



# User's Guide

Last update: 2016.02.28

Copyright: © 2012, 2013, 2014, 2015, 2016 Anywhere Software

Edition 1.7

1	General information.....	5
2	Conditional compilation .....	6
2.1	Build configurations .....	6
2.1.1	Built-in symbols.....	6
2.2	Code Exclusion.....	7
2.2.1	Example from the forum .....	7
2.3	Unsupported structure.....	10
3	Starter Service .....	11
4	Libraries .....	13
4.1	IME Input Methods Editor .....	13
4.1.1	Handling the screen size changed event.....	15
4.1.2	Showing and hiding the keyboard .....	16
4.1.3	Handle the action button .....	16
4.1.4	Custom filters .....	17
4.2	#AdditionalJar attribute.....	18
5	SQLite Database.....	22
5.1	SQLite Database basics.....	23
5.1.1	Database initialisation SQL1.Initialize .....	23
5.1.2	Table creation CREATE TABLE .....	23
5.1.3	INTEGER PRIMARY KEY rowID.....	24
5.1.4	Adding data INSERT INTO.....	25
5.1.5	Updating data UPDATE.....	25
5.1.6	Reading data SELECT .....	25
5.1.7	Filtering WHERE.....	27
5.1.8	Sorting ORDER BY.....	28
5.1.9	Date / Time functions.....	29
5.1.10	Other functions .....	30
5.1.11	Get Table information PRAGMA.....	31
5.1.12	Deleting data DELETE FROM.....	32
5.1.13	Rename a table ALTER TABLE Name ADD COLUMN .....	32
5.1.14	Add a column ALTER TABLE Name ADD COLUMN .....	32
5.1.14.1	Update the database after having added a column .....	32
5.1.15	Delete a table DROP TABLE.....	32
5.1.16	Insert an image.....	33
5.1.17	Read an image.....	33
5.1.18	ExecQuery vs ExecQuery2 / ExecNonQuery vs ExecNonQuery2.....	34
5.1.19	Insert many rows SQL.BeginTransaction / SQL.EndTransaction.....	35
5.1.20	Asynchronous queries .....	36
5.1.21	Batch inserts AddNonQueryToBatch / ExecNonQueryBatch.....	36
5.2	Multiple tables .....	37
5.3	Transaction speed .....	38
5.4	First steps .....	39
5.5	SQLite Viewer.....	41
5.6	SQLite Database first simple example program SQLiteLight1 .....	42
5.7	SQLite Database second simple example program SQLiteLight2.....	48
5.8	SQLite Database third simple example program SQLiteLight3 .....	54
5.9	SQLite Database forth example program SQLiteLight4 .....	55
5.10	SQLite Database fifth example program .....	57
5.10.1	Editing .....	59
5.10.2	Filtering .....	60
5.10.3	Code .....	61
5.10.3.1	Starter Service .....	62

5.10.3.2	Main Activity .....	62
5.10.3.3	Edit activity .....	64
5.10.3.4	Filter Activity .....	65
6	DBUtils .....	66
6.1	DBUtils functions .....	67
6.2	Examples .....	69
6.2.1	Example program Main module .....	69
6.2.2	Show the table in a WebView .....	71
6.2.3	Show FirstName and LastName in a ListView .....	75
6.2.4	Display database in Spinners .....	77
6.2.5	Edit database .....	79
7	GPS .....	87
7.1	GPS Library .....	87
7.1.1	GPS Object .....	87
7.1.2	GPS Satellite .....	88
7.1.3	GPS Location .....	88
7.1.4	NMEA data sentences .....	89
7.2	GPS Program .....	90
7.2.1	General explanations .....	93
7.2.2	Setup .....	94
7.2.3	GPS display .....	96
7.2.4	Satellites .....	97
7.2.5	Map display .....	98
7.2.6	GPS path .....	100
7.2.7	Save GPS path file / KML file .....	103
7.3	GPS Program Code .....	104
7.3.1	Initialization of the GPS .....	105
7.3.2	Button with tooltip .....	106
7.3.3	Button with tooltip and additional buttons .....	109
7.3.4	GPS Calculate distance scales .....	112
7.3.5	Drawing GPS position .....	113
8	Widgets, home screen widgets .....	116
8.1	Widgets Part I .....	116
8.2	Widgets Part II .....	119
9	okHttpUtils2 .....	122
9.1	okHttpUtils2 Objects .....	122
9.2	HttpUtils2 Functions .....	122
9.3	okHttpUtils Example1 .....	125
9.4	HttpUtils Example2 .....	126
9.5	The Flickr Viewer example .....	127
10	Network / AsyncStreams .....	128
11	Advanced drawings .....	132
11.1	View Drawables .....	132
11.1.1	ColorDrawable .....	132
11.1.2	GradientDrawable .....	133
11.1.3	BitmapDrawable .....	134
11.1.4	StateListDrawable .....	135
11.1.5	NinePatchDrawable .....	138
11.2	Layers with Panels / ImageViews / Images .....	140
11.2.1	Source code .....	141
11.3	Diagrams / Charts .....	145
11.3.1	Diagrams / Graph example program .....	145
11.3.2	Second Graph program .....	154

11.3.3	Charts Framework.....	155
11.3.3.1	Pie Chart .....	156
11.3.3.2	Bar Chart.....	159
11.3.3.3	Stacked Bar Chart.....	162
11.3.3.4	Lines Chart.....	164
11.4	Antialiasing filter .....	169
12	Class modules.....	170
12.1	Getting started .....	170
12.1.1	Adding a class module .....	171
12.1.2	Polymorphism.....	172
12.1.3	Self reference .....	173
12.1.4	Activity object.....	174
12.2	Standard class structure.....	175
12.3	Classes from the forum .....	177
12.4	Custom views .....	178
12.4.1	Custom view class structure .....	178
12.4.1.1	Event declaration.....	178
12.4.1.2	Designer properties declarations .....	179
12.4.1.3	Global variable declarations .....	179
12.4.1.4	Initialization routine .....	179
12.4.1.5	Designer support routine.....	180
12.4.1.6	Routine to get the base Panel.....	180
12.4.2	Adding a custom view by code.....	181
12.4.3	Add properties.....	182
12.4.4	Custom view in the Designer.....	183
12.5	First example LimitBar .....	186
12.6	Compile a class into a Library.....	192
12.6.1	Example with the LimitBar class example.....	194
12.6.2	Using the library in a program.....	196
12.7	Second example Wheel selection .....	197
12.7.1	Simple example.....	197
12.7.2	Show the selected entry centred in the middle window.....	201
12.7.3	Return the selected value.....	203
12.7.4	Color properties .....	205
12.7.5	A more advanced example .....	209

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

## 1 General information

This guide is dedicated for more advanced users and treats more specific topics.

It covers Basic4Android version 5.80.

All the source code and files needed (layouts, images etc) of the example projects in this guide are included in the SourceCode folder.

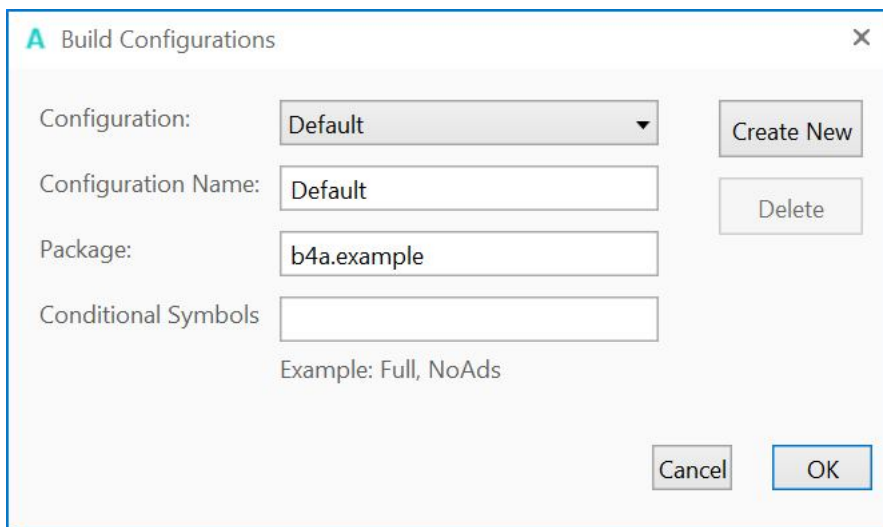
Beginners should first read the [Beginner's Guide](#).

## 2 Conditional compilation

In computer programming, conditional compilation is compilation implementing methods which allow the compiler to produce differences in the executable produced controlled by parameters that are provided during compilation.

### 2.1 Build configurations

The build configurations dialog is available in the IDE menu under Project -> Build Configurations (Ctrl + B).



This dialog allows you to edit or add new configurations and to choose the current active configuration.

A build configuration is made of a package name and a set of conditional symbols.

The package name replaces the previously global package field. This means that you can produce APKs with different package names from the same project. Note that multiple configurations can share the same package name.

The conditional symbols define the active compiler symbols. This allows you to exclude parts of the code based on the chosen build configuration.

You can set multiple symbols separated with commas.

#### 2.1.1 Built-in symbols

There are several built-in symbols:

- For B4A:  
B4A, DEBUG and RELEASE.  
Either DEBUG or RELEASE will be active based on the deployment mode.
- For B4J:  
B4J, DEBUG, RELEASE, UI or NON\_UI (based on the app type)

## 2.2 Code Exclusion

With the conditional compilation feature you can exclude any code you like from the **code editor**, **manifest editor** and **designer script** (any text can be excluded, including complete subs and attributes).

Excluded code will not be parsed and will be effectively removed before it reaches the compiler.

```

15 Sub Activity_Create (FirstTime As Boolean)
16 #If FULL
17     'this symbol is currently not active
18     'The code here is excluded.
19     kjsljkljklfj
20 #End If
21 |
22 #If TRIAL
23     'this symbol is active
24     Activity.LoadLayout ("Layout1")
25     '....
26 #End if
27 End Sub

```

The code exclusion syntax:

Each build configuration holds a set of symbols. Multiple configurations can share all or some of the symbols. This makes it possible to include or exclude code in several different configurations.

There is no support for #Else (it is related to the code editor lexer implementation). You can however achieve the same result by adding the same symbol to all other build configurations.

### 2.2.1 Example from the forum

The example code is in the ConditionalCompiling project in the SourceCode folder.

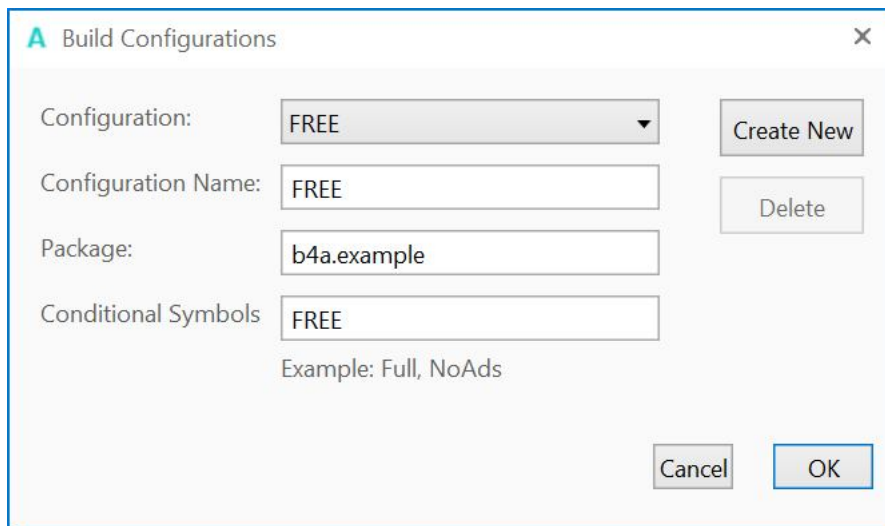
Let's create three configurations:

- FREE
- PAID
- AMAZON

In the IDE menu click on **Project / Build Configuration**.

Fill the in the fields.

Click on **OK** to confirm.



Build Configurations

Configuration: FREE

Configuration Name: FREE

Package: b4a.example

Conditional Symbols: FREE

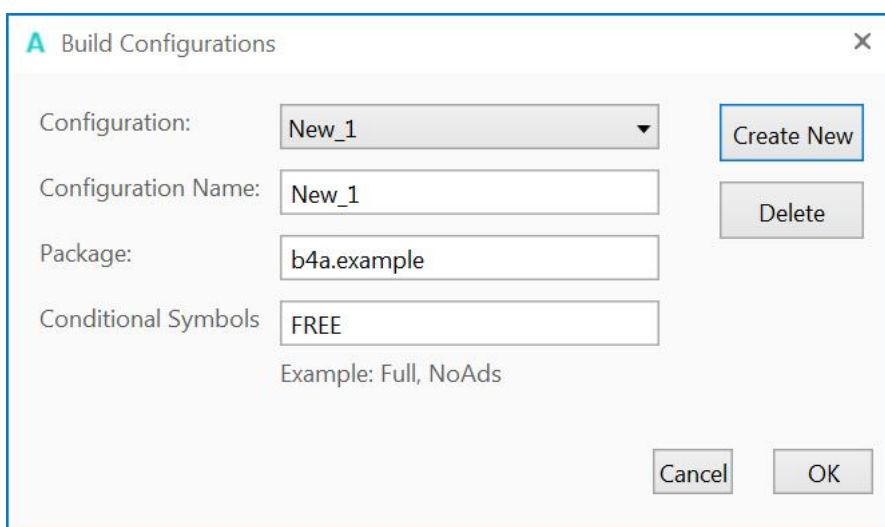
Example: Full, NoAds

Create New

Delete

Cancel OK

Select Project / Build Configuration again you'll see this.  
In the drop down list Default is replaced by FREE.



Build Configurations

Configuration: New\_1

Configuration Name: New\_1

Package: b4a.example

Conditional Symbols: FREE

Example: Full, NoAds

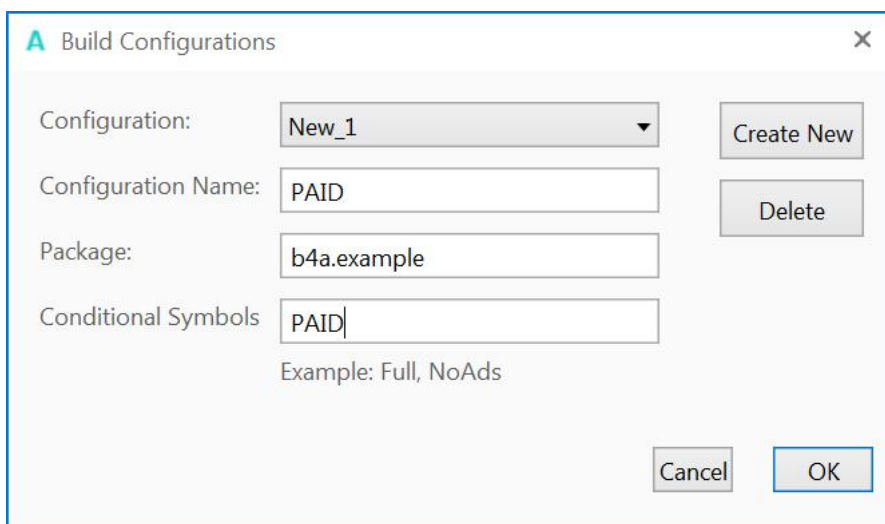
Create New

Delete

Cancel OK

Click on **Create New** to create a new configuration.

