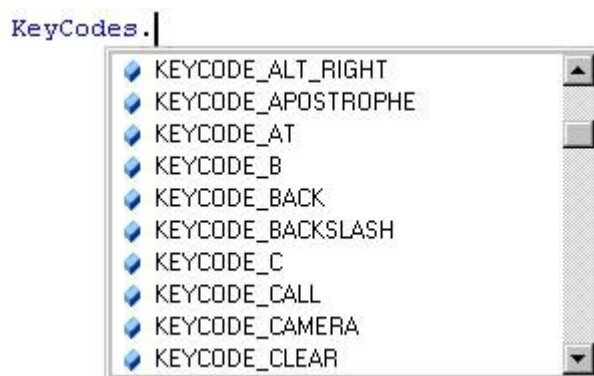
Example:

```
Sub CheckBox1_CheckedChanged(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
```

5.8.2 B4i

User interface objects are called 'Views' in iOS.

Summary of the events for different views:

Views	Events											
	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

.2 Events B4i

The most common

- **Click** Event
the user clicks
Example:

```

Private Sub
Button1_Click
  ' Your code
End Sub

```

- **LongClick**
raised when the
the view and
pressed for a
Example:

```

Private Sub Button1_LongClick
  ' Your code End
Sub

```

- **Touch** (Action As Int, X As Float, Y As Float)

events are:

raised when
on the view.

Event
user clicks on
holds it
while.

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.

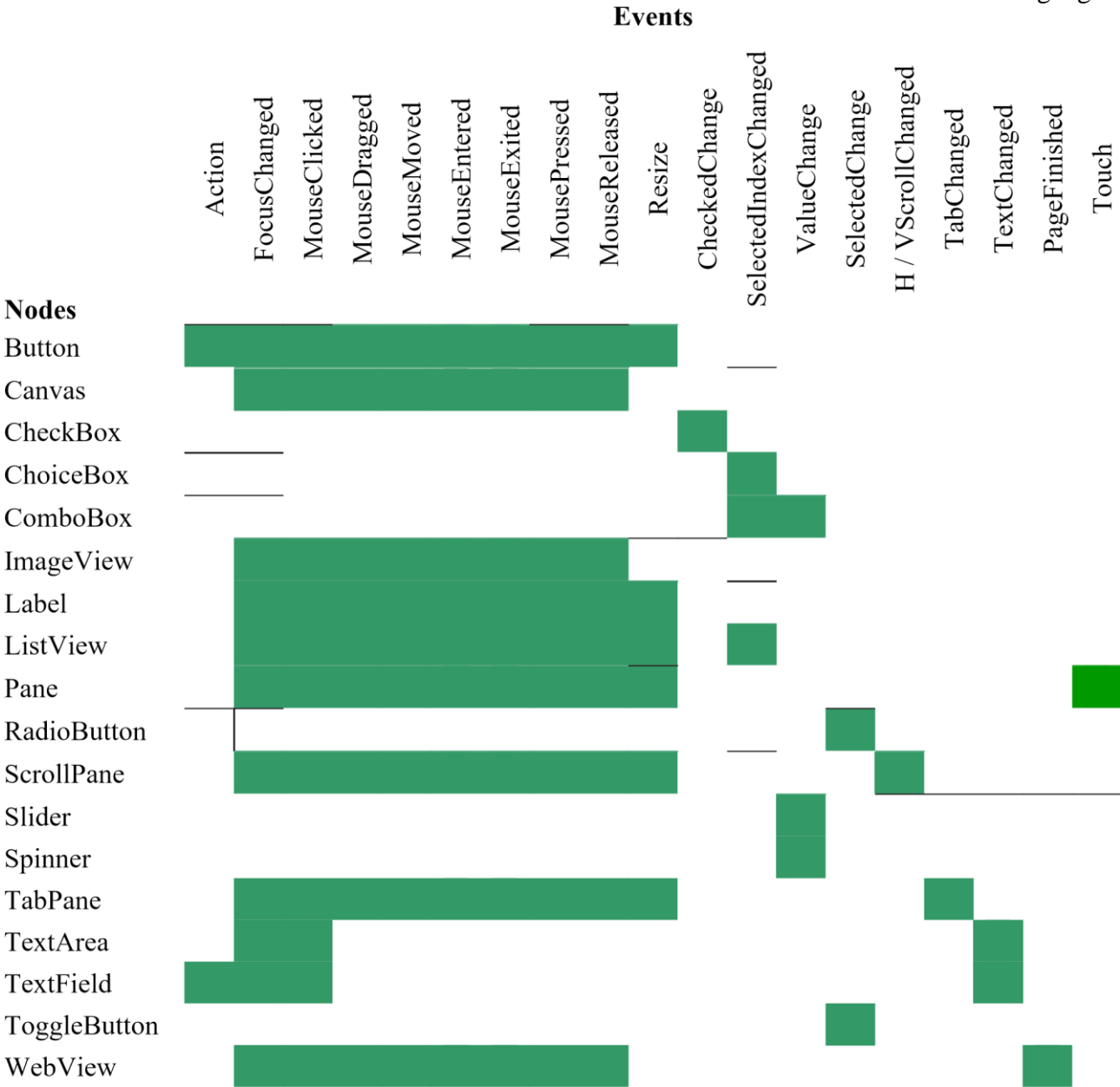
Example:

```
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
```

5.8.3 B4J

User interface objects are called 'Nodes' in Java.

Summary of the events for different nodes:



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 loses 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 drags 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
```
- **MouseEntered** (EventData As MouseEvent)
Event raised when the user enters the node.
Example:

```
Private Sub Pane1_MouseEntered (EventData As MouseEvent)
    ' Your code
End Sub
```
- **MouseExited** (EventData As MouseEvent)
Event raised when the user exits the node.
Example:

```
Private Sub Pane1_MouseExited (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.

- **Touch** (Action As Int, X As Float, Y As Float)

Event raised when the user 'touches' the screen.

This event is similar to the Touch events in B4A and B4i.

Three different actions are handled:

- Pane1.TOUCH_ACTION_DOWN, the user touches the screen.
- Pane1.TOUCH_ACTION_MOVE, the user moves the finger without leaving the screen. -
- Pane1.TOUCH_ACTION_UP, the user leaves the screen.

The X and Y coordinates of the mouse cursor position are given.

Example:

```
Sub Pane1_Touch (Action As Int, X As Float, Y As Float)
    Select Action
    Case Pane1.TOUCH_ACTION_DOWN
```

```
' Your code for DOWN action
Case Panel1.TOUCH_ACTION_MOVE
' Your code for MOVE action
Case Panel1.TOUCH_ACTION_UP
' Your code for UP action
End Select
End Sub
```

OR

```
Sub Panel1_Touch (Action As Int, X As Float, Y As Float)
  Select Action
  Case 0 'DOWN
    ' Your code for DOWN action
  Case 2 'MOVE
    ' Your code for MOVE action
  Case 1 'UP
    ' Your code for UP action
  End Select
End Sub
```

5.8.4 B4R

In B4R, the Pin and [Timer](#) objects are the only ones raising an event:

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

Example:

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

- Timer
Tick Event raised at every given interval

Example:

```
Private Timer1 As Timer

Timer1.Initialize("Timer1_Tick",1000)

Sub Timer1_Tick
    ' Your code
End Sub
```

Be aware that in B4R the initialize method is different from the other B4X products. You must declare the full sub name like "Timer1_Tick", and not "Timer1" like in the other products.

.5 User interface summary

5.8.5 User interface summary

The ‘standard’ user interface objects.

This shows the difference between the three operating systems.

Some views / nodes which don’t exist as standard objects can exist 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



For cross-platform projects you might look at the [B4X Cross-platform projects](#) booklet and more specific [chapter 4. Compatibilities B4A B4i B4J XUI](#).

5.9 Libraries

Libraries add more objects and functionalities to B4X.

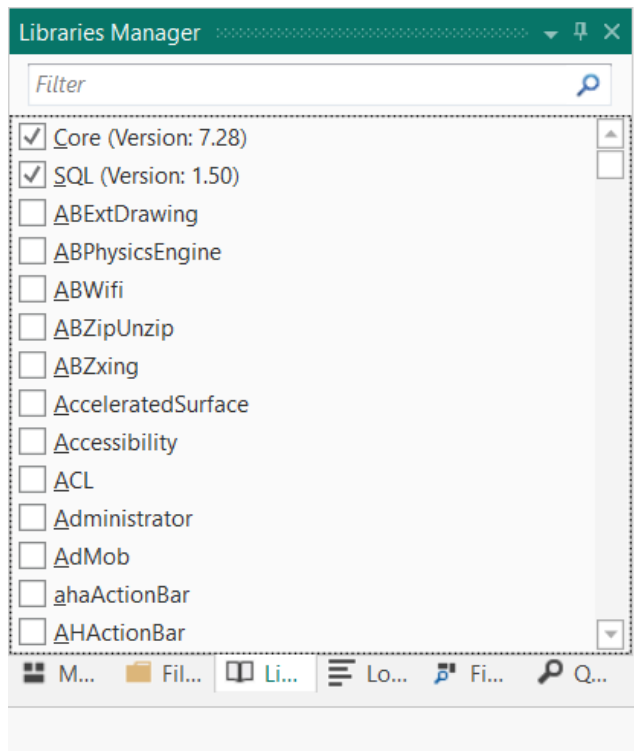
Some of these libraries are shipped with the B4X products and are part of the standard development system.