Fill in the new values.



Build Configurations

Configuration: New\_1

Configuration Name: PAID

Package: b4a.example

Conditional Symbols: PAID

Example: Full, NoAds

Create New

Delete

Cancel OK

Click on **Create New** to create the third configuration.  
Replace PAID by AMAZON and click **OK** to finish.

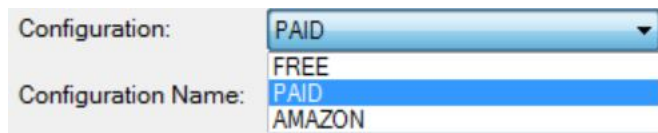


Enter the code below: We see that the code after IF AMAZON is colored because the last selected configuration is AMAZON.

### Sub Global s

```
#If FREE
    Dim sRelease As String = "(Free)"
    Dim sDist As String = ""
#End If
#If PAID
    Dim sRelease As String = "(Paid)"
    Dim sDist As String = ""
#End If
#If AMAZON
    Dim sRelease As String = "(Free)"
    Dim sDist As String = "Amazon"
#End If
End Sub
```

Now select PAID in the BuildConfigurations window:



We see now the three configurations in the list.

### Sub Global s

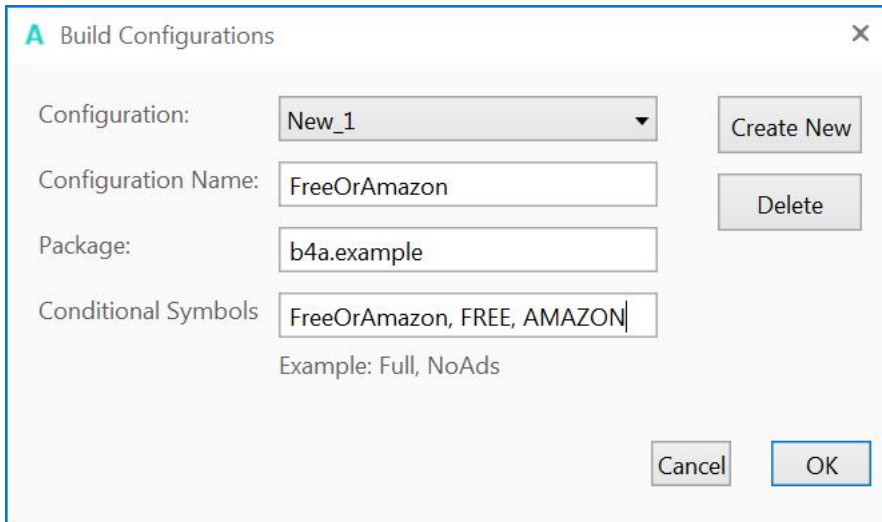
```
#If FREE
    Dim sRelease As String = "(Free)"
    Dim sDist As String = ""
#End If
#If PAID
    Dim sRelease As String = "(Paid)"
    Dim sDist As String = ""
#End If
#If AMAZON
    Dim sRelease As String = "(Free)"
    Dim sDist As String = "Amazon"
#End If
End Sub
```

Note that now the code after IF PAID is colored showing that this code will be executed and the other code will not be executed.

If you have a code like this, where the code for FREE and AMAZON is the same:

```
#If FREE
    #ApplicationLabel: Yahtzee! (Free)
    #VersionCode: 119
    #VersionName: 11.9
#End If
#If AMAZON
    #ApplicationLabel: Yahtzee! (Free)
    #VersionCode: 119
    #VersionName: 11.9
#End If
#If PAID
    #ApplicationLabel: Yahtzee! (Paid)
    #VersionCode: 89
    #VersionName: 8.9
#End If
```

You could add a new configuration FreeOrAmazon like this:



The screenshot shows a dialog box titled "Build Configurations". It has a close button (X) in the top right corner. The dialog contains the following fields and buttons:

- Configuration:** A dropdown menu currently showing "New\_1". To its right is a "Create New" button.
- Configuration Name:** A text input field containing "FreeOrAmazon". To its right is a "Delete" button.
- Package:** A text input field containing "b4a.example".
- Conditional Symbols:** A text input field containing "FreeOrAmazon, FREE, AMAZON". Below this field is the text "Example: Full, NoAds".
- At the bottom right, there are "Cancel" and "OK" buttons.

In Conditional Symbols we need to add also FREE and AMAZON to allow also these configurations.

In this case we need to add the FreeOrAmazon configuration to both FREE and AMAZON configurations.

And replace the code by:

```
#If FreeOrAmazon
  #ApplicationLabel: Yahtzee! (Free)
  #VersionCode: 119
  #VersionName: 11.9
#End If
#If PAID
  #ApplicationLabel: Yahtzee! (Free)
  #VersionCode: 119
  #VersionName: 11.9
#End If
```

## 2.3 Unsupported structure

This structure is not supported, no Else nor Else If.

```
#If XXX
#Else
#End If
```

## 3 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 Process_Globals
    Public SQL1 As SQL
    Public SomeBitmap As Bitmap
End Sub

Sub Activity_Create(FirstTime As Boolean)
    If FirstTime Then
        SQL1.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.

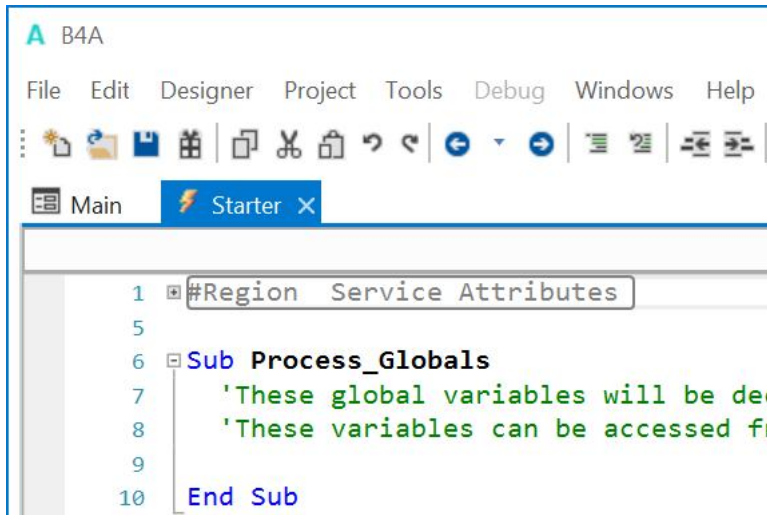
This means that the Starter service is the best place to initialize all the application-wide resources. Other modules can safely access these resources.

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 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.

When you open the IDE you see a service module called Starter.



You should put the code from the previous example

```

Sub Process_Globals
    Public SQL1 As SQL
    Public SomeBitmap As Bitmap
End Sub

Sub Activity_Create(FirstTime As Boolean)
    If FirstTime Then
        SQL1.Initialize(...)
        SomeBitmap = LoadBitmap(...)
        'additional code that loads application-wide resources
    End If
End Sub

```

to the Starter service module.

```

Sub Process_Globals
    Public SQL1 As SQL
    Public SomeBitmap As Bitmap

    SQL1.Initialize(...)
    SomeBitmap = LoadBitmap(...)
    'additional code that loads application-wide resources
End Sub

```

When you open a project developed with an older version of B4A than 5.20, the Starter service module will not be added, you need to add it yourself.

## 4 Libraries

In this chapter we will study some specific libraries.

### 4.1 IME Input Methods Editor

The IME library allows to modify the soft keyboard behaviour.

The library can be found here [IME library](#).

The most part of this chapter has been taken over from Erels' IME Tutorial. The example code has been changed a little bit.

Android has very good support for custom input method editors (IMEs). The downside for this powerful feature is that interacting with the soft keyboard can be sometimes quite complicated.

This library includes several utilities that will help you better handle the soft keyboard.

The methods are:

- **AddHandleActionEvent** (EditText As EditText)  
Adds the HandleAction event to the given EditText.
- **AddHeightChangedEvent**  
Enables the HeightChanged event.  
This event is raised when the soft keyboard state changes.  
You can use this event to resize other views to fit the new screen size.  
Note that this event will not be raised in full screen activities (an Android limitation)
- **HideKeyboard**  
Hides the soft keyboard if it is visible.
- **Initialize** (EventName As String)  
Initializes the object and sets the subs that will handle the events.
- **SetCustomFilter** (EditText As EditText, DefaultInputType As Int, AcceptedCharacters As String)  
Sets a custom filter.  
EditText - The target EditText.  
DefaultInputType - Sets the keyboard mode.  
AcceptedCharacters - The accepted characters.

Example: Create a filter that will accept IP addresses (numbers with multiple dots)  
`IME.SetCustomFilter(EditText1, EditText1.INPUT_TYPE_NUMBERS, "0123456789.")`

- **ShowKeyboard** (View As View)  
Sets the focus to the given view and opens the soft keyboard.  
The keyboard will only show if the view has received the focus.

The events are:

- **HandleAction** As Boolean
- **HeightChanged** (NewHeight As Int, OldHeight As Int)  
Raised when the keyboard has changed, but only if the event has been activated with the AddHeightChangedEvent method

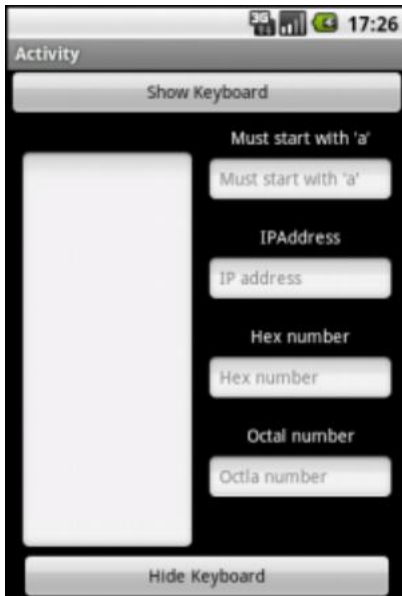
The attached example demonstrates the available methods.

The example is an extended example of Erels' project from the IME library.

Note that the IME object should be initialized before it can be used.

```
IME.Initialize(EventName As String)
```

```
IME1.Initialize("IME1")
```



Shows the keyboard

Must start with 'a'

limited to an IP address

limited to an hex number

limited to an octal number

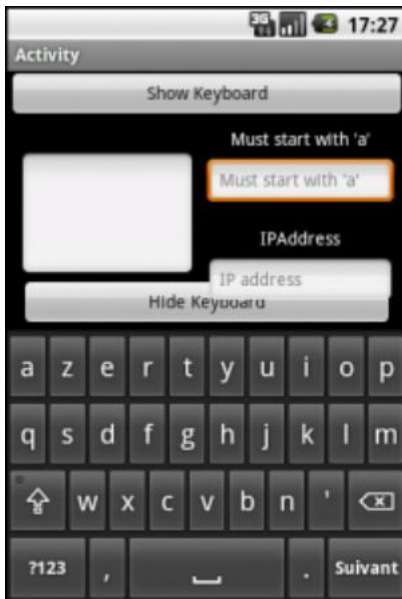
hides the keyboard

### 4.1.1 Handling the screen size changed event

When the keyboard opens the available screen size becomes much shorter. By default if the EditText is located near the bottom of the screen, Android will "push" the whole activity and make the EditText visible. This mode is named "adjustPan" mode.

By calling `IME1.AddHeightChangedEvent` you are changing the activity to "adjustSize" mode. In this mode the activity will not be pushed automatically. Instead the `HeightChanged` event will be raised when the keyboard is shown or hidden.

For example the following code makes sure that the button at the bottom is always visible and sets the large EditText height to match the available height:



When the keyboard is displayed the left EditText views height is adapted according to the keyboard size.

The Hide keyboard button is moved above the keyboard.

The code:

```
Sub IME1_HeightChanged(NewHeight As Int, OldHeight As Int)
    btnHideKeyboard.Top = NewHeight - btnHideKeyboard.Height
    edtTest.Height = btnHideKeyboard.Top - edtTest.Top
End Sub
```

Note that this method will not work if the activity is in full screen mode ([Issue 5497 - android - adjustResize windowSoftInputMode breaks when activity is fullscreen - Android](#)).

## 4.1.2 Showing and hiding the keyboard

**IME1.ShowKeyboard(EditText)** - Sets the focus to the given view and opens the soft keyboard.

```
Sub btnShowKeyboard_Click
    IME1.ShowKeyboard(edtStartWithA)
End Sub
```

**IME1.HideKeyboard** - Hides the keyboard (this method is the same as Phone.HideKeyboard).

```
Sub btnHideKeyboard_Click
    IME1.HideKeyboard
End Sub
```

## 4.1.3 Handle the action button

By calling `IME1.AddHandleActionEvent` you can override the default behaviour of the action button (the button that shows Next or Done).

This event is similar to `EditText_EnterPressed` event. However it is more powerful. It also allows you to handle the Next button and also to consume the message (and keep the keyboard opened and the focus on the current `EditText`).

This can be useful in several cases.

For example in a chat application you can send the message when the user presses on the done button and keep the keyboard open by consuming the message.

You can also use it to validate the input before jumping to the next view by pressing on the Next button (note that the user will still be able to manually move to the next field).

You can use the `Sender` keyword to get the `EditText` that raised the event.

```
Sub IME1_HandleAction As Boolean
    Dim edt As EditText

    edt = Sender

    Select edt.Tag
    Case "edtStartWithA"
        If edt.Text.StartsWith("a") = False Then
            ToastMessageShow("Text must start with 'a'.", True)
            'Consume the event.
            'The keyboard will not be closed
            Return True
        Else
            Return False 'will close the keyboard
        End If
    End Select
End Sub
```



### 4.1.4 Custom filters

`EditText.InputType` allows you to set the keyboard mode and the allowed input.

However there are situations where you need to use a custom filter. For example if you want to accept IP addresses (ex: 192.168.0.1). In this case none of the built-in types will work. Setting the input type to `INPUT_TYPE_DECIMAL_NUMBERS` will get you close but it will not allow the user to write more than a single dot.

`IME1.SetCustomFilter` allows you to both set the keyboard mode and also to set the accepted characters.

In this case we will need a code such as:

#### IP address :

```
IME1.SetCustomFilter(edtIPAddress, edtIPAddress.INPUT_TYPE_NUMBERS, "0123456789.")
```

- `edtIPAddress` is the given `EditText` view
- `edtIPAddress.INPUT_TYPE_NUMBERS` is the keyboard type
- "0123456789." are the allowed characters, in our case we accept not only numbers as with `INPUT_TYPE_NUMBERS` but we accept also dots.

Note that this is only a simple filter. It will accept the following input ...999 (which is not a valid IP address):

#### Hex number input :

' 0x0080000 is the flag of `NO_SUGGESTIONS`.

```
IME1.SetCustomFilter(edtHexNumber, Bit.Or(edtHexNumber.INPUT_TYPE_TEXT, _ 0x00080000), "01234567890abcdef")
```

0x00080000 is the flag for `NO_SUGGESTION`

```
Bit.Or(edtHexNumber.INPUT_TYPE_TEXT, 0x00080000)
```

combines the `INPUT_TYPE_TEXT` flag with the `_SUGGESTION` flag

#### Octal number input :

```
IME1.SetCustomFilter(edtOctalNumber, Bit.Or(edtOctalNumber.INPUT_TYPE_NUMBERS, _ 0x00080000), "01234567")
```

Note: With these filters the default keyboards is displayed but only the defined characters are authorized all the other keys are disabled but still visible.

## 4.2 #AdditionalJar attribute

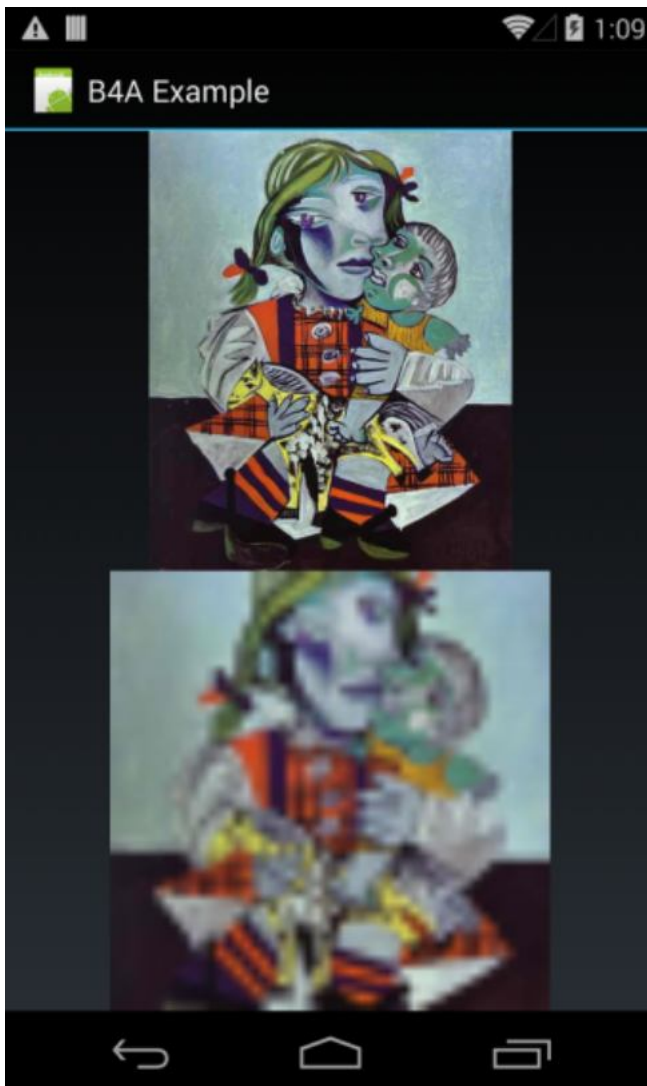
This is a copy of Erels tutorial in the forum.

The #AdditionalJar module attribute (introduced in B4A v3.80) allows us to reference external jars. With the help of JavaObject it is now possible to integrate third party jars without a wrapper.

This solution is good for "simple" libraries. If the API is complicated with many interfaces then it will be easier to create a wrapper.

As an example we will use Picasso image downloader to download images:

<http://square.github.io/picasso/>



The first step is to download the third party jar and put it in the additional libraries folder. We then use #AdditionalJar to tell the compiler to add a reference to this jar:

```
#AdditionalJar: picasso-2.2.0
```

Note that the jar extension is omitted. You can call #AdditionalJar multiple times if multiple jars are required.

The following two subs will usually be required. They allow you to get the "context" (it will be an android.app.Activity when called from an Activity module).

This code should be added to an activity or service directly.

```
Sub GetContext As JavaObject
    Return GetBA.GetField("context")
End Sub
```

```
Sub GetBA As JavaObject
    Dim jo As JavaObject
    Dim cls As String = Me
    cls = cls.SubString("class ".Length)
    jo.InitializeStatic(cls)
    Return jo.GetFieldJO("processBA")
End Sub
```

```
Picasso.with(context).load("http://i.imgur.com/DvpvklR.png").into(imageView);
```

As you can see in their examples we always start by calling Picasso static method 'with'. It is more clear in the JavaDocs page:

<http://square.github.io/picasso/javadoc/com/squareup/picasso/Picasso.html>

This sub will call the static method:

```
Sub GetPicasso As JavaObject
    Dim jo As JavaObject
    ' com.squareup.picasso.Picasso.with(context)
    Return jo.InitializeStatic("com.squareup.picasso.Picasso").RunMethod("with", _ Array(
GetContext))
End Sub
```

Now we will implement the above Java code:

```
GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("into", Array(img1))
```

In the second example we call a more complex API:

```
' second example: Picasso.with(context).load(url).resize(50, 50).centerCrop().into(ImageView)
GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("resize", Array(50, 50)) _
    .RunMethodJO("centerCrop", Null).RunMethodJO("into", Array(img2))
```

The third example is more interesting. We download an image with a callback event that is raised when download completes.

The first step is to create the interface. This is done with JavaObject.CreateEvent (or CreateEventFromUI).

In this case we are implementing com.squareup.picasso.Callback:

<http://square.github.io/picasso/javadoc/com/squareup/picasso/Callback.html>

```
Dim callback As Object = jo.CreateEvent("com.squareup.picasso.Callback", _
"Callback", Null)
```

The last parameter is the default return value. This value will be used if the event cannot be raised (activity is paused for example). In this case we return Null.

The event sub:

```
Sub CallBack_Event (MethodName As String, Args() As Object) As Object
    If MethodName = "onSuccess" Then
        ToastMessageShow("Success!!!", True)
    Else If MethodName = "onError" Then
        ToastMessageShow("Error downloading image.", True)
    End If
    Return Null
End Sub
```

MethodName - The interface method name (onSuccess or onError in this case).

Args - An array of parameters passed to this method. In this case there are no parameters.

All this information is from Picasso JavaDocs:

<http://square.github.io/picasso/javadoc/index.html?com/squareup/picasso/Callback.html>

The last step is to call the method that expects the callback:

```
GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("into", Array(img1, _
callback))
```

As this library requires the INTERNET permission we need to manually add it to the manifest editor:

```
AddPermission(android.permission.INTERNET)
```

The complete code:

```
#Region Project Attributes
    #ApplicationLabel: B4A Example
    #VersionCode: 1
    #VersionName:
    'SupportedOrientations possible values: unspecified, landscape or portrait.
    #SupportedOrientations: unspecified
    #CanInstallToExternalStorage: False
#End Region

#Region Activity Attributes
    #FullScreen: False
    #IncludeTitle: True
#End Region

#AdditionalJar: picasso-2.2.0

Sub ProcessGlobals

End Sub

Sub Globals
    Dim img1, img2 As ImageView
End Sub
```

```

Sub Activity_Create(FirstTime As Boolean)
    img1.Initialize("")
    Activity.AddView(img1, 0, 0, 100%x, 50%y)
    img2.Initialize("")
    Activity.AddView(img2, 0, 50%y, 100%x, 50%y)
    Dim url As String = "http://i.imgur.com/DvpvklR.png"
    'first example: Picasso.with(context).load(url).into(imageView);
    GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("into", Array(img1))

    'second example: Picasso.with(context).load(url).resize(50, 50).centerCrop().into(ImageView)
    GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("resize", Array(50, 50)) _
        .RunMethodJO("centerCrop", Null).RunMethodJO("into", Array(img2))

    'third example: download image with callback
    Dim jo As JavaObject = GetPicasso
    Dim callback As Object = jo.CreateEvent("com.squareup.picasso.Callback", "Callback",
    Null)
    GetPicasso.RunMethodJO("load", Array(url)).RunMethodJO("into", Array(img1, callback)
)
End Sub

Sub Callback_Event (MethodName As String, Args() As Object) As Object
    If MethodName = "onSuccess" Then
        ToastMessageShow("Success!!!", True)
    Else If MethodName = "onError" Then
        ToastMessageShow("Error downloading image.", True)
    End If
    Return Null
End Sub

Sub GetPicasso As JavaObject
    Dim jo As JavaObject
    'com.squareup.picasso.Picasso.with(context)
    Return jo.InitializeStatic("com.squareup.picasso.Picasso").RunMethod("with", Array(GetContext))
End Sub

Sub GetContext As JavaObject
    Return GetBA.GetField("context")
End Sub

Sub GetBA As JavaObject
    Dim jo As JavaObject
    Dim cls As String = Me
    cls = cls.SubString("class ".Length)
    jo.InitializeStatic(cls)
    Return jo.GetFieldJO("processBA")
End Sub

Sub Activity_Resume

End Sub

Sub Activity_Pause (UserClosed As Boolean)

End Sub

```

## 5 SQLite Database

What is a database (source Wikipedia [Database](#)):

A **database** is an organized collection of data for one or more purposes, usually in digital form. The data are typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies). The term "database" refers both to the way its users view it, and to the logical and physical materialization of its data, content, in files, computer memory, and computer data storage. This definition is very general, and is independent of the technology used. However, not every collection of data is a database; the term database implies that the data is managed to some level of quality (measured in terms of accuracy, availability, usability, and resilience) and this in turn often implies the use of a general-purpose [Database management system](#) (DBMS). A general-purpose DBMS is typically a complex software system that meets many usage requirements, and the databases that it maintains are often large and complex.

The standard database system in Android is [SQLite](#).

The interface between your program and the database is the SQL language.

The data is stored in tables, each table has a certain number of columns and rows.

Each row contains a data set and the different data of a given set are stored in the columns.

**If you add a default database to your project in the files Tab it is located in the DirAssets folder. Databases cannot be accessed in DirAssets even if it's only for reading.**

Therefore you must copy it to another folder for example DirInternal or DirRootExternal.

With DirRootExternal you can also add a subdirectory.

For example: DirRootExternal & "/MyDatabase"

Don't forget to create the subdirectory : `File.MakeDir(File.DirRootExternal, "MyDatabase")`

Example code in the Starter module:

```
If File.Exists(File.DirRootExternal, "Database.db") = False Then
    File.Copy(File.DirAssets, "Database", File.DirRootExternal, "Database.db")
End If
SQL1.Initialize(File.DirRootExternal, "Database.db", True)
```

Or in Activity\_Create if you have only one Activity:

```
If FirstTime Then
    If File.Exists(File.DirRootExternal, "Database.db") = False Then
        File.Copy(File.DirAssets, "Database", File.DirRootExternal, "Database.db")
    End If
    SQL1.Initialize(File.DirRootExternal, "Database.db", True)
End If
```

## 5.1 SQLite Database basics

Some simple SQL instructions.

Here you find the SQLite site : [SQLite](#)

Here you find the SQLite syntax : [SQLite syntax](#)

A very interesting website to learn SQL is this one : [W3Schools SQL](#).

### 5.1.1 Database initialisation SQL1.Initialize

To use a database you must first initialize it !

This is ideally done in the Starter service.

```
SQL1.Initialize(DBDirName, DBFileName, True)
```

DBDirName = Directory name of the database.

DBFileName = Database file name.

True = Create if necessary      False don't create

```
SQL1.Initialize(DBDirName, DBFileName, True)
```

### 5.1.2 Table creation CREATE TABLE

You can create a database in a SQLite program on the PC or you can create it in the code like below.

```
CREATE TABLE TableName (Col1 INTEGER, Col1 TEXT, Col2 REAL )
```

Creates a table with the name 'TableName' and three columns:

Column Index	Name	Variable Type
1	Col1	INTEGER
2	Col2	TEXT
3	Col3	REAL

```
SQL1.ExecNonQuery("CREATE TABLE TableName(Col 1 INTEGER, Col 2 TEXT, Col 3 REAL")
```

Only these data types are available:

INTEGER is a 64-bit signed integer number.

REAL is a 64-bit IEEE floating point number.

TEXT is a string.

BLOB **B**inary **L**arge **O**bject, the value is stored exactly as it was input.

NULL

INTEGER PRIMARY KEY is a special variable type used for identifiers ID's. It is a long integer value beginning with 1 and it is incremented by one each time a new data set is added to the database.

### 5.1.3 INTEGER PRIMARY KEY rowID

INTEGER PRIMARY KEY is a special data type which is unique and will never change. You can define a specific column dedicated to the PRIMARY KEY. This is used in the SQLiteLight1, SQLiteLight2 and SQLiteLight3 examples.

But this is not mandatory, SQLite has an internal column named *rowID* which can be used. This is used in the SQLiteLight4 example.

Each time you add a new record the PRIMARY KEY is incremented by 1.

When you delete a record the PRIMARY KEY of this record is **lost**.

When you load a database and display it in a table be aware that the row indexes in the table are not the same as the database rowIDs. Therefore you must read and memorize the PRIMARY KEYS somewhere to know which record is in which line.

Comparison:

- Creation.
  - With a specific ID column.  
"CREATE TABLE persons (ID INTEGER PRIMARY KEY, FirstName TEXT, LastName TEXT, City TEXT)"
  - With no specific column.  
"CREATE TABLE persons (FirstName TEXT, LastName TEXT, City TEXT)"
- Reading.
  - With a specific ID column.  
"SELECT ID, FirstName AS [First name], LastName AS [Last name], City FROM persons"
  - With no specific column.  
Reads the PRIMARY Key in the query.  
"SELECT rowID AS ID, FirstName AS [First name], LastName AS [Last name], City FROM persons"  
Doesn't read the PRIMARY Key in the query.  
"SELECT FirstName AS [First name], LastName AS [Last name], City FROM persons"

Note: If you use this query "SELECT \* FROM persons" the rowID column is not included. If you want it you must specify it like in the examples above, or read it in a separate query.
- Inserting.
  - With a specific ID column.  
"INSERT INTO persons VALUES (NULL, 'John', 'KERRY, 'Boston')"  
You must use NULL for the PRIMARY KEY column.
  - With no specific column.  
"INSERT INTO persons VALUES ('John', 'KERRY, 'Boston')"



**5.1.4 Adding data****INSERT INTO**

INSERT INTO TableName VALUES ( Val1, 'Val2' Val3 )

SQL1.ExecNonQuery("INSERT INTO TableName VALUES (Val 1, ' Val 2' , Val 2")

Text variable must be between two quotes like 'Val2', numbers not like Val1 and Val3.