Other, often developed by users, can be downloaded (by registered users only) to add supplementary functionalities to the B4X development environments.

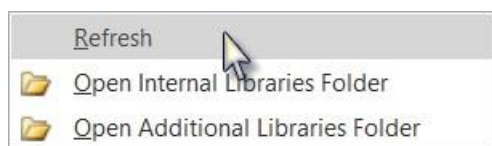
When you need a library, you have to:

- Check it in the Libs Tab, if you already have the library.

- For additional libraries, check if it's the latest version.
You can check the versions in the documentation page [B4A](#), [B4i](#), [B4J](#), [B4R](#)
Or in the [Libraries Google sheet](#) in the forum. 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 for the give product.
If it's a [B4XLibrary](#), copy the <LibraryName>.b4xlib file To AdditionalLibraries\B4X folder.
- Right click in the Lib area and click on **Refresh** and check the library in the list to select it.



5.9.1 Standard libraries

The standard B4X libraries are saved in the Libraries folder in the B4X program folder.

Normally in:

C:\Program Files\Anywhere Software\B4A\Libraries

C:\Program Files\Anywhere Software\B4i\Libraries

C:\Program Files\Anywhere Software\B4J\Libraries

C:\Program Files\Anywhere Software\B4R\Libraries

5.9.2 Additional libraries folder

Additional Libraries are composed of two files: an *xxx.jar* and an *xxx.xml* file.

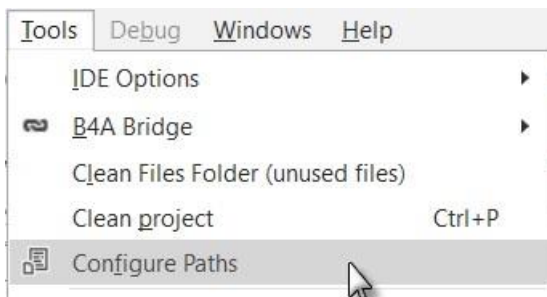
B4X libraries have only one file *xxx.b4xlib*.

For the additional libraries it is necessary to setup a special folder to save them somewhere else. This folder must have the following structure:

▼	AdditionalLibraries	
	B4A	Folder for B4A additional libraries.
	B4i	Folder for B4i additional libraries.
	B4J	Folder for B4J additional libraries.
>	B4R	Folder for B4R additional libraries.
	B4X	Folder for B4X libraries .
	B4XlibXMLFiles	Folder for B4X libraries XML files.

One subfolder for each product: B4A, B4i, B4J, B4R and another B4X for B4X libraries.

When you install a new version of a B4X product, 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 additional libraries folders.

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

In my system, I added a B4XlibXMLFiles folder for XML help files.

The standard and additional libraries have an XML file. B4X Libraries do not.

But, if you use the [B4X Help Viewer](#) you would be interested in having these help files if they are available. The B4X Help Viewer is explained in the [B4X Help tools booklet](#).

You can create xml files for b4xlib libraries with this tool: [b4xlib – XML generation](#).

5.9.2.1 Paths configuration B4A

A Paths Configuration

javac.exe
Usually found under C:\Program Files\Java\jdk1.8.x_xx\bin

SDK Manager Proxy Host: Proxy Port:

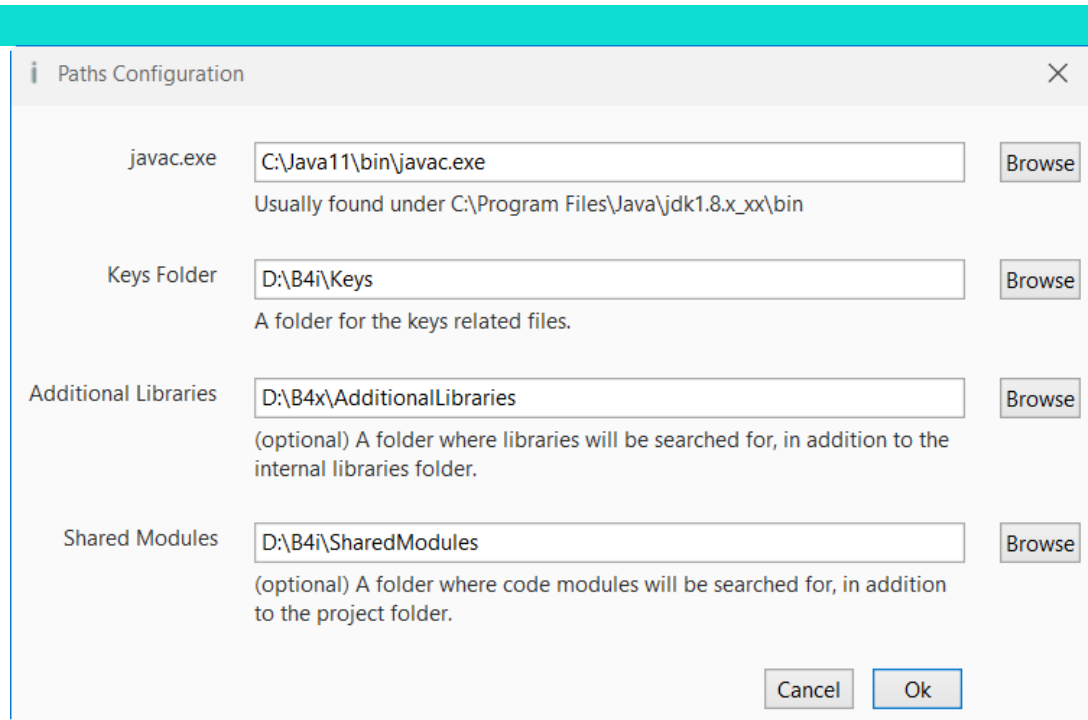
android.jar
Usually found under C:\android-sdk\platforms\android-x

Additional Libraries
(optional) A folder where libraries will be searched for, in addition to the internal libraries folder. Make sure NOT to set it to the B4A libraries folder.

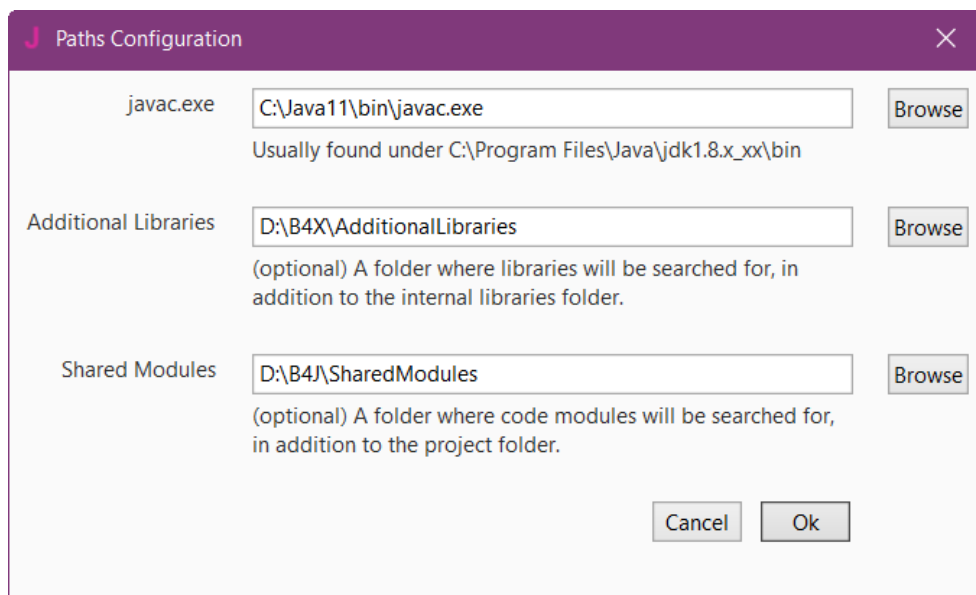
Shared Modules
(optional) A folder where code modules will be searched for, in addition to the project folder.

Enter the folder names and click on .

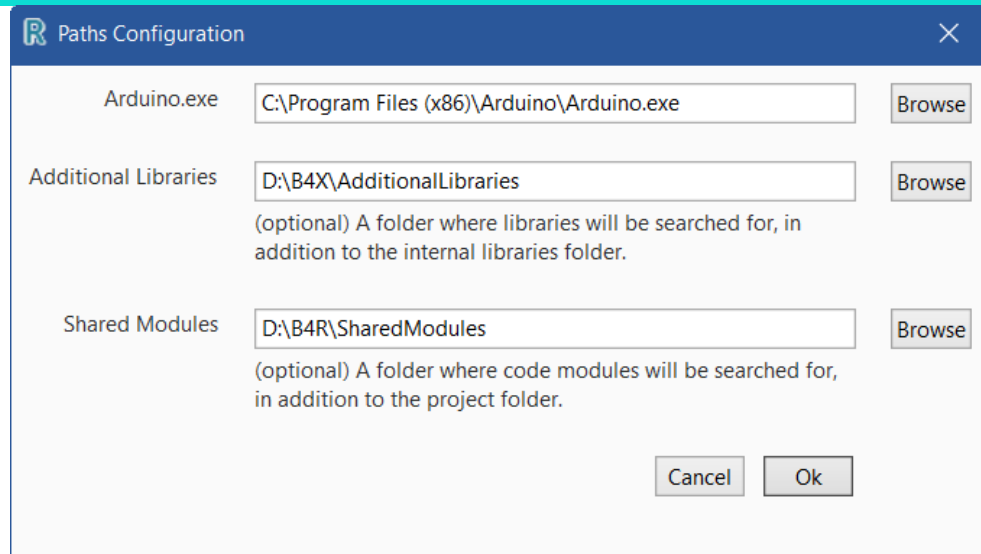
5.9.2.2 Paths configuration B4i



5.9.2.3 Paths configuration B4J



5.9.2.4 Paths configuration B4R



5.9.3 B4X Libraries *.b4xlib

B4X libraries are cross platform libraries introduced in B4A 8.80, B4i 5.50 and B4J 7.00.