**5.1.5 Updating data****UPDATE**

UPDATE TableName Set Col1 = Val1, Col2 = 'Val2', Col3 = Val3 WHERE ID = idVal

SQL1.ExecNonQuery("UPDATE TableName Set Col 1 = Val 1, Col 2 = ' Val 2' , Col 3 = Val 3 WHERE ID = i dVal ")

Again, text variable must be between two quotes like 'Val2', numbers not like Val1 and Val3.

**5.1.6 Reading data****SELECT**

The SELECT statement is used to query the database. The result of a SELECT is zero or more rows of data where each row has a fixed number of columns. A SELECT statement does not make any changes to the database.

Examples:

- The whole database:  
SELECT \* FROM TableName  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName")
- A single column  
SELECT Col1 FROM TableName  
Cursor1 = SQL1.ExecQuery("SELECT Col 1 FROM TableName")
- Distinct values from a column, no duplicate values  
SELECT DISTINCT Col1 FROM TableName  
Cursor1 = SQL1.ExecQuery("SELECT DI STINCT Col 1 FROM TableName")
- Single entry (value)  
SELECT Col1 FROM TableName WHERE rowID = idVal  
Val ue = SQL1.ExecQuerySi ngl eResul t("SELECT Col 1 FROM TableName WHERE rowID = i dVal ")
- Max / min value in a column, in the examples the max and min values of the given column.  
SELECT max(Col1) FROM TableName  
SELECT min(Col1) FROM TableName  
Max = SQL1.ExecQuerySi ngl eResul t("SELECT max(Col 1) FROM TableName")  
Mi n = SQL1.ExecQuerySi ngl eResul t("SELECT mi n(Col 1) FROM TableName")

- Get the sum or average of a column  

```
SELECT sum(Col1) FROM TableName
SELECT avg(Col1) FROM TableName
```

Sum = SQL1.ExecQuerySingleResult("SELECT sum(Col 1) FROM Table Name")  
Average = SQL1.ExecQuerySingleResult("SELECT avg(Col 1) FROM Table Name")
- Get calculations of columns.  
For example, in a database with a column *Number* of type INTEGER and another column *Price* of type REAL we want to get the Cost = Number \* Price.  

```
SELECT Number, Price, Number * Price FROM TableName
```

Cursor1 = SQL1.ExecQuery("SELECT Number, Price, Number \* Price FROM Table Name")  
Number = Cursor1.GetInt2(0)  
Price = Cursor1.GetDouble2(1)  
Cost = Cursor1.GetDouble2(2)

Or giving the result a column name with Number \* Price AS Cost.  

```
SELECT Number, Price, Number * Price AS Cost FROM
TableName")
```

Number = Cursor1.GetInt("Number")  
Price = Cursor1.GetDouble("Price")  
Cost = Cursor1.GetDouble("Cost")

Some functions:

- sum()           Calculates the sum of a column.
- avg()           Calculates the average of a column.
- min()           Calculates the min value of column.
- max()           Calculates the min value of column.
- length()        Calculates the number of characters of a string or the number of characters of the string representation of a number.
- lower()Returns a string in lower case characters.
- upper()Returns a string in upper case characters.
- typeof()       Returns the data type of a column.

More details can be found in the SQLite documentaion here: [Core Functions](#)

and here : [expressions](#)

and here : [Date And Time Functions](#)

## 5.1.7 Filtering

## WHERE

After the SELECT expression you can add a WHERE expression for filtering.

The WHERE expression is evaluated for each row in the input data as a boolean expression. Only rows for which the WHERE clause expression evaluates to true are included from the dataset before continuing. Rows are excluded from the result if the WHERE clause evaluates to either false or NULL.

Some filtering functions:

- = > < >= <=
- AND OR BETWEEN
- LIKE

Examples:

- A single row.  
Where the rowID has the value of the variable idVal  
SELECT \* FROM TableName WHERE rowID = idVal  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE rowID = " & idVal)  
Where an ID column has the value of the variable idVal  
SELECT \* FROM TableName WHERE ID = idVal  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE ID = " & idVal)
- A single entry (value).  
SELECT Col1 FROM TableName WHERE rowID = idVal  
Value = SQL1.ExecQuerySingleResult("SELECT Col1 FROM TableName WHERE rowID = idVal")
- The rows where columns have given values.  
SELECT \* FROM TableName WHERE Col1 LIKE 'abc' AND Col2 LIKE 123  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE Col1 LIKE 'abc%' AND Col2 LIKE 123")  
The % character can be used as a wildcard:  
abc means the exact sequence  
%abc means beginning with any characters and ending with abc  
abc% means beginning with abc and ending with any characters  
%abc% means abc anywhere in the string
- The rows where a value in a column is between two given values.  
SELECT \* FROM TableName WHERE Col1 >= minVal AND Col1 <= maxVal  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE Col1 >= minVal AND Col1 <= maxVal")  
Or with BETWEEN which is the same.  
SELECT \* FROM TableName WHERE Col1 BETWEEN minVal AND maxVal  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE Col1 BETWEEN minVal AND maxVal")  
Or with minVal and maxVal being variables:  
Cursor1 = SQL1.ExecQuery("SELECT \* FROM TableName WHERE Col1 BETWEEN " & minVal & " AND " & maxVal)

### 5.1.8 Sorting

### ORDER BY

If a SELECT statement that returns more than one row does not have an ORDER BY clause, the order in which the rows are returned is undefined.

Or, if a SELECT statement does have an ORDER BY clause, then the list of expressions attached to the ORDER BY determine the order in which rows are returned to the user.

A query can be sorted either ascending or descending.

Add an ORDER BY expression at the end of the query.

- Read the whole database and ordering according to a given column:

```
SELECT * FROM TableName ORDER BY Col1 ASC           ascending
Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName ORDER BY Col1 ASC")
```

```
SELECT * FROM TableName ORDER BY 2 DESC           descending
Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName ORDER BY 2 DESC")
```

The column to order can be given either by its name Col1 or its number 2.

The column numbering begins with 1.

- Read the given columns and sort on two of them.

```
SELECT FirstName AS [First name], LastName AS [Last name], City FROM persons
ORDER BY LastName ASC, FirstName ASC
Cursor1 = SQL1.ExecQuery("SELECT FirstName AS [First name], LastName AS [Last
name], Ci ty FROM persons ORDER BY LastName ASC, FirstName")
```

The braquets [Fi rst name] are needed because of the spaces in the alias column names.

### 5.1.9 Date / Time functions

SQLite has several date / time functions.

Below the most useful for B4A:

- `date(timestring, modifier, modifier, ...)`  
Returns a date.
- `time(timestring, modifier, modifier, ...)`  
Returns a time.
- `datetime(timestring, modifier, modifier, ...)`  
Returns a date and time.

For more details, examples and what timestring and modifiers are, please look at the [SQLite documentation](#).

In B4A the best way to store dates is to store them as ticks, which are the number of milliseconds since January 1, 1970.

SQLite doesn't have the same ticks but has "unixepoch" ticks which are the number of seconds since January 1, 1970.

Examples of queries with Ticks = 1448795854111 (B4A ticks):

- `SQL1.ExecQuerySingleResult("SELECT date(" & (Ticks / 1000) & " , 'unixepoch')")`  
Returns: 2015-11-29
- `SQL1.ExecQuerySingleResult("SELECT time(" & (Ticks / 1000) & " , 'unixepoch')")`  
Returns: 11:17:34
- `SQL1.ExecQuerySingleResult("SELECT datetime(" & (Ticks / 1000) & " , 'unixepoch')")`  
Returns: 2015-11-29 11:17:34

In these examples `(Ticks / 1000)` is the timestring and `'unixepoch'` a modifier.

The `date()` function is used in the SQLiteLight4 example.

### 5.1.10 Other functions

- Get the data type of columns.

In a table with a column *Part* of type TEXT *Number* of type INTEGER and another column *Price* of type REAL

```
SELECT typeof(Item), typeof(Number), typeof(Price) FROM TableName
```

```
Cursor1 = SQL1.ExecQuery("SELECT typeof(Part), typeof(Number), typeof(Price) FROM TableName")
```

Get the data type with:

Column	request	or	other request	>	result
Part:	Cursor1.GetString("Part")	or	Cursor1.GetString2(0)	>	text
Number:	Cursor1.GetString("Number")	or	Cursor1.GetString2(1)	>	integer
Price:	Cursor1.GetString("Price")	or	Cursor1.GetString2(2)	>	real

- Get the max length of the data in a column.

For a string, the returned value is the number of characters not the number of bytes.

For a number, the returned value is the number of characters of its string representation.

For a blob, the returned value is the number of bytes.

```
SELECT max(length(Col1)) FROM TableName
```

```
MaxChars = SQL1.ExecQuerySingleResult("SELECT max(length(Col1)) FROM TableName")
```

- Get the total number of rows

```
SELECT count() FROM TableName
```

```
NumberOfRows = SQL1.ExecQuerySingleResult("SELECT count() FROM TableName")
```

- Get the tables in the database

```
SELECT name FROM sqlite_master WHERE Type='table'
```

```
Cursor1 = SQL1.ExecQuery("SELECT name FROM sqlite_master Where Type=' table' ")
```

- Get the column names of a table.

```
SELECT * FROM TableName
```

```
Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName")
```

```
For i = 0 to Cursor1.ColumnCount - 1
```

```
    ColumnName(i) = Cursor1.GetColumnName(i)
```

```
Next
```

- Get the number of database rows that were changed or inserted or deleted by the most recently completed INSERT, DELETE, or UPDATE.

```
SELECT changes() FROM TableName
```

```
NbChanges = SQL1.ExecQuerySingleResult("SELECT changes() FROM TableName")
```

- Get the PRIMARY KEYs from a table and save them in a List, *rowid* is a reserved column name. This is valid even if there is no column defined with PRIMARY KEY.

This function throughs an error if the table is empty !

```
SELECT rowid FROM TableName
```

```
Dim IDList As List
```

```
IDList.Initialize
```

```
Cursor1 = SQL1.ExecQuery("SELECT rowid FROM TableName")
```

```
For i = 0 To Cursor1.RowCount - 1
```

```
    Cursor1.Position = i
```

```
    IDList.Add(Cursor1.GetLong2(0))
```

```
Next
```

### 5.1.11 Get Table information PRAGMA

It uses a special query PRAGMA.

This query returns one row per column with following data :

Column index name Explanation

- 0 `ci d` column index
- 1 `name` column name
- 2 `type` data type
- 3 `dfl t_val ue` default value
- 4 `notnul l` null if the database accepts null values
- 5 `pk` primary key = 1 if the column is a PRIMARY KEY otherwise = 0  
This is valid only if a column with a primary key was created.

```
Cursor1 = SQL1.ExecQuery("PRAGMA tabl e_i nfo (Tabl eName)")
```

```
For i = 0 To Cursor1.RowCount - 1
```

```
Cursor1.Position = i
```

```
For j = 0 To Cursor1.ColumnCount - 1
```

```
Log(i & " / " & j & " : " & Cursor1.GetStri ng2(j))
```

```
Next
```

```
Log(" ")
```

```
Next
```

Or this code:

```
Cursor1 = SQL1.ExecQuery("PRAGMA tabl e_i nfo (Tabl eName)")
```

```
For i = 0 To Cursor1.RowCount - 1
```

```
Cursor1.Position = i
```

```
Log("ID : " & Cursor1.GetStri ng("ci d"))
```

```
Log("Name : " & Cursor1.GetStri ng("name"))
```

```
Log("Type : " & Cursor1.GetStri ng("type"))
```

```
Log("Defaul t val ue : " & Cursor1.GetStri ng("dfl t_val ue"))
```

```
Log("Not nul l : " & Cursor1.GetStri ng("notnul l"))
```

```
Log("Primaty key : " & Cursor1.GetStri ng("pk"))
```

```
Log(" ")
```

```
Next
```

**5.1.12 Deleting data DELETE FROM**

DELETE FROM TableName WHERE ID = idVal  
 SQL1.ExecNonQuery("DELETE FROM TableName WHERE ID = idVal")

**5.1.13 Rename a table ALTER TABLE Name ADD COLUMN**

Renames a given table.

ALTER TABLE TableName RENAME TO NewTableName)  
 SQL1.ExecNonQuery("ALTER TABLE TableName RENAME TO NewTableName")

**5.1.14 Add a column ALTER TABLE Name ADD COLUMN**

Add a new column to the database.

ALTER TABLE TableName ADD COLUMN Colname ColType)  
 SQL1.ExecNonQuery("ALTER TABLE TableName ADD COLUMN ColN TEXT")

**5.1.14.1 Update the database after having added a column**

Update the database after having added a new column.

- Sets the values of all rows in the new column to an empty string.  
 UPDATE TableName SET ColName = "  
 SQL1.ExecNonQuery("UPDATE TableName SET ColN = ''")
- Sets the values of the rows in a column to a given new value where the value is another old value.  
 UPDATE TableName SET ColName = 'ValueNew' WHERE ColName = 'ValueOld'  
 SQL1.ExecNonQuery("UPDATE TableName SET ColN = 'ValueNew' WHERE ColN = 'ValueOld'")

**5.1.15 Delete a table DROP TABLE**

The DROP TABLE statement removes a table added with the CREATE TABLE statement. The name specified is the table name. The dropped table is completely removed from the database schema and the disk file. The table can not be recovered. All indices and triggers associated with the table are also deleted.

The optional IF EXISTS clause suppresses the error that would normally result if the table does not exist.

DROP TABLE IF EXISTS TableName  
 SQL1.ExecNonQuery("DROP TABLE IF EXISTS TableName")



### 5.1.16 Insert an image

To insert an image we need a BLOB (Binary Large Object).  
The column type in the database must be set to BLOB !

#### Sub InsertBlob

```
'convert the image file to a bytes array
Dim InputStream1 As InputStream
InputStream1 = File.OpenInput(File.DirAssets, "smiley.gif")
Dim OutputStream1 As OutputStream
OutputStream1.InitializeToByteArray(1000)
File.Copy2(InputStream1, OutputStream1)
Dim Buffer() As Byte 'declares an empty array
Buffer = OutputStream1.ToByteArray

'write the image to the database
SQL1.ExecNonQuery2("INSERT INTO table2 VALUES('smiley', ?)", Array As Object(Buffer))
```

End Sub

Here we are using a special type of OutputStream which writes to a dynamic bytes array. File.Copy2 copies all available data from the input stream into the output stream. Then the bytes array is written to the database.

### 5.1.17 Read an image

Using a Cursor.GetBlob we fetch the stored image.  
Now we are using an input stream that reads from this array and load the image.

#### Sub ReadBlob

```
Dim Cursor1 As Cursor
'Using ExecQuery2 is safer as it escapes special characters automatically.
'In this case it doesn't really matter.
Cursor1 = SQL1.ExecQuery2("SELECT image FROM table2 WHERE name = ?", Array As
String("smiley"))
Cursor1.Position = 0
Dim Buffer() As Byte 'declare an empty byte array
Buffer = Cursor1.GetBlob("image")
Dim InputStream1 As InputStream
InputStream1.InitializeFromByteArray(Buffer, 0, Buffer.Length)

Dim Bitmap1 As Bitmap
Bitmap1.Initialize2(InputStream1)
InputStream1.Close
```

End Sub

### 5.1.18 ExecQuery vs ExecQuery2 / ExecNonQuery vs ExecNonQuery2

The examples below suppose a table with three columns:  
Col1 TEXT, Col2 INTEGER, Col3 INTEGER

There exist two methods to execute a query.

- **ExecQuery(Query As String)**

Executes the query, you must take care of the datatype.

Example:

```
Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 = ' " & MyText & "'
AND Col2 >= " & minVal & " AND Col2 <= " & maxVal )
```

Note that MyText is between two quotes because the data field is a TEXT field!

- **ExecQuery2(Query As String, StringArgs As Object())**

The query includes question marks which will be replaced with the values in the array.

Example:

```
Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 = ? AND Col2 >= ?
AND Col2 <= ? ", Array As Object MyText, minVal, maxVal))
```

Note that ExecQuery2 is safer because it takes care of the column data type!

The same for ExecNonQuery.

- **ExecNonQuery(Query As String)**

Executes the query, you must take care of the datatype.

Example:

```
SQL1.ExecNonQuery("INSERT INTO table1 VALUES(' abc' , 1, 2)")
```

Note that abc is between two quotes because the data field is a TEXT field!

- **ExecNonQuery2(Query As String, StringArgs As Object())**

The query includes question marks which will be replaced with the values in the array.

Example:

```
SQL1.ExecNonQuery2("INSERT INTO table1 VALUES(?, ?, ?)", Array As Object("abc",
3, 4))
```

Note that ExecQuery2 is safer because it takes care of the column data type!

The same exists for ExecQuerySingleResult and ExecQuerySingleResult2.

### 5.1.19 Insert many rows SQL.BeginTransaction / SQL.EndTransaction

#### Sub InsertManyRows

```

SQL1. BeginTransaction
Try
  For i = 1 To 500
    SQL1.ExecNonQuery2("INSERT INTO table1 VALUES ('def', ?, ?)", Array As Object(i,
i))
  Next
  SQL1.TransactionSuccessful
Catch
  Log(LastException.Message)
End Try
SQL1.EndTransaction
End Sub

```

This code is an example of adding many rows. Internally a lock is acquired each time a "writing" operation is done.

By explicitly creating a transaction the lock is acquired once.

The above code took less than half a second to run on a real device.

Without the BeginTransaction / EndTransaction block it took about 70 seconds.

A transaction block can also be used to guarantee that a set of changes were successfully done.

Either all changes are made or none are made.

By calling SQL.TransactionSuccessful we are marking this transaction as a successful transaction.

If you omit this line, all the 500 INSERTS will be ignored.

It is very important to call EndTransaction eventually.

Therefore the transaction block should usually look like:

```

SQL1. BeginTransaction
Try
  'Execute the sql statements.
SQL1.TransactionSuccessful
Catch
  'the transaction will be cancelled
End Try
SQL1.EndTransaction

```

### 5.1.20 Asynchronous queries

The SQL library supports asynchronous select queries and asynchronous batch inserts.

Asynchronous means that the task will be processed in the background and an event will be raised when the task completes. This is useful when you need to issue a slow query and keep your application responsive.

The usage is quite simple:

```
sql 1. ExecQueryAsync("SQL", "SELECT * FROM table1", Null)
...
Sub SQL_QueryComplete (Success As Boolean, Crsr As Cursor)
  If Success Then
    For i = 0 To Crsr.RowCount - 1
      Crsr.Position = i
      Log(Crsr.GetInt2(0))
    Next
  Else
    Log(LastException)
  End If
End Sub
```

The first parameter is the "event name". It determines which sub will handle the QueryComplete event.

### 5.1.21 Batch inserts AddNonQueryToBatch / ExecNonQueryBatch

SQL.AddNonQueryToBatch / ExecNonQueryBatch allow you to asynchronously process a batch of non-query statements (such as INSERT statements).

You should add the statements by calling AddNonQueryToBatch and eventually call ExecNonQueryBatch.

The task will be processed in the background. The NonQueryComplete event will be raised after all the statements execute.

```
For i = 1 To 10000
  SQL1.AddNonQueryToBatch("INSERT INTO table1 VALUES (?)", Array As Object(Rnd(0,
  10000)))
Next
SQL1.ExecNonQueryBatch("SQL")
...
Sub SQL_NonQueryComplete (Success As Boolean)
  Log("NonQuery: " & Success)
  If Success = False Then Log(LastException)
End Sub
```

## 5.2 Multiple tables

A database can, of course, have more than one table.

Example: This is only a small simple example to demonstrate the principle.  
Demo code example project SQLiteLight4.

Database with 3 tables:

- |             |                                       |                                    |                                   |
|-------------|---------------------------------------|------------------------------------|-----------------------------------|
| • Stock     | Number INTEGER,<br>number of products | ProductID INTEGER,<br>product ID   | Date INTEGER<br>date in Ticks     |
| • Products  | Name TEXT,<br>product name            | Price REAL,<br>product price       | SupplierID INTEGER<br>supplier ID |
| • Suppliers | Name TEXT,<br>suppliers name          | Address TEXT,<br>suppliers address | City TEXT<br>suppliers city       |

In the table Stock we use the ID of the product rather its name.  
The same in the table Products for the Supplier.

Query example of a call for display:

```
Query = "SELECT Stock.Number, Products.Name AS Product, Suppliers.Name AS Supplier,
Products.Price AS Price, Stock.Number * Products.Price AS Value, date(Stock.Date /
1000, 'unixepoch') AS Date"
Query = Query & " FROM Stock, Products, Suppliers"
Query = Query & " WHERE Products.rowID = Stock.ProductID AND Suppliers.rowID =
Products.SupplierID"
```

We want to read following data:

- The number of items in stock `Stock.Number`.  
The Number column in the Stock table.
- The product name `Products.Name AS Product`.  
The Name column in the Products table and give this column the name 'Product'.
- The supplier name `Suppliers.Name AS Supplier`.  
The Name column in the Suppliers tabel and give this column the name 'Supplier'.
- The product price `Products.Price AS Price`.  
The Price column in the Products table and give this column the name 'SPrice'.
- The value of these products in stock `Stock.Number * Products.Price AS Value`.  
The multiplication of the number of items in stock with the product price and give this column the name 'Value'.
- The date when the product was entered `date(Stock.Date / 1000, 'unixepoch') AS Date`.  
We use the SQLite date function where we give the Date column of the Stock table.  
As the date is in B4A ticks we need to devide the value by 1000 to adapt it to 'SQL ticks'  
and we must add the parameter 'unixepoch' for 'SQL ticks'.

The query concerns the three tables Stock, Products and Suppliers:

```
FROM Stock, Products, Suppliers
```

We must add a WHERE expression:

- To connect the Products table rowID to the Stock ProductID column value.  
`Products.rowID = Stock.ProductID`
- To connect the Suppliers table rowID to the Products SupplierID column value.  
`Suppliers.rowID = Products.SupplierID`

Example of the result:



Number	Product	Supplier	Price	Value	Date
5	Break	Renault	350	1750	2004-08-19
10	Break	VW	175	1750	2010-02-07
15	Exhaust pipe	Ford	150	2250	2012-02-07

For more details look at the SQLiteLight4 example program.

### 5.3 Transaction speed

If you have to do many inserts into a database you should `BeginTransaction` and `EndTransaction` this will considerably speed up the process.

A transaction is a set of multiple "writing" statements that are atomically committed.

It is very important to handle transaction carefully and close them.

The transaction is considered successful only if `TransactionSuccessful` is called. Otherwise no changes will be made.

Typical usage:

```
SQL1. BeginTransaction
```

```
Try
```

```
    'block of statements like:
```

```
    For i = 1 To 1000
```

```
        SQL1.ExecNonQuery("INSERT INTO table1 VALUES(...)")
```

```
    Next
```

```
    SQL1.TransactionSuccessful
```

```
Catch
```

```
    Log(LastException.Message) 'no changes will be made
```

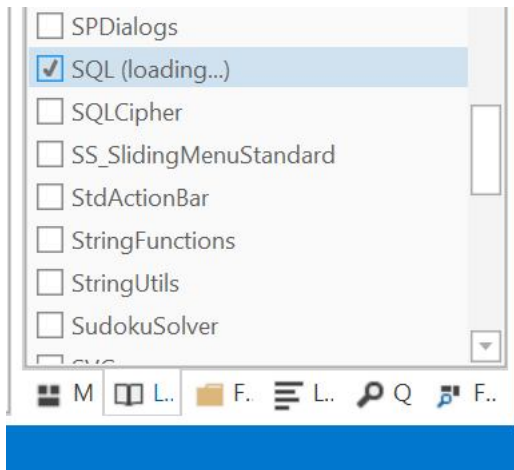
```
End Try
```

```
SQL1.EndTransaction
```

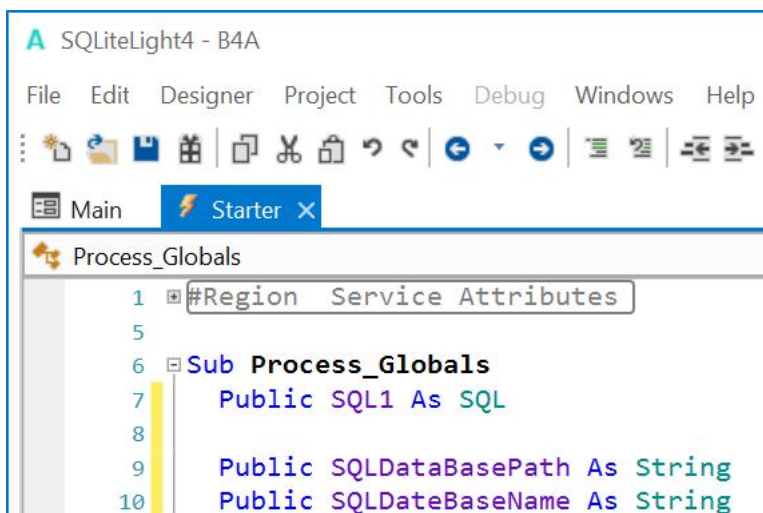
## 5.4 First steps

To use a database, we must:

- First reference the SQL library in the Libs Tab in the lower right corner in the IDE.



- Declare it with `Public` in the `Process_Globals` routine of the Starter Service module or in the `Process_Globals` routine of the Main module if you don't use the Starter Service. I suggest to define two other variables for the database path and file name:



- Initialize it in the Service\_Create routine in the Starter Service.

We give the two variables values, for example:

#### Sub Service\_Create

```
'initialize the variables
SQLDataBasePath = File.DirRootExternal
SQLDateBaseName = "stock.db"
```

- If you already have a database in the Files folder of the project you need to copy it from File.DirAssets in another folder.  
Databases are NOT accessible from File.DirAssets !

```
'check if the database already exists
If File.Exists(SQLDataBasePath, SQLDateBaseName) = True Then
  'if yes initialize it
  SQL1.Initialize(SQLDataBasePath, SQLDateBaseName, True)
Else
  'if no copy it
  File.Copy(File.DirAssets, SQLDateBaseName, SQLDataBasePath,
SQLDateBaseName)
  'and initialize it
  SQL1.Initialize(SQLDataBasePath, SQLDateBaseName, True)
End If
```

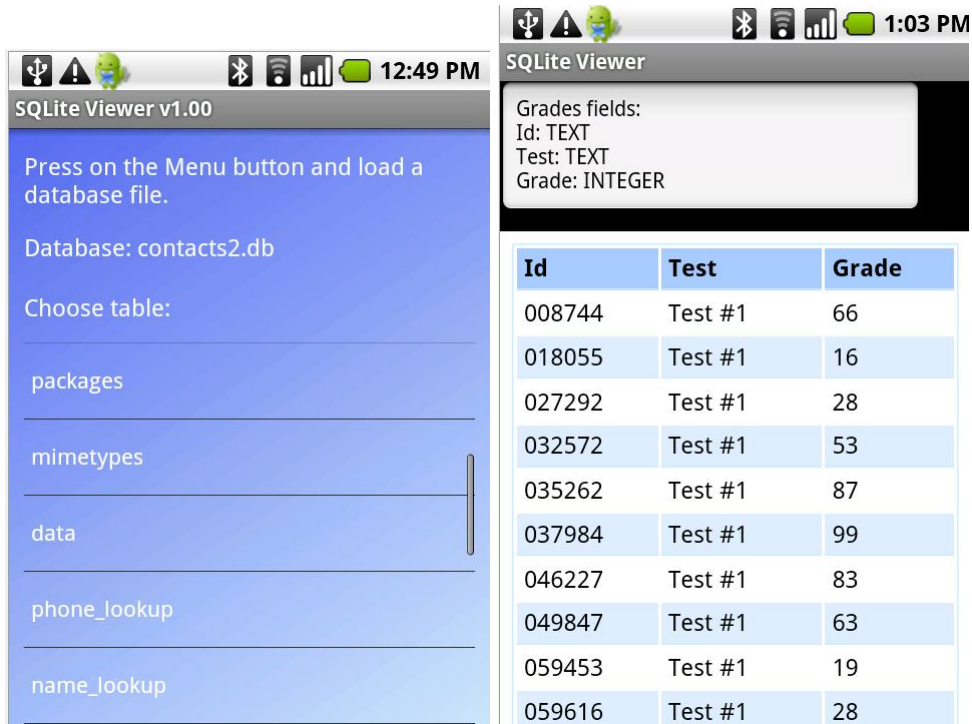
- If, at the beginning, you create the database in the code you can use the code below :

```
'check if the database already exists
If File.Exists(SQLDataBasePath, SQLDateBaseName) = True Then
  'if yes, initialize it
  SQL1.Initialize(SQLDataBasePath, SQLDateBaseName, True)
Else
  'if no, initialize it
  SQL1.Initialize(SQLDataBasePath, SQLDateBaseName, True)
  ' and create it
  SQLInit
End If
```



## 5.5 SQLite Viewer

There is a [SQLiteViewer](#) program in the forum, that allows you to load and display databases. The program uses the [DBUtils](#) module and the table is shown in a WebView view. The usage of the DBUtils module is explained in [chapter 4 DBUtils](#).