These libraries contain cross platform classes which don't need to be compiled as libraries.

A B4X library is a simple zip file with the following structure:

- Code modules. All types are supported including Activities and Services.
- Files, including layout files.
- Optional manifest file with the following fields:
 - Version
 - Author
 - Depends On (list of required libraries), Supported Platforms
 - . Fields can be shared between the platforms or be platform specific.

Files and code modules can also be platform specific.

Creating a b4x library is very simple. You just need to create a zip file with these resources. The zip file extension should be b4xlib. That's all.

Note that the source code can be extracted from a b4x library.

b4x libraries appear like all other libraries in the Libraries tab.

Example: the AnotherDatePicker.b4xlib

The zip file structure:



Files contains all the needed files, the three layout files in the example.

AnotherDatePicker.bas is the crossplatform Custom View file.

Manifest.txt contains:

Version=2.00	version number.
B4J.DependsOn=jXUI, jDateUtils	libraries used for B4J.
B4A.DependsOn=XUI, DateUtils	libraries used for B4A.
B4i.DependsOn=iXUI, iDateUtils	libraries used for B4i.

Copy the xxx.b4xlib file to the AdditionalLibraries\B4X folder.

If there is an xxx.xml file, you must not save it there but in another folder.

B4XLibraries are explained in the [B4X Custom Views Booklet](#).

5.9.4 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](#).

B4i Documentation page: [List of Libraries](#).

B4J Documentation page: [List of Libraries](#).


B4R Documentation page: [List of Libraries](#).

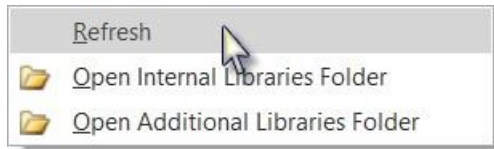
Or in the [B4X Libraries Google sheet](#).

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

To load or update a library follow the steps below:

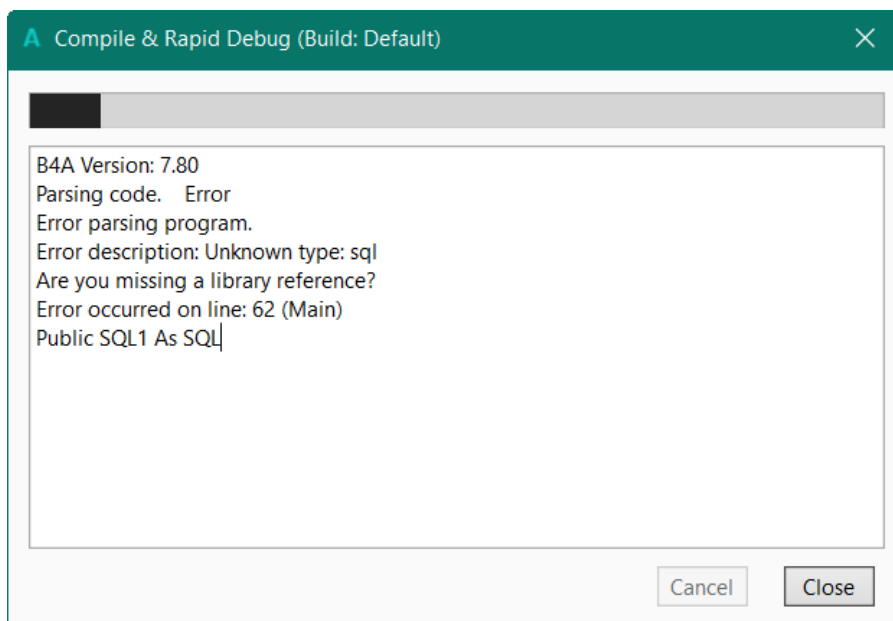
- Download the library zip file somewhere.

- Unzip it.
- Copy the xxx.jar and xxx.xml files to the
 - B4X Library folder for a standard B4X library
 - [Additional libraries folder](#) for an additional library.
- Right click in the libraries list in the [Lib Tab](#) and click on  and select the library.



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

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



5.10 String manipulation

5.10.1 B4A, B4i, B4J String

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")`

5.10.2 String concatenation

The concatenation character to join Strings is: `&`

Examples:

- Strings

```
Private MyString As String
MyString = "aaa" & "bbb" & "ccc"      result: aaabbbccc
```
- String and number `MyString = "$: " & 1.25` result: `$: 1.25`
- String and variable, it can be either another string or a number.

```
Private Val As Double
Val = 1.25
MyString = "$: " & Val                result: $: 1.25
```

Don't confuse with VB syntax:

```
MyString = "aaa" + "bbb" + "ccc"
```

This doesn't work!

5.10.3 B4A, B4i, B4J StringBuilder

StringBuilder is a mutable string, unlike regular strings which are immutable. StringBuilder is especially useful when you need to concatenate many strings.

The following code demonstrates the performance boosting of StringBuilder:

```
Dim start As Long start
= DateTime.Now
'Regular string
Dim s As String For
i = 1 To 5000 s =
s & i
Next
Log(DateTime.Now - start)
'StringBuilder start =
DateTime.Now Dim sb As
StringBuilder
sb.Initialize For i =
1 To 5000
sb.Append(i)
Next
Log(DateTime.Now - start)
```

Tested on a real device, the first 'for loop' took about 20 seconds and the second took less than a tenth of a second.

The reason is that the code: `s = s & i` creates a new string each iteration (strings are immutable). The method `StringBuilder.ToString` converts the object to a string.

5.10.3.1 StringBuilder Methods

Append (Text As String) As StringBuilder

Appends the specified text at the end.

Returns the same object, so you can chain methods. Example:

```
sb.Append("First line").Append(CRLF).Append("Second line")
```

Initialize

Initializes the object.

Example:

```
Dim sb As StringBuilder sb.Initialize
sb.Append("The value is: ").Append(SomeOtherVariable).Append(CRLF)
```

Insert (Offset As Int, Text As String) As StringBuilder

Inserts the specified text at the specified offset.

IsInitialized As Boolean

Length As Int [read only]

Returns the number of characters.

Remove (StartOffset As Int, EndOffset As Int) As StringBuilder

Removes the specified characters.

StartOffset - The first character to remove.

EndOffset - The ending index. This character will not be removed.

ToString As String

Converts the object to a string.

5.10.4 Smart String Literal

The "smart string" literal is a more powerful version of the standard string literal.

It has three advantages:

1. Supports multi-line strings.
2. No need to escape quotes.
3. Supports string interpolation.

The smart string literal starts with "\$" and ends with "\$".

Example:

```
Dim s As String = $"Hello world"$
Dim query As String = $"
SELECT value_id FROM table3
WHERE rowid >= random()%(SELECT max(rowid)FROM table3)
AND second_value ISNOTNULL
LIMIT 1"$
Log($"No need to escape "quotes"! "$)
```

5.10.4.1 String Interpolation

Smart strings can hold zero or more placeholders with code. The placeholders can be easily formatted.

A placeholder starts with `[$[optional formatter]{` and ends with `}`:

```
Log($"5 * 3 = ${5 * 3}"$) '5 * 3 = 15
```

You can put any code you like inside the placeholders.

```
Dim x = 1, y = 2, z = 4 As Int
Log($"x = ${x}, y = ${y}, z = ${Sin(z)}"$) 'x = 1, y = 2, z = -0.7568024953079282
```

This is a compile time feature. You cannot load the strings from a file for example.

5.10.4.2 Number Formatter

The number formatter allows you to set the minimum number of integers and the maximum number of fractions digits. It is similar to `NumberFormat` keyword.

The number formatter structure: `MinIntegers.MaxFractions`. `MaxFractions` component is optional. Examples:

```
Dim h = 2, m = 15, s = 7 As Int
Log($"Remaining time $2{h}:$2{m}:$2{s}"$) 'Remaining time 02:15:07
Log($"10 / 7 = $0.3{10 / 7}"$) '10 / 7 = 1.429
Log($"1.2{"The value is not a number!"}"$) 'NaN
```

5.10.4.3 Other Formatters

Note that the formatters are case insensitive.

Date - Equivalent to `DateTime.Date`:

```
Log($"Current date is $date{DateTime.Now}"$) 'Current date is 02/02/2015
```

Time - Equivalent to `DateTime.Time`:

```
Log($"Current time is $time{DateTime.Now}"$) 'Current time is 11:17:45
```

DateTime - Equivalent to `DateTime.Date & " " & DateTime.Time`:

```
Log($"Current time is $DateTime{DateTime.Now}"$) 'Current time is 02/02/2015 11:18:36
```

XML - Escapes the five XML entities (`"`, `'`, `<`, `>`, `&`):

```
Dim UserString As String = $"will it break your parser ><'&?"$  
Log($"User input is: $xml{UserString}$") 'User input is: will it  
break your parser &gt;&lt;&#39;&quot;&amp;?
```

This is also useful for html content.

5.10.5 B4A, B4i CharSequence CSBuilder

CharSequence is a native interface in Android SDK.

A String is one implementation of CharSequence.

There are other implementations of CharSequence that provide more features and allow us to format the string, add images and even make parts of the text clickable.

Starting from B4A v6.80 many methods accept CharSequence instead of String. Existing code will work properly as you can pass regular strings. However you can now also pass more interesting CharSequences.

Note to library developers, if your library makes calls to APIs that work with CharSequences then you should change your method signatures to expect CharSequence instead of String. This will allow developers to format the text.

This tutorial covers the CSBuilder object.

CSBuilder is similar to StringBuilder. Instead of building strings, it builds CharSequences that include style information.

The examples are made with B4A, but the principles are the same for B4i

Using it is quite simple.

5.10.5.1 Text

```
Private cs As CSBuilder
cs = cs.Initialize.Color(Colors.Red).Append("Hello World!").PopAll
Label1.Text = cs
```




The default background color can be different depending on the Android version.

Almost all methods of CSBuilder return the object itself. This allows us to chain the method calls. Text is always appended with the Append method.

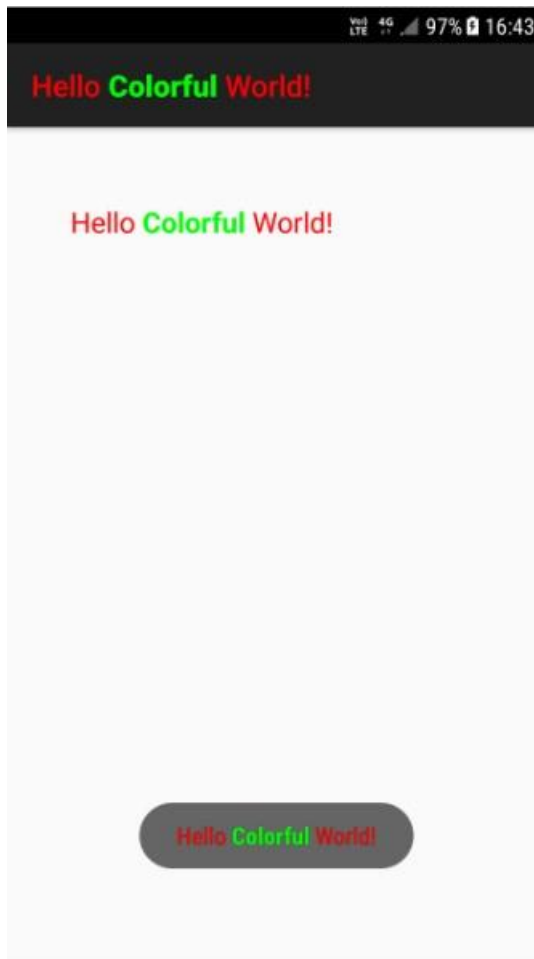
There are various attributes that can be set. Setting an attribute marks the beginning of a style span. Calling Pop ends the last span that was added (and not ended yet).

Calling PopAll ends all open spans. It is convenient to always call PopAll at the end to ensure that all spans are closed.

```
'example of explicitly popping an attribute:
Label1.Text = cs.Initialize.Color(Colors.Red).Append("Hello
").Pop.Append("World!").PopAll
```

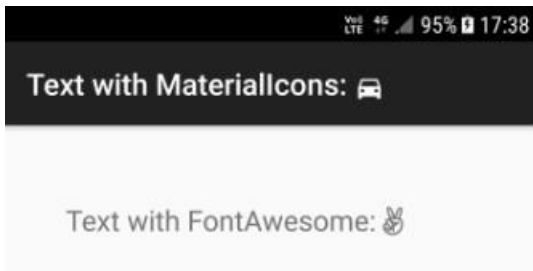


```
'It doesn't matter whether the methods are chained or split into several lines: Private
cs As CSBuilder
cs.Initialize.Color(Colors.Red).Append("Hello ")
cs.Bold.Color(Colors.Green).Append("Colorful ").Pop.Pop
'two pops: the first removes the green color and the second removes the bold style
cs.Append("World!").PopAll
Label1.Text = cs
'can also be set as the activity title
Activity.Title = cs
'and Toast messages and in other places...
ToastMessageShow(cs, True)
```



5.10.5.2 With FontAwesome or MaterialIcons

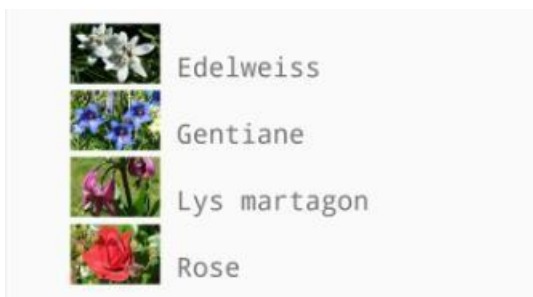
```
Private cs As CSBuilder
Label1.Text = cs.Initialize.Append("Text with FontAwesome:
").Typeface(Typeface.FONTAWESOME).Append(Chr(0xF209)).PopAll
'Using the same builder multiple times. Note that it is initialized each time.
'Note that we vertically align the material icon character. cs.Initialize.Append("Text
with MaterialIcons:
").Typeface(Typeface.MATERIALICONS).VerticalAlign(5dip).Append(Chr(0xE531)).PopAll
Activity.Title = cs
```



Note: The hex values of Materialicons characters begin with 0xE and FontAwesome characters begin with 0xF

5.10.5.3 Images

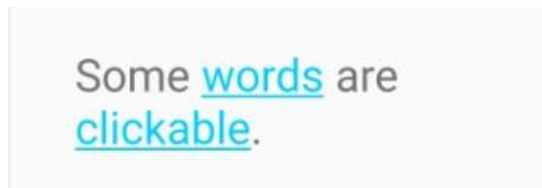
```
Private cs As CSBuilder
cs.Initialize.Size(18).Typeface(Typeface.MONOSPACE)
cs.Image(LoadBitmap(File.DirAssets, "edelweiss.jpg"), 60dip, 40dip, False).Append("
Edelweiss").Append(CRLF)
cs.Image(LoadBitmap(File.DirAssets, "gentiane.jpg"), 60dip, 40dip, False).Append("
Gentiane").Append(CRLF)
cs.Image(LoadBitmap(File.DirAssets, "lys_martagon.jpg"), 60dip, 40dip, False).Append("
Lys martagon").Append(CRLF)
cs.Image(LoadBitmap(File.DirAssets, "rose.jpg"), 60dip, 40dip, False).Append("
Rose").Append(CRLF)
cs.PopAll Label1.Text
= cs
```



5.10.5.4 Clickable text

The Clickable method creates clickable text. For the event to be raised you must call `cs.EnableClickEvents`.

The Append method accepts a CharSequence. In the following code the CreateClickableWord sub returns a CharSequence that is then appended to the other CharSequence.



5.10.5.5 Highlight text

Example from the [SearchView](#) class.

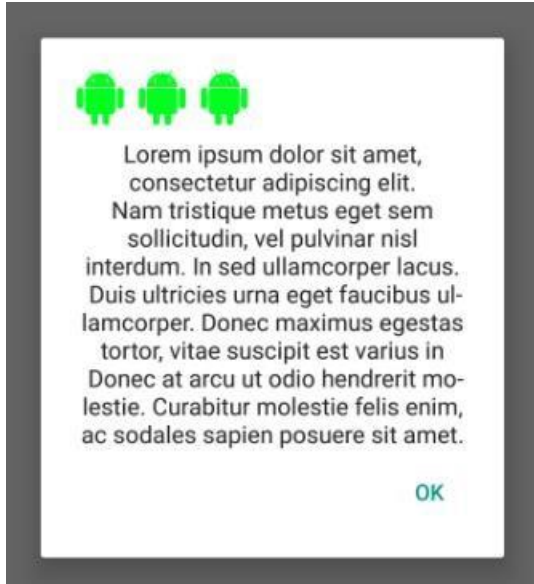
```
Private Sub AddItemsToList(li As List, full As String)
  If li.IsInitialized = False Then Return
  Dim cs As CSBuilder
  For i = 0 To li.Size - 1
    Dim item As String = li.Get(i)
    Dim x As Int = item.ToLowerCase.IndexOf(full)
    If x = -1 Then
      Continue End
    If
      cs.Initialize.Append(item.SubString2(0,
x)).Color(highlightColor).Append(item.SubString2(x, x + full.Length)).Pop
      cs.Append(item.SubString(x + full.Length))
      lv.AddSingleLine(cs)
    Next
  End Sub
```