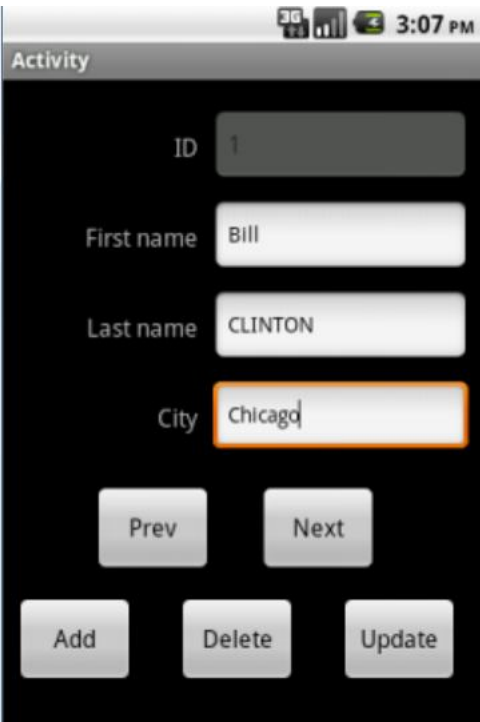
## 5.6 SQLite Database first simple example program SQLiteLight1

This example program is a very simple project with a very simple user interface.

The source code of this project is located in the *SourceCode\SQL\SQLiteLight1* folder.

The database name, the table name and the column names are hard coded, to make the code better readable. The user interface is kept very simple.

At the first run the database is empty and the user must add entries.



Following functions are implemented:

- Add an entry
- Edit an entry
- Update an entry
- Delete an entry
- Display next and previous entry

### Source code

It is selfexplaining.

### Program initialization:

### Process\_Globals

We dim the process global variables.

### Sub Process\_Global s

```
Public SQL1 As SQL
```

```
Public CurrentIndex = -1 As Int ' index of the current entry
```

```
Public RowNumber = 0 As Int ' number of rows
```

```
Public IDList As List ' list containing the IDs of the database
```

```
' we need it because the IDs can be different from the list indexes
```

```
' if we delete an entry its ID is lost
```

```
End Sub
```

## Globals

We dim all the views of the layout.

### Sub Global s

```
Private edtID, edtFirstName, edtLastName, edtCity As EditText
Private btnAdd, btnDelete, btnUpdate, btnPrevious, btnNext As Button
End Sub
```

## Activity\_Create

We check if the database already exists, initialize it, load it and show the first entry.

```
Sub Activity_Create(FirstTime As Boolean)
  If FirstTime Then
    ' File.Delete(File.DirInternal, "persons.db") ' only for testing, removes the
    database

    'check if the database already exists
    If File.Exists(File.DirInternal, "persons.db") = False Then
      'if not, initialize it
      SQL1.Initialize(File.DirInternal, "persons.db", True)
      'and create it
      CreateDataBase
    End If
  End If

  Activity.LoadLayout("Main")
End Sub
```

## Activity\_Resume

If the database is not initialized we initialize it, initialize the IDList list, read the database and show the first entry.

```
Sub Activity_Resume
  ' If the database is not initialized we initialize it
  If SQL1.IsInitialized = False Then
    SQL1.Initialize(File.DirInternal, "persons.db", True)
  End If

  IDList.Initialize           'initialize the ID list
  ReadDataBase               'read the database
  ShowEntry(0)              'show the first entry
End Sub
```

## Program closing :

### Activity\_Pause

If the program is closed by the user we close the database.

```
Sub Activity_Pause (UserClosed As Boolean)
  If UserClosed Then
    SQL1.Close           'if the user closes the program we close the database
  End If
End Sub
```

**Database handling :****Create the database**

We create the database with the four following columns :

- ID                    INTEGER PRIMARY KEY                    the ID of the entry.  
We use the INTEGER PRIMARY KEY data type, which is automatically incremented when we add a new entry.
- FirstName        the persons first name with TEXT data type.
- LastName        the persons last name with TEXT data type.
- City                the city where the person is living with TEXT data type.

**Sub CreateDataBase**

```
Private Query As String
```

```
Query = "CREATE TABLE persons (ID INTEGER PRIMARY KEY, FirstName TEXT, LastName TEXT, City TEXT)"
```

```
SQL1.ExecNonQuery(Query)
```

```
End Sub
```

**ReadDataBase**

We

- Define a Cursor and read IDs from the database.
- Check if there database is not empty.
- Fill IDList with the IDs of all entries.
- Set the current index to 0
- Close the cursor

Why do we use a List with the IDs ?

We use for the ID the INTEGER PRIMARY KEY data type which is unique.

If we delete an entry its ID is lost, which means that the ID numbers are not simply the row indexes but there can be 'holes' in the list.

**Sub ReadDataBase**

```
Private Row As Int
```

```
Private Cursor1 As Cursor
```

```
'We read only the ID column and put them in a List
```

```
Cursor1 = SQL1.ExecQuery("SELECT ID FROM persons")
```

```
If Cursor1.RowCount > 0 Then        'check if entries exist
```

```
    RowNumber = Cursor1.RowCount    'set the row count variable
```

```
    IDList.Initialize                'initialize the ID list
```

```
    For Row = 0 To RowNumber - 1
```

```
        Cursor1.Position = Row        'set the Cursor to each row
```

```
        IDList.Add(Cursor1.GetInt("ID")) 'add the ID's to the ID list
```

```
    Next
```

```
    CurrentIndex = 0                'set the current index to 0
```

```
End If
```

```
Cursor1.Close                        'close the cursor, we don't need it anymore
```

```
End Sub
```

## ShowEntry

We get the selected entry's ID from IDList, read the entry from the database, fill the EditText views and close the Cursor.

```
Sub ShowEntry(EntryIndex As Int)
    Private Cursor1 As Cursor
    Private ID As Int

    If IDList.Size = 0 Then 'check if the database is empty
        Return 'if yes leave the routine
    End If

    ID = IDList.Get(EntryIndex) 'get the ID for the given entry index
    'read the entry with the given ID
    Cursor1 = SQL1.ExecQuery("SELECT * FROM persons WHERE ID = " & ID)
    edtID.Text = ID 'display the ID
    Cursor1.Position = 0 'set the cursor
    edtFirstName.Text = Cursor1.GetString("FirstName") 'read the FirstName column
    edtLastName.Text = Cursor1.GetString("LastName") 'read the LastName column
    edtCity.Text = Cursor1.GetString("City") 'read the value of the City column
    Cursor1.Close 'close the cursor, we don't it anymore
End Sub
```

## AddEntry

We first check if an entry with the same data already exists.

If yes, we display a message.

If not, we add the new entry.

Display the new entries ID.

Close the cursor.

We use ExecQuery2 instead of ExecQuery, it's easier because we don't need to take care of the data type, the routine converts the data to the correct type.

The ? sign is a placeholder for the data which must be given in the array.

### Sub AddEntry

```

Private Query As String
Private Cursor1 As Cursor
Private ID As Int

'first we check if the entry already does exist
Query = "SELECT * FROM persons WHERE FirstName = ? AND LastName = ? AND City = ?"
Cursor1 = SQL1.ExecQuery2(Query, Array As String (edtFirstName.Text, edtLastName.Text,
edtCity.Text))

If Cursor1.RowCount > 0 Then
    'if it exists show a message and do nothing else
    ToastMessageShow("This entry already exists", False)
Else
    'if not, add the entry
    'a NULL for the ID column increments the primary key automatically by one
    'we use ExecNonQuery2 because it's easier, we don't need to take care of the data types
    Query = "INSERT INTO persons VALUES (NULL, ?, ?, ?)"
    SQL1.ExecNonQuery2(Query, Array As String(edtFirstName.Text, edtLastName.Text,
edtCity.Text))

    ToastMessageShow("Entry added", False)      ' confirmation for the user

    'to display the ID of the last entry we read the max value of the ID column
    ID = SQL1.ExecQuerySingleResult("SELECT max(ID) FROM persons")
    RowNumber = RowNumber + 1      'increase the row count
    IDList.Add(ID)                  'add the last ID to the list
    CurrentIndex = IDList.Size - 1 'set the current index to the last one
    edtID.Text = ID                 'display the last index
End If
Cursor1.Close                      'close the cursor, we don't it anymore
End Sub

```

## DeleteEntry

We ask the user if he really wants to delete the selected entry.  
If the answer is yes then we delete it.  
And set the new CurrentIndex.

### Sub DeleteEntry

```
Private Query As String
Private Answ As Int

'ask the user for confirmation
Answ = MsgBox2("Do you really want to delete " & edtFirstName.Text & " " &
edtLastName.Text, "Delete entry", "Yes", "", "No", Null)

If Answ = DialogResult.POSITIVE Then 'if yes, delete the entry
    Query = "DELETE FROM persons WHERE ID = " & IDList.Get(CurrentIndex)
    SQL1.ExecNonQuery(Query) 'delete the entry
    IDList.RemoveAt(CurrentIndex) 'remove the ID from the list
    If CurrentIndex = RowNumber - 1 Then 'if the current index is the last one
        CurrentIndex = CurrentIndex - 1 'decrement it by 1
    End If
    RowNumber = RowNumber - 1 'decrement the row count by 1
    ShowEntry(CurrentIndex) 'show the next entry
    ToastMessageShow("Entry deleted", False) 'confirmation for the user
End If
End Sub
```

## UpdateEntry

We use ExecNonQuery2 instead of ExecNonQuery because it's easier, we don't need to take care of the data type.  
The ? sign is a placeholder for the data which must be given in the array.

### Sub UpdateEntry

```
Private Query As String

Query = "UPDATE persons Set FirstName = ?, LastName = ?, City = ? _
WHERE ID = " & IDList.Get(CurrentIndex)
SQL1.ExecNonQuery2(Query, Array As String(edtFirstName.Text, _
edtLastName.Text, edtCity.Text))
ToastMessageShow("Entry updated", False)
End Sub
```

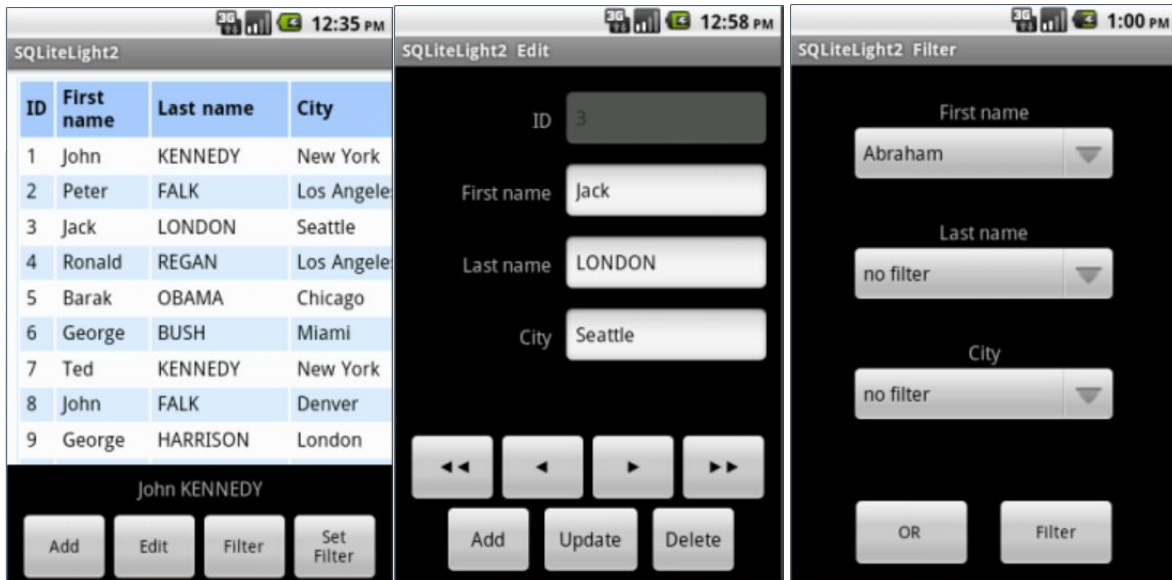
## 5.7 SQLite Database second simple example program SQLiteLight2

This example program is an evolution of the SQLiteLight1 project.

The source code of this project is located in the *SourceCode\SQL\SQLiteLight2* folder.

The program generates a default database if there is none available.

Added an Activity displaying the database in a table, using a WebView.



Main screen

Edit screen

Filter screen

Following functions are implemented:

- Add an entry
- Edit an entry
  - Update an entry
  - Delete an entry
  - Go to First, Prev, Next and Last entry
- Filter
  - AND / OR Boolean operator
  - Filter



## Parts of the source code

Not all the code will be explained here, only some more specific parts.

Since B4A version 5.2 the declaration of process global variables should be done in the Starter Service as explained in the [Starter Service](#) chapter.

## Starter Service

### Declaration of the Process global variables.

We declare all the process global variables in the Starter Service.

#### Sub Process\_Globals

```
Public SQL1 As SQL

Public CurrentIndex = -1 As Int           ' index of the current entry
Public RowNumber = 0 As Int              ' number of rows

Public IDList As List                    ' list containing the IDs of the database

' we need it because the IDs can be different from the list indexes
' if we delete an entry its ID is lost

' Variables for Edit
' the Edit Activity has two modes "Add" and "Edit"
Public EditMode = "Add" As String

' Variables for Filter
' variable for the filter query,
' defined in the btnFilter_Click routine
' and will be used in the DBWebView ShowTable routine
Public FilterQuery = "" As String

' flag for the filter active or inactive
Public flagFilterActive = False As Boolean

' Boolean operator used in the filter
' can be OR or AND
Public BooleanOperator = "OR " As String

' Variables used to hold the selected indexes of the Spinners
Public SelectedFirstName = 0 As Int
Public SelectedLastName = 0 As Int
Public SelectedCity = 0 As Int
End Sub
```

## Starter Service

### Initialization of the Process global variables.

#### Sub Service\_Create

```
File.Delete(File.Directory, "persons.db") ' only for testing, removes the database

' check if the database already exists
If File.Exists(File.Directory, "persons.db") = False Then
    ' copy the default DB
    File.Copy(File.DirectoryAssets, "persons.db", File.Directory, "persons.db")
    ' if not, initialize it
    SQLite.Initialize(File.Directory, "persons.db", True)
    ' and create it
    ' CreateDataBase
    ' copy the default DB
    File.Copy(File.DirectoryAssets, "persons.db", File.Directory, "persons.db")
Else
    ' if yes, initialize it
    SQLite.Initialize(File.Directory, "persons.db", True)
End If
End Sub
```

## Main Activity

### Declaration of the Global variables.

#### Sub Global s

```

Private wbvTable As WebView
Private btnAdd, btnEdit, btnFilter, btnSetFilter As Button
Private lblSelectedItem As Label

' used in ExecuteHTML
Private HtmlCSS As String
HtmlCSS = "table {width: 100%;border: 1px solid #cef;text-align: left; }" _
& " th { font-weight: bold; background-color: #acf; border-bottom: 1px _
solid #cef; }" _
& "td,th { padding: 4px 5px; }" _
& ".odd {background-color: #def; } .odd td {border-bottom: 1px solid #cef; }" _
& "a { text-decoration: none; color: #000;}"
End Sub

```

HtmlCSS is the string for the html call.