5.10.5.6 Center aligned text

```
Msgbox(cs.Initialize.Alignment("ALIGN_CENTER").Append($"Lorem ipsum dolor sit am
et, consectetur adipiscing elit.
Nam tristique metus eget sem sollicitudin, vel pulvinar nisl interdum. In sed ul
lamcorper lacus.
```

```
Duis ultricies urna eget faucibus ullamcorper. Donec maximus egestas tortor, vit
ae suscipit est varius in
Donec at arcu ut odio hendrerit molestie. Curabitur molestie felis enim, ac soda
les sapien posuere sit amet."$).PopAll, _
cs.Initialize.Typeface(Typeface.FONTAWESOME).Color(0xFF01FF20).Size(40).Append(C
hr(0xF17B) & " " & Chr(0xF17B) & " " & Chr(0xF17B)).PopAll)
```



5.10.5.7 CSBuilder Methods

5.10.5.7.1 B4A / B4i

- **Alignment** (Alignment As Alignment Enum) Starts an alignment span.
Alignment - One of the following strings:
ALIGN_NORMAL, ALIGN_OPPOSITE or ALIGN_CENTER
- **Append** (Text As CharSequence)
Appends the provided String or CharSequence.
- **BackgroundColor** (Color As Int) Starts a background color span.
- **Color** (Color As Int)
Starts a foreground color span.
- **Initialize**
Initializes the builder. You can call this method multiple times to create new CharSequences.
Note that like most other methods it returns the current object.
- **IsInitialized**
Tests whether this object was initialized. Returns a Boolean.
- **Pop**
Closes the most recent span. All spans must be closed. You can call PopAll to close all open spans.

- **PopAll**
Closes all open spans.
It is convenient to always call PopAll at the end to ensure that all spans are closed.
- **Strikethrough**
Starts a strikethrough span.
- **ToString**
Returns a string with the characters.
- **Underline**
Starts an underline span.
- **VerticalAlign** (Shift As Int)
Starts a vertical alignment span (positive = downwards).

5.10.5.7.2 B4A only

- **Bold**
Starts a bold span.
- **Clickable** (EventName As String, Tag As Object)
Starts a clickable span. For the event to be raised you need to call the EnableClickEvents method. Example:

```

Sub Activity_Create(FirstTime As Boolean)
    Activity.LoadLayout("1")
    Dim cs As CSBuilder
    cs.Initialize.Size(30).Append("Some ").Append(CreateClickableWord("words"))
    cs.Append(" are ").Append(CreateClickableWord("clickable")).Append(".").PopAll
    Label1.Text = cs
    cs.EnableClickEvents(Label1)
End Sub

Sub CreateClickableWord(Text As String) As CSBuilder
    Dim cs As CSBuilder
    Return cs.Initialize.Underline.Color(0xFF00D0FF).Clickable("word", Text).Append(Text).PopAll
End Sub

Sub Word_Click (Tag As Object)
    Log($"You have clicked on word: ${Tag}$")
End Sub

```
- **EnableClickEvents** (Label As TextView)
This method should be called when using clickable spans.
- **Image** (Bitmap As Bitmap, Width As Int, Height As Int, Baseline As Boolean)
Adds an image span. This method will add a space character as a placeholder for the image. Unlike the other methods you do not need to call Pop to close this span as it is closed automatically.
Bitmap - The image.

Width / Height - Image dimensions, use 'dip' units.

Baseline - If true then the image will be aligned based on the baseline. Otherwise it will be aligned based on the lowest descender in the text.

- **RelativeSize** (Proportion As Float)
Starts a relative size span. The actual text size will be multiplied with the set Proportion.
- **ScaleX** (Proportion As Float)
Starts a scale X span. It horizontally scales the text.
- **Size** (Size As Int)
Starts a text size span. Note that you should not use 'dip' units with text size dimensions.
- **TypeFace** (Typeface As Typeface) Starts a custom typeface span.
Similar to Font for B4i.

5.10.5.7.3 B4i only

- **Font** (Font As B4IFontWrapper)
Starts a font span.
Note that when AutoScaleAll is called the font is reset.
You should change the font in the parent Resize event or remove the call to AutoScaleAll from the layout designer script.
Similar to TypeFace for B4A.
- **KerningScale** (Scale As Float)
Sets the kerning (horizontal spacing) scale.
- **Link** (URL As NSString)
Creates a link. Links will be clickable in non-editable TextViews.

5.10.6 B4J TextFlow class

The [TextFlow Class](#) uses JavaObject to create a TextFlow node. With a TextFlow you can display rich text with different colors, fonts and other attributes.

Usage:

- Add the TextFlow class module to your project (Tools - Add Existing Module).
- Create a TextFlow object.
- Call AddText to add a text section and set its attributes.
- Eventually you should call CreateTextFlow to create the node that will be added to the layout.

Note that the set attributes return the class instance which allows chaining the calls.

Example code:

```
Dim tf As TextFlow tf.Initialize
tf.AddText("1 2 3").SetColor(fx.Colors.Red).SetUnderline(True)
tf.AddText(" 4 5 6 ").SetColor(fx.Colors.Green).SetFont(fx.CreateFont("", 17, True, True))
tf.AddText("7 8 9").SetColor(fx.Colors.Blue).SetStrikethrough(True).SetFont(fx.DefaultFont(20))
Dim pane As Pane = tf.CreateTextFlow
MainForm.RootPane.AddNode(pane, 10, 10, 200, 100)
```

5.10.7 B4R

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

These kind of manipulations can be done with the ByteConverter object in the rRandomAccessFile 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 ByteConverter object in the rRandomAccessFile library:

- **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.

5.11 Number formatting

5.11 Number formatting**5.11.1 B4A, B4i, B4J**

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

- **NumberFormat**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int)
`NumberFormat(12345.6789, 0, 2) = 12,345.68` `NumberFormat(1, 3, 0) = 001`
`NumberFormat(Value, 3, 0)` variables can be used.
`NumberFormat(Value + 10, 3, 0)` arithmetic operations can be used.
`NumberFormat((lblscore.Text + 10), 0, 0)` if one variable is a string add parentheses.
- **NumberFormat2**(Number As Double, MinimumIntegers As Int, MaximumFractions As Int, MinimumFractions As Int, GroupingUsed As Boolean)
`NumberFormat2(12345.67, 0, 3, 3, True) = 12,345.670`
`NumberFormat2(12345.67, 0, 3, 3, False) = 12345.670`

5.11.2 B4X NumberFormatter

[B4XFormatter](#) is an alternative to `NumberFormat` / `NumberFormat2` keywords. It is implemented in B4X as a b4xlib and it is cross platform.

There are two types in the library:

B4XFormatter - The main class.

B4XFormatData - A type with various configurable fields.

The formatter holds a list of format data objects. A new formatter starts with a single format data which acts as the default format.

5.11.3 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.

5.12 Timers

5.12 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
```

5.12 Timers

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

B4A: Activity_Create routine

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

B4i: Application_Start routine

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

B4J: AppStart routine

```
Sub AppStart (Form1 As Form, Args() As String)
  Timer1.Initialize("Timer1_Tick", 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
```


5.13 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.

5.13.1 File object

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

5.13.1.1 File 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.

5.13.1.1.1 B4X

To save data generated by the application and used only by the application you might use the `xui`, (`jxui` or `ixui`) library get the default folder.

xui.DefaultFolder

This folder is the same as:

- B4A - Same as `File.DirInternal`.
- B4i - Same as `File.DirDocuments`.
- B4J - Same as `File.DirData`.

You must first call `SetDataFolder` once before you can use this folder.

xui.SetDataFolder(AppName As String)

5.13.1.1.2 B4A only

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 Use this folder only if you really need it.

The storage card root folder. In most cases this is an internal storage card and not an external SD card.

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**.

External storage.

You should use the `RuntimePermissions` library to get the best folder with:

```
MyFolder = RuntimePermissions.GetSafeDirDefaultExternal(SubFolder As String)
```

Returns the path to the app's default folder on the secondary storage device.

The path to `File.DirInternal` will be returned if there is no secondary storage available.

It is a better alternative to `File.DirDefaultExternal`.

On Android 4.4+ no permission is required to access this folder.

SubFolder - A sub folder that will be created for your app. Pass an empty string if not needed.

Access a file in external storage devices has become cumbersome in Android.

Erel has written a Class [ExternalStorage - Access SD cards and USB sticks](#) to 'simplify' the access.

Extract from Erel's thread:

Before we start:

1. External storage means a real sd card or a connected mass storage USB device.
2. It has nothing to do with `File.DirRootExternal` / `DirDefaultExternal` which actually point to an internal storage.
3. It has nothing to do with runtime permissions.
4. You can use `RuntimePermissions.GetAllSafeDirsExternal` to directly access a specific folder on the SD card.
5. The minimum version for this class is Android 5. It might work with Android 4.4 (change `minSdkVersion` if you like to try it).

Starting from Android 4.4 it is no longer possible to directly access external storages.

The only way to access these storages is through the Storage Access Framework (SAF), which is a quite complex and under-documented framework.

The ExternalStorage class makes it simpler to work with SAF.

Usage:

1. Call ExternalStorage.SelectDir. This will open a dialog that will allow the user to select the root folder. Once selected the uri of the root folder is stored and can be later used without requiring the user to select the folder again. Even after the device is booted.
2. Wait For the ExternalFolderAvailable event.
Now you can access the files under Storage.Root, including inside subfolders.
3. Files are represented as a custom type named ExternalFile.
4. The following operations are supported: ListFiles, Delete, CreateNewFile, FindFile, OpenInputStream and OpenOutputStream.

See the attached example.

Depends on: ContentResolver and JavaObject libraries.

Add:

#AdditionalJar: com.android.support:support-core-utils

5.13.1.1.3 B4i only

File.DirDocuments

The documents folder should only be used to store user generated content. It is possible to make this folder sharable through iTunes.

This folder is backed up by iTunes automatically.

File.DirLibrary

The place for any non-user generated persistent files. This folder is backed up by iTunes automatically.

You can create a subfolder named Caches. Files under that folder will not be backed up.

File.DirTemp

A temporary folder. Files in this folder are not backed up by iTunes and may be deleted from time to time.

B4i Methods to access external resources or share to external apps.

This thread in the forum shows some methods to share files:

[List of methods to access external resources or share to external apps.](#)

5.13.1.1.4 B4J only

File.DirApp

Returns the application folder.

File.DirData

Returns the path to a folder that is suitable for writing files.

On Windows, folders under Program Files are read-only. Therefore File.DirApp will be read-only as well.

This method returns the same path as File.DirApp on non-Windows computers.

On Windows it returns the path to the user data folder. For example:

C:\Users\[user name]\AppData\Roaming\[AppName]

File.DirTemp

Returns the temporary folder.

5.13.1.2 File exists ? B4A, B4i, B4J

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.

Note: File.Exists does not work with File.DirAssets !!!

5.13.1.3 Common methods B4A, B4i, B4J

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 separate line.

Note that if a value contains the new line character it will be saved over more than one line and when you read it, it will be read as multiple items.

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

- Reads a file and stores each line as an item in a list.

File.WriteMap (Dir As String, FileName As String, Map As Map)

- Takes a map object which holds pairs of key and value elements and stores it in a text file. The file format is known as Java Properties file: [.properties - Wikipedia, the free encyclopedia](#)
The file format is not too important unless the file is supposed to be edited manually. This format makes it easy to edit it manually.

One common usage of File.WriteMap is to save a map of "settings" to a file.

File.ReadMap (Dir As String, FileName As String) As Map

- Reads a properties file and returns its key/value pairs as a Map object. Note that the order of entries returned might be different than the original order.

File.WriteBytes (Dir As String, FileName As String, Data As Byte()) -

Writes the given text to a new file.

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

Reads the data from the given file.

Returns: Byte()

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.Copy2 (In As InputStream, Out As OutputStream)

- Copies all the available data from the input stream into the output stream. The input stream is automatically closed at the end.

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 given directory. Example:

```
Private List1 As List
List1 = File.ListFiles(File.DirInternal)
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.

File.MakeDir (Parent As String, Dir)

- Creates the given folder (creates all folders as needed). Example:

```
File.MakeDir(File.DirInternal, "music/90")
```


5.13.2 Filenames

B4X file names allow following characters : **a** to **z**, **A** to **Z**, **0** to **9** dot . underscore **_** and even following characters **+ - % &** Spaces and following characters *** ?** are not allowed.

Example : MyFile.txt

Note that B4X file names are case sensitive !
MyFile.txt is different from myfile.txt

5.13.3 Subfolders

You can define subfolders in B4X with.

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

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

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

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

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

Both possibilities work.

5.13.4 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.DirInternal, "Test.txt" , False))
```

Writer could be declared in Sub Globals.

TextWriter.Initialize2 (OutputStream As OutputStream , Encoding As String)

- Initializes a TextWriter object as an output stream.

- Encoding indicates the CodePage (also called CharSet) for text encoding (see next chapter).

Example:

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

Writer could be declared in Sub Globals.

See : [Text encoding](#)

TextWriter.Write (Text As String)

- Writes the given Text to the stream.

TextWriter.WriteLine (Text As String)

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

TextWriter.WriteLineList (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.DirInternal, "Text.txt", False))
Writer.WriteLine("This is the first line")
Writer.WriteLine("This is the second line")
Writer.Close
```

5.13.5 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 As TextReader
Reader.Initialize(File.OpenInput(File.DirInternal, "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 CharSet), the text encoding.

Example:

```
Private Reader As TextReader
Reader.Initialize2(File.OpenInput(File.DirInternal, "Test.txt", "ISO-8859-1"))
```


Reader could be declared in Sub Globals.

See : [Text encoding](#)

TextReader.ReadAll As String

- Reads all of the remaining text and closes the stream.

Example:

```
txt = Reader.ReadAll
```

TextReader.ReadLine As String -

Reads the next line from the stream.

The new line characters are not returned.

Returns Null if there are no more characters to read.

Example:

```
Private Reader As TextReader
Reader.Initialize(File.OpenInput(File.DirInternal, "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
```

5.13.6 Text encoding

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

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

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

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

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

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* characterset.

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)
```

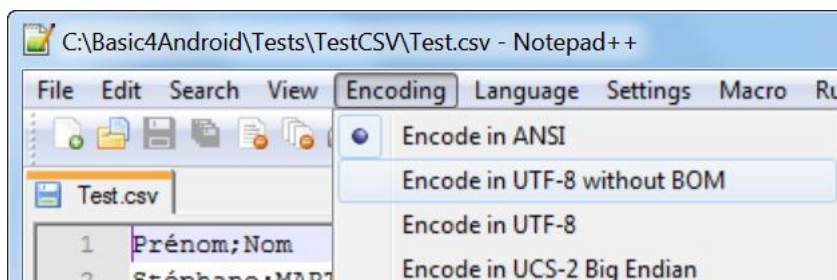
When you save a file with *NotePad* three additional bytes are added .

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")
```

5.14 Lists B4A, B4i and B4J only

5.14 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)
```

5.14 Lists B4A, B4i and B4J only

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.WriteLine(Dir As String, FileName As String, List As List)`
`File.WriteLine(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`

5.15 Maps B4A, B4i and B4J only

5.15 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`

5.15 Maps B4A, B4i and B4J only

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)`

5.16 Class modules

In B4X, you can use three types of Class Modules:

- Standard Class modules standard classes
- B4XPages B4XPages
- CustomView Class Modules specialized for custom views

In this chapter we will see only Standard Class modules.

B4XPages are explained in the [B4XPages Cross-platform projects](#) booklet.

CustomView Class Modules are explained in the [B4X CustomViews](#) booklet.

5.16.1 Getting started

Classes definition from [Wikipedia](#):

In object-oriented programming, a class is a construct that is used to create instances of itself – referred to as class instances, class objects, instance objects or simply objects. A class defines constituent members which enable its instances to have state and behaviour. Data field members (member variables or instance variables) enable a class instance to maintain state. Other kinds of members, especially methods, enable the behaviour of a class instances. Classes define the type of their instances.

A class usually represents a noun, such as a person, place or thing, or something nominalized. For example, a "Banana" class would represent the properties and functionality of bananas in general. A single, particular banana would be an instance of the "Banana" class, an object of the type "Banana".

Let us start with an example, the source code: *SourceCode\Person* in the / Person folder.

In the Person module

```
'Class Person module
Sub Class_Globals
    Private FirstName, LastName As String
    Private BirthDate As Long
End Sub

Sub Initialize (aFirstName As String, aLastName As String, aBirthDate As Long)
    FirstName = aFirstName
    LastName = aLastName
    BirthDate = aBirthDate
End Sub

Public Sub GetName As String
    Return FirstName & " " & LastName
```


End Sub

```
Public Sub GetCurrentAge As Int
    Return GetAgeAt(DateTime.Now)
End Sub
```

```
Public Sub GetAgeAt(Date As Long) As Int
Private diff As Long diff = Date -
BirthDate
    Return Floor(diff / DateTime.TicksPerDay / 365)
End Sub
```

Main module.