### Show the database in a WebView.

We define the SQL query.

Depending if the filter is active we add the filter query and change the filter button text.

We load the database query result in a WebView and read the database IDs.

'Shows the database in a table in a WebView

#### Sub ShowTable

```

Dim Query As String

Query = "SELECT ID, FirstName As [First name], LastName As [Last name], City FROM
persons"
'depending if the filter is active or not we add the filter query at the end of the
query
' the filter query is defined in the Filter Activity
If Filter.flagFilterActive = False Then
    btnFilter.Text = "Filter" 'change the text in the Filter button
Else
    Query = Query & Filter.Query
    btnFilter.Text = "UnFilter"'change the text in the Filter button
End If
'displays the database in a table
wbvTable.LoadHtml (ExecuteHtml (Starter.SQL1, Query, Null, 0, True))
ReadDataBaseIDs
End Sub

```

**Generate the html string used in *wbvTable.LoadHtml*.**

```

' This routine is extracted from the DBUtils code module
' Creates a html text that displays the data in a table.
' The style of the table can be changed by modifying HtmlCSS variable.
Sub ExecuteHtml (SQL As SQL, Query As String, StringArgs() As String, Limit As Int,
Clickable As Boolean) As String
    Private cur As Cursor
    If StringArgs <> Null Then
        cur = SQL.ExecQuery2(Query, StringArgs)
    Else
        cur = SQL.ExecQuery(Query)
    End If
    Log("ExecuteHtml: " & Query)
    If Limit > 0 Then Limit = Min(Limit, cur.RowCount) Else Limit = cur.RowCount
    Private sb As StringBuilder
    sb.Initialize
    sb.Append("<html><body>").Append(CRLF)
    sb.Append("<style type=' text/css' >").Append(HtmlCSS).Append("</style>").Append(CRLF)
    sb.Append("<table><tr>").Append(CRLF)
    For i = 0 To cur.ColumnCount - 1
        sb.Append("<th>").Append(cur.GetColumnName(i)).Append("</th>")
    Next

    sb.Append("</tr>").Append(CRLF)
    For row = 0 To Limit - 1
        cur.Position = row
        If row Mod 2 = 0 Then
            sb.Append("<tr>")
        Else
            sb.Append("<tr class=' odd' >")
        End If
        For i = 0 To cur.ColumnCount - 1
            sb.Append("<td>")
            If Clickable Then
                sb.Append("<a href=' http://'").Append(i).Append(". ")
                sb.Append(row)
                sb.Append(". com' >").Append(cur.GetString2(i)).Append("</a>")
            Else
                sb.Append(cur.GetString2(i))
            End If
            sb.Append("</td>")
        Next
        sb.Append("</tr>").Append(CRLF)
    Next
    cur.Close
    sb.Append("</table></body></html >")
    Return sb.ToString
End Sub

```

### Event handling of the WebView.

This routine is taken from the DBUtils module.

The URL variable hold the return value from the WebView event.

It could look like this `http://2.7.com/` where 2 is the col index and 7 is the row index.

The col and row values are extracted in `values = Regex.Split("[.]", Uri.SubString(7))`

`values(0)` holds the col value

`values(1)` holds the row value

`values(2)` holds the end of the string

```
'Routine from the DBUtils demo program
Sub wbvTable_OverrideUri (Uri As String) As Boolean
    'parse the row and column numbers from the URL
    Private values() As String
    values = Regex.Split("[.]", Uri.SubString(7))
    Private col, row As Int
    col = values(0)
    row = values(1)
    CurrentIndex = row
    UpdateSelectedItem
    Return True 'Don't try to navigate to this URL
End Sub
```

### Edit Activity.

In the Edit activity there is nothing really special.

### Filter Activity.

Process global variables are initialized in the Starter Service.

Most of the code is self explanatory.

For the data selection in the filter we use Spinners, these are filled with the data from the database. But as there can be multiple entries with the same data we fill them with 'distinct' data, one name is shown only once.

The code is shown for one Spinner only, the principle is the same for the others.

```
'Initialize the Spinners
Sub InitSpinners
    Private i As Int
    Private Query1 As String
    Private Curs As Cursor

    'We execute a query for each column and fill the Spinner
    'We use SELECT DISTINCT to have each existing first name in the database only once
    Query1 = "SELECT DISTINCT FirstName FROM persons"
    Query1 = "SELECT DISTINCT FirstName FROM persons ORDER BY FirstName ASC"
    Curs = Main.SQLite1.ExecQuery(Query1)
    'we add 'no filter' as no selection
    spnFirstName.Add("no filter")
    'we fill the Spinner with the data from the database
    For i = 0 To Curs.RowCount - 1
        Curs.Position = i
        spnFirstName.Add(Curs.GetString("FirstName"))
    Next
```

## 5.8 SQLite Database third simple example program SQLiteLight3

A third example program is in the *SourceCode\SQL\SQLiteLight3* folder.

This program is almost the same as SQLiteLight2, all functions are the same.

The differences are the database path, database name, table name, column number, column names, column alias names and column data types are variables instead being hard coded.

It allows also to generate a new database by:

- changing in Globals the values of the variables listed above
- in Activity\_Create
- comment this line: File.Copy(File.DirAssets, SQLDateBaseName, SQLDataBasePath, SQLDateBaseName)
- uncomment this line: CreateDataBase

The code has comments and is, I hope, self explanatory.

One example to show the difference:

For the query to show the table.

In SQLiteLight2 the names are hard coded:

```
Sub ShowTable
  Private i As Int
  Private Query As String
  Query = "SELECT ID, FirstName As [First name], LastName As [Last name], _
         City FROM persons"
```

In SQLiteLight3 the names are variables defined in Globals:

```
Sub ShowTable
  Private i As Int
  Private Query As String

  Query = "SELECT "
  For i = 0 To ColNumber - 1
    If i < ColNumber - 1 Then
      Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & "], "
    Else
      Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & " ] "
    End If
  Next
  Query = Query & " FROM " & SQLTableName
```

## 5.9 SQLite Database forth example program SQLiteLight4

This SQLite example program, SQLiteLight4, is a bit more elaborated than SQLiteLight2. In SQLiteLight2 there is only one table, in this program there are three tables. The purpose of this example is to show the principle of the management of several tables. To make the code easier readable all names are hard coded and not in variables like in SQLiteLight3.

The source code is in the *SourceCode\SQL\SQLiteLight4* folder.

The program manages a spare part stock. The tables are intentionally very simple with just a few columns and not all possible errors or mistakes a user can make are checked to keep the code simple and easier to read and understand.

The database has three tables:

- |             |                                       |                                    |                                   |
|-------------|---------------------------------------|------------------------------------|-----------------------------------|
| • Stock     | Number INTEGER,<br>number of products | ProductID INTEGER,<br>product ID   | Date INTEGER<br>date in Ticks     |
| • Products  | Name TEXT,<br>product name            | Price REAL,<br>product price       | SupplierID INTEGER<br>supplier ID |
| • Suppliers | Name TEXT,<br>suppliers name          | Address TEXT,<br>suppliers address | City TEXT<br>suppliers city       |

In the table Stock we use the ID of the product rather its name. The same in the table Products for the Supplier. The advantage is that we memorize a reference to the data in the original table instead of copying the data into another table. If we change once the data in the original table all the data in other tables are updated automatically.

Query example of a call for display:

```
Query = "SELECT Stock.Number, Products.Name AS Product, Suppliers.Name AS Supplier,
Products.Price AS Price, Stock.Number * Products.Price AS Value, date(Stock.Date /
1000, 'unixepoch') AS Date"
Query = Query & " FROM Stock, Products, Suppliers"
Query = Query & " WHERE Products.rowID = Stock.ProductID AND Suppliers.rowID =
Products.SupplierID"
```

We want to read following data:

- The number of items in stock `Stock.Number`.  
The *Number* column in the *Stock* table.
- The product name `Products.Name AS Product`.  
The *Name* column in the *Products* table and give this column the name 'Product'.
- The supplier name `Suppliers.Name AS Supplier`.  
The *Name* column in the *Suppliers* tabel and give this column the name 'Supplier'.
- The product price `Products.Price AS Price`.  
The *Price* column in the *Products* table and give this column the name 'Price'.
- The value of these products in stock `Stock.Number * Products.Price AS Value`.  
The multiplication of the number of items in stock with the product price and give this column the name 'Value'.
- The date when the product was entered `date(Stock.Date / 1000, 'unixepoch') AS Date`.  
We use the SQLite date function where we give the *Date* column of the *Stock* table.  
As the date is in B4A ticks we need to devide the value by 1000 to adapt it to 'SQL ticks' and we must add the parameter 'unixepoch' for 'SQL ticks'.

The query concerns the three tables *Stock*, *Products* and *Suppliers*:

```
FROM Stock, Products, Suppliers
```

We must add a WHERE expression:

- To connect the *Products* table *rowID* to the *Stock ProductID* column value.  
`Products.rowID = Stock.ProductID`
- To connect the *Suppliers* table *rowID* to the *Products SupplierID* column value.  
`Suppliers.rowID = Products.SupplierID`

Example of the result:



The screenshot shows an Android application interface with a blue header bar containing the text "SQLiteLight4 Stock". Below the header is a table with six columns: Number, Product, Supplier, Price, Value, and Date. The table contains three rows of data.

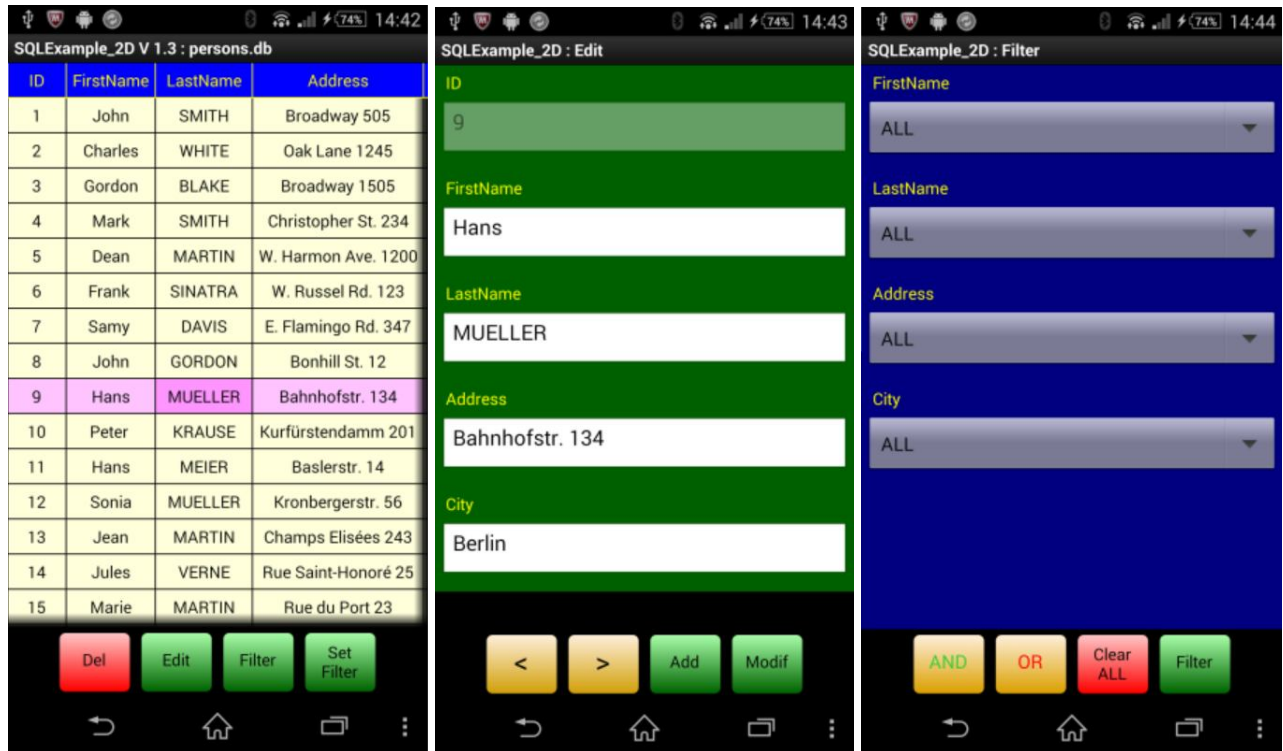
Number	Product	Supplier	Price	Value	Date
5	Break	Renault	350	1750	2004-08-19
10	Break	VW	175	1750	2010-02-07
15	Exhaust pipe	Ford	150	2250	2012-02-07



## 5.10 SQLite Database fifth example program

This SQLite example program is a more elaborated application with a small database with persons, with First name, Last name, Address and City as the persons' parameters.

The source code of this example is in the *SourceCode\SQL\SQLExample* project.



Database view

Edit activity

Filter activity

The layouts in the ScrollViews of the Edit and Filter activities are created automatically.

What we can do:

Ordering according to a given column

Clicking on a header sorts the database according to this column in ASC ascending mode.

Default or click on 


Click on 



ID	FirstName	LastName	Address
1	John	SMITH	Broadway 505
2	Charles	WHITE	Oak Lane 1245
3	Gordon	BLAKE	Broadway 1505
4	Mark	SMITH	Christopher St. 234
5	Dean	MARTIN	W. Harmon Ave. 1200
6	Frank	SINATRA	W. Russel Rd. 123
7	Samy	DAVIS	E. Flamingo Rd. 347
8	John	GORDON	Bonhill St. 12
9	Hans	MUELLER	Bahnhofstr. 134
10	Peter	KRAUSE	Kurfürstendamm 201
11	Hans	MEIER	Baslerstr. 14
12	Sonia	MUELLER	Kronbergerstr. 56
13	Jean	MARTIN	Champs Elisées 243
14	Jules	VERNE	Rue Saint-Honoré 25
15	Marie	MARTIN	Rue du Port 23

Click on  to delete the current data set.