```
Sub Activity_Create(FirstTime As Boolean)
    Private p As Person
    p.Initialize("John", "Doe", DateTime.Parse("05/12/1970"))
    Log(p.GetCurrentAge)
End Sub
```

I will start by explaining the differences between classes, code modules and types.

Similar to types, classes are templates. From this template, you can instantiate any number of objects.

The type fields are similar to the classes global variables. However, unlike types which only define the data structure, classes also define the behaviour. The behaviour is defined in the classes' subs.

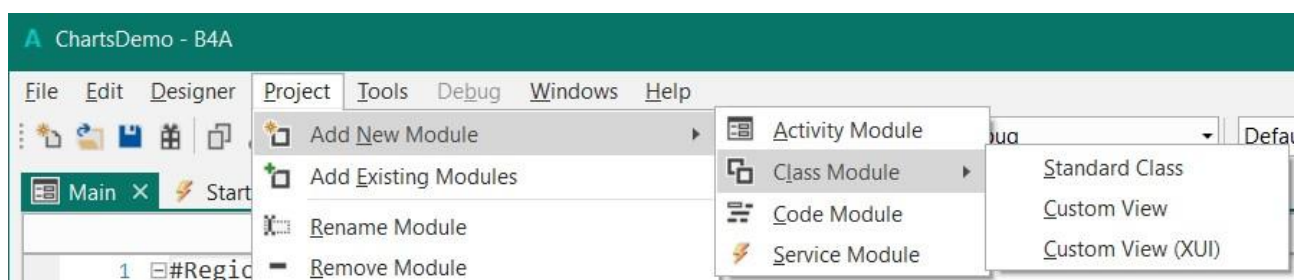
Unlike classes which are a template for objects, code modules are collections of subs. Another important difference between code modules and classes is that code modules always run in the context of the calling sub. The code module doesn't hold a reference to any context. For that reason, it is impossible to handle events or use CallSub with code modules.

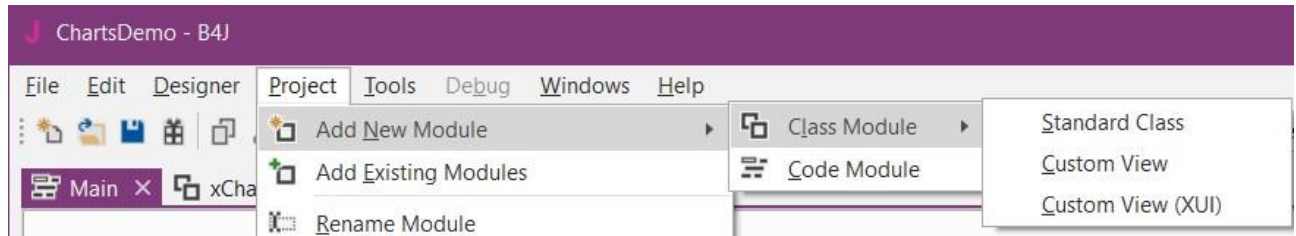
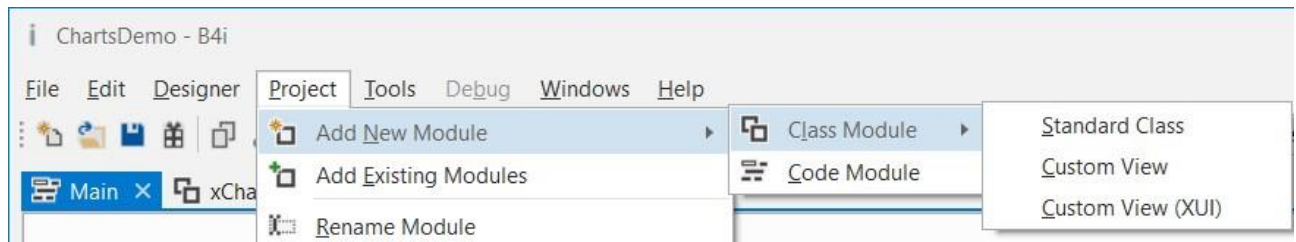
Classes store a reference to the context of the module that called the Initialize sub. This means that classes objects share the same life cycle as the module that initialized them.

5.16.1.1 Adding a Class module

Adding a new or existing class module is done by choosing Project > Add New Module > Class module or Add Existing module.

Like other modules, classes are saved as files with *bas* extension.



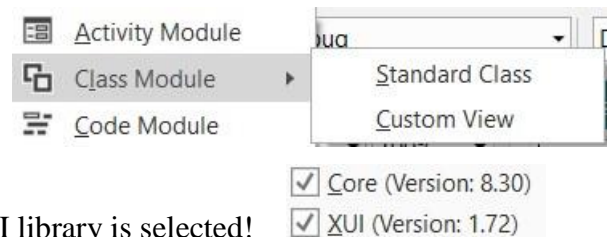


There are two class module types:

[Standard Class](#)

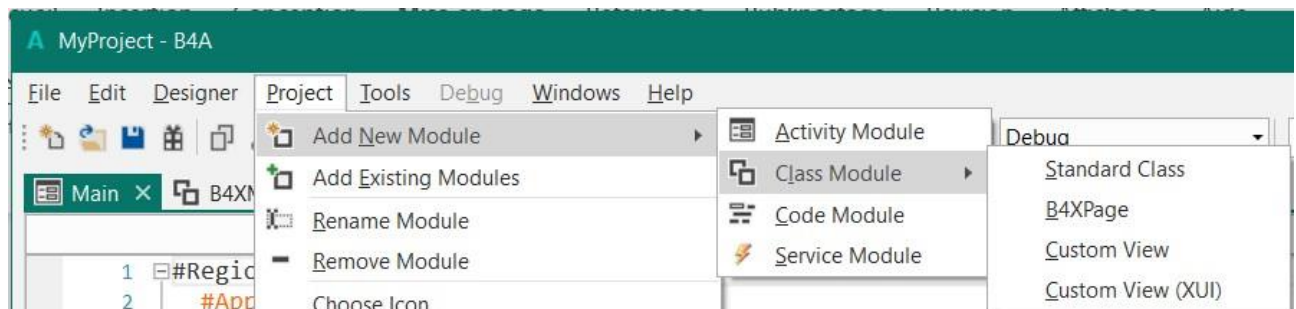
CustomView

CustomView (XUI)



The CustomView (XUI) is shown only when the XUI library is selected!

If you use the B4XPages template you can select B4XPage to create a B4XPage class.



5.16.1.2 Polymorphism

Polymorphism allows you to treat different types of objects that adhere to the same interface in the same way.

B4X polymorphism is similar to the [Duck typing](#) concept.

As an example we will create two classes named: Square and Circle.

Each class has a sub named Draw that draws the object to a canvas:

Source code *Draw* in the Draw folder.

The code below is the B4A code.

```
'Class Square module
Sub Class_Globals
    Private mx, my, mWidth As Int
```

```

End Sub
'Initializes the object. You can add parameters to this method if needed.
Sub Initialize (Shapes As List, x As Int, y As Int, length As Int) mx =
x my = y mLength = length
  Shapes.Add(Me)
End Sub

```

```

Sub Draw(c As Canvas)
  Private r As Rect
  r.Initialize(mx, my, mx + mLength, my + mLength)
  c.DrawRect(r, Colors.Red, False, 1dip)
End Sub

```

```

'Class Circle module
Sub Class_Globals
  Private mx, my, mRadius As Int
End Sub
'Initializes the object. You can add parameters to this method if needed.
Sub Initialize (Shapes As List, x As Int, y As Int, radius As Int) mx =
x my = y mRadius = radius
  Shapes.Add(Me)
End Sub

```

```

Sub Draw(cvs As Canvas) cvs.DrawCircle(mx, my, mRadius,
Colors.Blue, False, 1dip) End Sub

```

In the main module, we create a list `Shapes` with Squares and Circles. We then go over the list and draw all the objects:

```

Sub Process_Globals
  Public Shapes As List
End Sub

```

```

Sub Globals
  Private cvs As Canvas
End Sub

```

```

Sub Activity_Create(FirstTime As Boolean)
cvs.Initialize(Activity)
  Private Square1, Square 2 As Square
  Private Circle1 As Circle
  Shapes.Initialize
  Square1.Initialize(Shapes, 110dip, 110dip, 50dip)
  Square2.Initialize(Shapes, 10dip, 10dip, 100dip)
  Circle1.Initialize(Shapes, 50%x, 50%y, 100dip)

```

```

  DrawAllShapes
End Sub

```

```

Sub DrawAllShapes
  For i = 0 To Shapes.Size - 1
    CallSub2(Shapes.Get(i), "Draw", cvs)
  Next

```

```

    Activity.Invalidate
End Sub

```

As you can see, we do not know the specific type of each object in the list. We just assume that it has a Draw method that expects a single Canvas argument. Later we can easily add more types of shapes.

You can use the SubExists keyword to check whether an object includes a specific sub.

You can also use the IS keyword to check if an object is of a specific type.

5.16.1.3 Self-reference

The Me keyword returns a reference to the current object. Me keyword can only be used inside a class module.

Consider the above example. We have passed the Shapes list to the Initialize sub and then add each object to the list from the Initialize sub:

```

Sub Initialize (Shapes As List, x As Int, y As Int, radius As Int)
    mx = x    my = y    mRadius = radius
    Shapes.Add(Me)
End Sub

```

5.16.1.4 Activity object B4A only

This point is related to the Android Activities special life cycle.

Make sure to first read the [activities and processes life-cycle tutorial](#).

Android UI elements hold a reference to the parent activity. As the OS is allowed to kill background activities in order to free memory, UI elements cannot be declared as process global variables (these variables live as long as the process lives). Such elements are named Activity objects. The same is true for custom classes. If one or more of the class global variables is of a UI type (or any activity object type) then the class will be treated as an "activity object". The meaning is that instances of this class cannot be declared as process global variables.

5.16.2 Standard Class module

5.16.2.1 Structure

Default template of a standard class:

B4A and B4i

```

Sub Class_Globals

```

```
End Sub
'Initializes the object. You can add parameters to this method if needed. Public
Sub Initialize

End Sub
```

B4J

```
Sub Class_Globals
    Private fx As JFX
End Sub
'Initializes the object. You can add parameters to this method if needed. Public
Sub Initialize

End Sub
```

Only two routines are predefined:

Sub Class_Globals - This sub is similar to the Main Globals sub. These variables will be the class global variables (sometimes referred to instance variables or instance members). In B4J, the fx library library is declared by default. You can remove it if not needed.

Sub Initialize - A class object must be initialized before you can call any other sub. Initializing an object is done by calling the Initialize sub. When you call Initialize you set the object's context (the parent object or service).

Note that you can modify this sub signature and add arguments as needed.

Example: Person class module

The source codes are in the Person folder.

The code is the same for all three B4X platforms (B4A, B4i, B4J).