Click on  to filter the database.

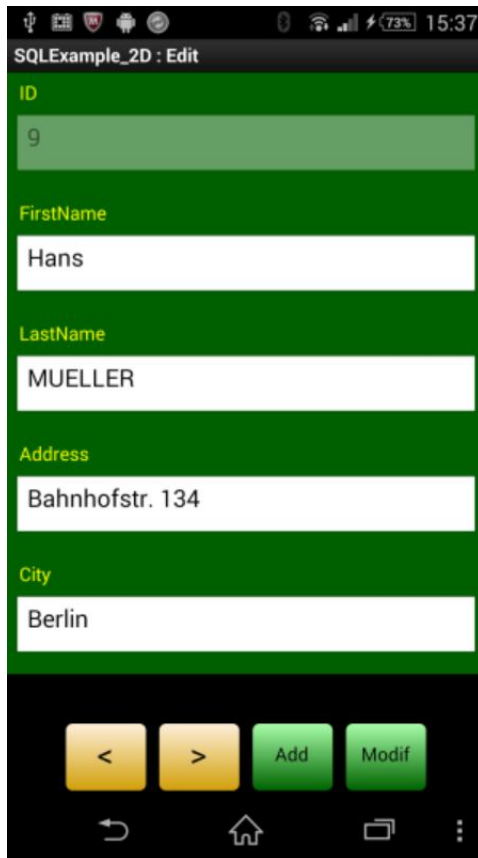
Click on  to show the Filter activity.

Click on  to edit the data set.

## 5.10.1 Editing



Select a data set in the main activity and click on  to display the Editor screen below.

A screenshot of an Android application screen titled "SQLiteExample\_2D : Edit". The screen has a green header and footer. The main content area has a green background with white text labels for fields: "ID" (value: 9), "FirstName" (value: Hans), "LastName" (value: MUELLER), "Address" (value: Bahnhofstr. 134), and "City" (value: Berlin). At the bottom, there is a black bar with four buttons: a yellow left arrow, a yellow right arrow, a green "Add" button, and a green "Modif" button. Below this bar is the Android navigation bar with icons for back, home, and recent apps.

Here we can: - Change values



Move to the previous data set



Move to the next data set.



Add the data set to the database.



Modify the current data set.





Go back to the main screen.

### 5.10.2 Filtering

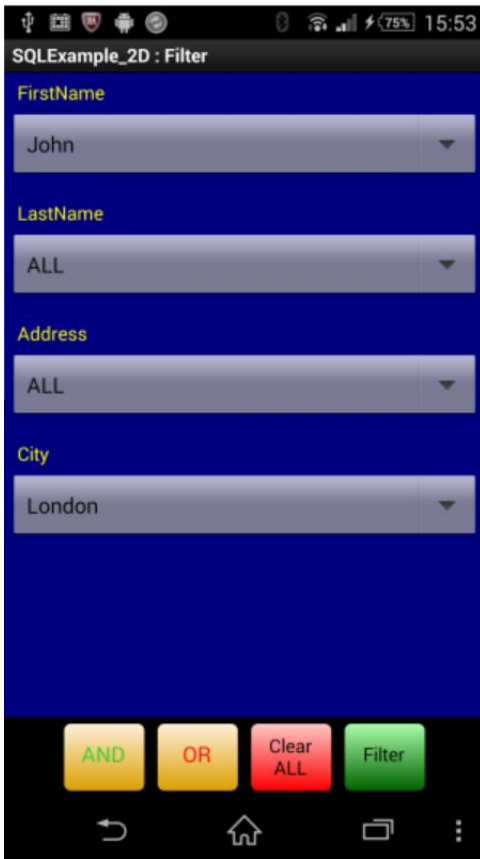


In the main view:

A click on  filters the database according to the filter parameters.

A click on  resets the database to unfiltered.

A click on  displays the Filter activity.



What can we do :

Enter different filtering parameters.

Select for example:

- FirstName John
- City London



Filtering AND function, green active function.



Filtering OR function.



Clear all parameters.



Filters the database.





Go back to the main screen.

FirstName	LastName	Address	City
John	GORDON	Bonhill St. 12	London

There exist only one person with the first name *John* AND living in *London*.



Click on  to change the AND parameter to OR and click on .

The active logical operator is displayed in green.

There are three persons with the first name of John OR living in London.

FirstName	LastName	Address	City
John	SMITH	Broadway 505	New Yo
Mark	SMITH	Christopher St. 234	London
John	GORDON	Bonhill St. 12	London

### 5.10.3 Code

Not all the code is explained in detail, only some parts needing a more detailed explanation. The code is, at least I hope so, well documented

The program has been updated taking advantage of recent features of Basic4Android. Example: Usage of 'anchors' in the layouts.

Let us define a simple database with persons. Each person has a certain number of parameters, called a data set or a database entry.

Person:

- First name
- Last name
- Address
- City

It is good practice to add an additional column, called 'ID' with a unique number to differentiate the data sets.

So the columns are: ID, FirstName, LastName, Address, City

Each column must be given a variable type: INTEGER, TEXT, REAL, BLOB .

In our database example we have following types:

```
ID          INTEGER PRIMARY KEY
FirstName   TEXT
LastName    TEXT
Address     TEXT
City        TEXT
```

### 5.10.3.1 Starter Service

#### In Process\_Global

We define the variables below:

```
Public ColumnName(NumberOfColumns) As String ' names of the columns
Public ColumnName(NumberOfColumns) As String ' names of the columns
```

#### And in Service\_Create

We define their values:

```
ColumnName(0) = "ID"
ColumnName(1) = "FirstName"
ColumnName(2) = "LastName"
ColumnName(3) = "Address"
ColumnName(4) = "City"

ColumnAliasName(0) = "ID"
ColumnAliasName(1) = "First name"
ColumnAliasName(2) = "Last name"
ColumnAliasName(3) = "Address"
ColumnAliasName(4) = "City"
```

ColumnName is obvious, the ColumnAliasNames are used to display the table headers, this is useful to give more meaningful names or to change the name in multilanguage programs.

### 5.10.3.2 Main Activity

#### In Activity\_Create

We initialize the RowID List which holds the IDs of the datasets of each row.

```
' initialize the RowID
If RowID.IsInitialized = False Then
    RowID.Initialize
End If
```

#### In Activity\_Resume

This code reads the table name of the database.

```
' gets the table name of the database
Private curs As Cursor
curs = SQL1.ExecQuery("SELECT name FROM sqlite_master WHERE Type='table'")
curs.Position = 0
DBTableName = curs.GetString("name")
curs.Close
```

The column widths are calculated in this routine:

```
'Calculates the width of the given column
Sub CalculateColumnWidth(HeaderName As String, Col umName As String) As Int
  Private Curs As Cursor
  Private row, MaxNumberOfChars As Int

  ' reads the max number of characters in the column
  MaxNumberOfChars = SQL1.ExecQuerySi ngl eResul t("SELECT max(length(" & _ Col umName &
")) FROM " & DBTabl eName)
  ' gets all rows with the max number of characters
  Curs = SQL1.ExecQuery("SELECT " & Col umName & " FROM " & DBTabl eName & " _ WHERE
length(" & Col umName & ") = " & MaxNumberOfChars)
  Curs.Positi on = 0
  Private TextMaxWidth As Int
  ' measures the text width in pixels of the header name
  TextMaxWidth = cvs.MeasureStri ngWi dth(HeaderName, Typeface.DEFAULT, FontSi ze)
  For row = 0 To Curs.RowCount - 1
    Curs.Positi on = row
    ' gets the max text width of the text
    TextMaxWidth = Max(TextMaxWidth, _
cvs.MeasureStri ngWi dth(Curs.GetStri ng(Col umName), Typeface.DEFAULT, FontSi ze))
  Next
  Curs.Close
  'returns the max text width + 10dip as margin
  Return TextMaxWidth + 10di p
End Sub
```

We :

- Read the maximum character of the data in each column.
- Read the rows with the longest content for each column.
- Calculate the width of header text.
- Calculate the width of the long column texts.
- Keep the maximum width value.
- Add a margin of 10dip.

### 5.10.3.3 Edit activity

Almost all the code is explained in the source file.

#### In the EditDispItem

We have this code:

```
' sets the Text properties
For col = 0 To Main.NumberOfColumns - 1
    Private edt As EditText
        edt = scvEdit.Panel.GetView(col * 2 + 1)
        edt.Text = Cursor1.GetString(Main.ColumnName(edt.Tag))
    Next
```

What does this `edt = scvEdit.Panel.GetView(col * 2 + 1)` mean ?

We want to get the view with the given index (`col * 2 + 1`) from the `ScrollView.Panel scvEdit.Panel`

In the `ScrollView.Panel` we have a `Label` and an `EditText` view per column.

The view index begins with 0 and is incremented by one each time we add a view.

So :

View	Column	Index
Label0	0	0
EditText0	0	1
Label1	1	2
EditText1	1	3
Label2	2	4
EditText2	2	5
Label3	3	6
EditText3	3	7

If we want to get the `EditText` view for column 3.

We calculate  $col * 2 + 1 = 3 * 2 + 1 = 7$  which is the index of the `EditText` view for column 3.

Therefore :

```
edt = scvEdit.Panel.GetView(col * 2 + 1)
```



### 5.10.3.4 Filter Activity

Almost all the code is explained in the source file.

For the Filter we use a Spinner for each column except for the first one, the ID column. The Spinners are filled with all DISTINCT values from each column.

The query looks like this:

```
SELECT DISTINCT ColumnName FROM TableName ORDER BY ColumnName ASC
```

And the code filling the Spinners:

```
' We get all DISTINCT values for each column  
' and fill the Spinners with these values  
Private txt As String  
txt = "SELECT DISTINCT " & Main.ColumnName(col) & " FROM " & _ Main.DBTableName & "  
ORDER BY " & Main.ColumnName(col) & " ASC"  
Curs = Main.SQLite1.ExecQuery(txt)  
spn.Clear  
spn.Add("ALL")  
For row = 0 To Curs.RowCount - 1  
    Curs.Position = row  
    spn.Add(Curs.GetString(Main.ColumnName(col)))  
Next
```

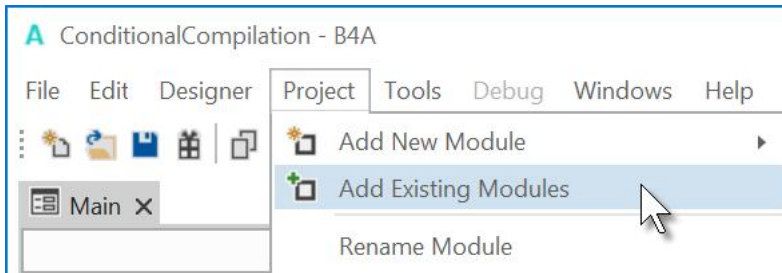
Another approach could have been to use EditText views, instead of the Spinners, to enter texts and check if there exist data with the given text or containing the given text in the given column.

## 6 DBUtils

For those who are not familiar with SQLite, Erel has written the DBUtils code module that should make things easier.

Note: DBUtils is a code module and not a library!

To use it, you must load the file DBUtils .bas to your project in the IDE menu Project / Add Existing Module. This will add the module to your project.



## 6.1 DBUtils functions

- CopyDBFromAssets**(FileName As String) As String  
 Copies a database file that was added in the Files tab. The database must be copied to a writable location because it is not possible to access a database located in File.DirAssets. This method copies the database to the storage card File.DirDefaultExternal. If the storage card is not available the file is copied to the internal folder File.DirInternal. The target folder is returned. If the database file already exists then no copying is done.
- CreateTable**(SQL As SQL, TableName As String, FieldsAndTypes As Map, PrimaryKey As String)  
 Creates a new table with the given name.  
 FieldsAndTypes - A map with the fields names as keys and the types as values. You can use the DB\_... constants for the types.  
 PrimaryKey - The column that will be the primary key. Pass empty string if not needed.
- DropTable**(SQL As SQL, TableName As String)  
 Deletes the given table.
- InsertMaps**(SQL As SQL, TableName As String, ListOfMaps As List)  
 Inserts the data to the table.  
 ListOfMaps - A list with maps as items. Each map represents a record where the map keys are the columns names and the maps values are the values.  
 Note that you should create a new map for each record (this can be done by calling Dim to redim the map).
- UpdateRecord**(SQL As SQL, TableName As String, Field As String, NewValue As Object, WhereFieldEquals As Map)  
 Updates a record in the database.  
 Field - Column name  
 NewValue - new value  
 WhereFieldEquals - Map where the map keys are the column names and the map values the values to look for.
- ExecuteMemoryTable**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int) As List  
 Executes the query and returns the result as a list of arrays.  
 Each item in the list is a strings array.  
 StringArgs() - Values to replace question marks in the query. Pass Null if not needed.  
 Limit - Limits the results. Pass 0 for all results.

- ExecuteMap**(SQL As SQL, Query As String, StringArgs() As String) As Map  
 Executes the query and returns a Map with the column names as the keys and the first record values As the entries values.  
 StringArgs() - Values to replace question marks in the query. Pass Null if not needed.  
 The keys are lower cased.  
 Returns Null if no results found.
- ExecuteSpinner**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, Spinner1 As Spinner)  
 Executes the query and fills the Spinner with the values in the first column.  
 StringArgs() - Values to replace question marks in the query. Pass Null if not needed.  
 Limit - Limits the results. Pass 0 for all results.
- ExecuteListView**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, ListView1 As ListView, TwoLines As Boolean)  
 Executes the query and fills the ListView with the value.  
 StringArgs()- Values to replace question marks in the query. Pass Null if not needed.  
 Limit - Limits the results. Pass 0 for all results.  
 If TwoLines is true then the first column is mapped to the first line and the second column is mapped to the second line.  
 In both cases the value set to the row is the array with all the records values.
- ExecuteJSON**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, DBTypes As List) As Map  
 Executes the given query and creates a Map that you can pass to JSONGenerator and generate JSON text.  
 StringArgs()- Values to replace question marks in the query. Pass Null if not needed.  
 Limit - Limits the results. Pass 0 for all results.  
 DBTypes - Lists the type of each column in the result set.  
 Usage example: (don't forget to add a reference to the JSON library)  

```
Dim gen As JSONGenerator
gen.Initialize(DBUtils.ExecuteJSON(SQL, "SELECT Id, Birthday FROM Students",
Null, 0, Array As String(DBUtils.DB_TEXT, DBUtils.DB_INTEGER)))
Dim jsonString As String
jsonString = gen.