```
'Class Person module
Sub Class_Globals
    Private mFirstName, mLastName As String
    Private mBirthDate As Long
End Sub

Sub Initialize (FirstName As String, LastName As String, BirthDate As Long)
    mFirstName = FirstName    mLastName = LastName    mBirthDate = BirthDate
End Sub

Public Sub GetName As String
    Return mFirstName & " " & mLastName
End Sub

Public Sub GetCurrentAge As Int
    Return GetAgeAt(DateTime.Now)
End Sub
```

```
Public Sub GetAgeAt(Date As Long) As Int
    Dim diff As Long diff =
    Date - mBirthDate
    Return Floor(diff / DateTime.TicksPerDay / 365)
End Sub
```

In the above code, we created a class named Person and later instantiate an object of this type in the main module:

```
Private p As Person
p.Initialize("John", "Doe", DateTime.DateParse("05/12/1970"))
Log(p.GetCurrentAge)
```

Calling initialize is not required if the object itself was already initialized:

```
Private p2 As Person p2 = p 'both variables now point to
the same Person object.
Log(p2.GetCurrentAge)
```

6 "Code smells" code to be avoided

"Code smells" are common patterns that can indicate that there is a problem in the code. A problem doesn't mean that the code doesn't work, it might be that it will be difficult to maintain it or that there are more elegant ways to implement the same thing.

Remember that not everything is clear cut and there are exceptions for any rule.

6.1 Initializing an object and then assigning a different object to the same variable

```
'bad
Dim List1 As List
List1.Initialize '<-- a new list was created here
List1 = SomeOtherList '<--- previous list was replaced

'good
Dim List1 As List = SomeOtherList
```

6.2 Deprecated methods - DoEvents, MsgBox

These methods are deprecated, so you should not use these anymore.

More information here:

<https://www.b4x.com/android/forum/t...cated-and-async-dialogs-msgbox.79578/#content>

6.3 Deprecated methods - Map.GetKeyAt / GetValueAt

Deprecated methods - Map.GetKeyAt / GetValueAt - these methods were added before the For Each loop was available. They are not cross platform and are not the correct way to work with maps.

```
'bad
For i = 0 To Map1.Size - 1
    Dim key As String = Map1.GetKeyAt(i)
    Dim value As String = Map1.GetValueAt(i)
Next

'good
For Each key As String In Map1.Keys
    Dim value As String = Map1.Get(key)

Next
```

For database queries, use parametrized queries.

```
'very bad
SQL.ExecNonQuery("INSERT INTO table1 VALUES (' & EditText1.Text & '") 'ugly, will
break if there is an apostrophe in the text and vulnerable to SQL injections.
'very good
```

6.4 Not using parameterized queries

6.7 Repeating the code

```
SQL.ExecNonQuery2("INSERT INTO table1 VALUES (?)", Array(EditText1.Text))
```

6.5 Using Cursor instead of ResultSet - Cursor

For database queries, use ResultSet instead of Cursor.

Cursor is a B4A only object. ResultSet is a bit simpler to use and is cross platform.

```
'good
Dim rs As ResultSet = SQL.ExecQuery2(...) Do
While rs.NextRow
... Loop
rs.Close
```

6.6 Building the complete layout programmatically

Building the complete layout programmatically. This is especially a mistake in B4J and B4i because of the resize event and also if you want to build a cross platform solution. Layouts can be ported very easily.

There are many patterns to this one and all of them are bad.

```
'bad
If b = False Then
    Button1.Text = "disabled"
    Button2.Text = "disabled"
    Button3.Text = "disabled"
    Button1.Enabled = False
    Button2.Enabled = False
    Button3.Enabled = False
Else
    Button1.Text = "enabled"
    Button2.Text = "enabled"
    Button3.Text = "enabled"
    Button1.Enabled = True
    Button2.Enabled = True
    Button3.Enabled = True
End If

'good
For Each btn As Button In Array(Button1, Button2, Button3)
    btn.Enabled = b
    If b Then btn.Text = "enabled" Else btn.Text = "disable"
Next
```

6.8 Long strings without using smart strings

More information: <https://www.b4x.com/android/forum/threads/50135/#content>

6.11 Using code modules instead of classes

```
'bad
Dim s As String = "This is the " & QUOTE & "first" & QUOTE & "line" & CRLF & _
    "and this is the second one. The time is " & DateTime.Time(DateTime.Now) & "."
'good
Dim s As String = $"This is the "first" line
and this is the second one. The time is $Time{DateTime.Now}.$"
```

6.9 Using global variables when not needed

```
'bad
Job.Initialize(Me, "") 'global variable ...

'good
Dim job As HttpJob
job.Initialize(Me, "")
```

6.10 Not using Wait For when possible

Not using Wait For when possible. JobDone is a good example: [B4X] OkHttpUtils2 with Wait For
Code modules are very limited in B4A. In most cases you should use classes instead of code modules.
A code module is a single instance of a class.

6.12 Understanding booleans

```
'not elegant
Dim result As Boolean = DoingSomethingThatReturnTrueOrFalse
If result = True Then
    Return True
Else
    Return False
End If

' elegant
Return DoingSomethingThatReturnTrueOrFalse
```

6.13 Converting "random" bytes to strings

The only valid raw bytes that should be converted to a string, with BytesToString, are bytes that represent text. In all other cases it is a mistake to convert to string. Even if it seems to work it will later fail in other cases.

If you think that it is more complicated to work with raw bytes then you are not familiar with the useful B4XBytesBuilder object: <https://www.b4x.com/android/forum/threads/b4x-b4xcollectionsmore-collections.101071/#content>

7 Tips

These are Erel's tips for B4X developers ([\[B4X\] Tips for B4X developers](#)).

7.1 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")
```

7.2 Don't Repeat Yourself (DRY principle).

If you find yourself copying 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](#)).

7.3 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-usefulcollection.60304/>

7.4 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](#) as a good example. I wasn't familiar with this technology. When I started learning about it I was 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>

7.5 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.

7.6 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](#) or [Wait For](#).

7.7 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`.

7.8 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.

7.9 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>

7.10 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 `B4XSerializer`.

It is easy to implement powerful server solutions with B4J. Especially when the clients are implemented with B4A, B4i or B4J.

7.11 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.



7.12 Notepad++.

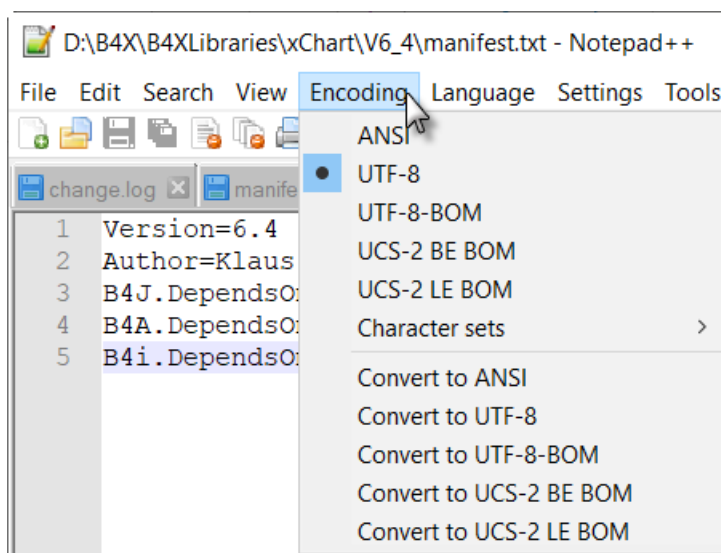
At one point or another we need to work with text files. I highly recommend all developers to use a good text editor that shows the encoding, the end of line characters and other important features.

<https://notepad-plus-plus.org/>

7.12.1 Encoding

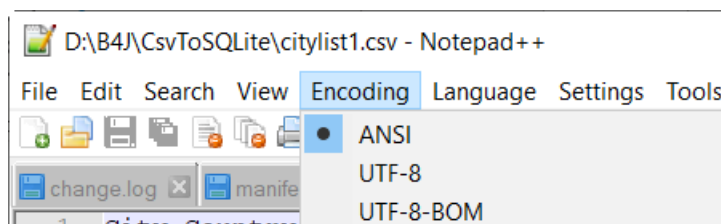
To show the current encoding of a text file, you can load it and then click in the menu on Encoding. The current encoding is checked.

You can select another encoding and save the file.

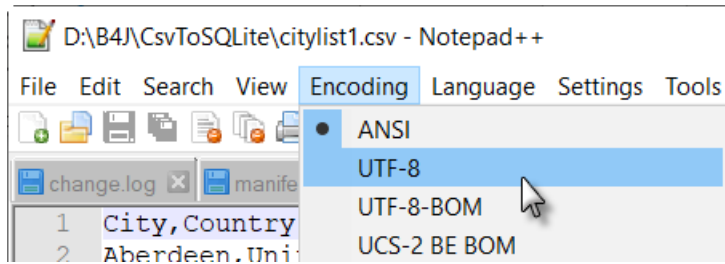


This can be useful when you have csv files generated with Excel, which are encoded with ANSI encoding, but, B4X uses UTF-8 encoding.

Original file:



Change the encoding and save the file with another file name.



When you reload this file and check the encoding, you will see this:

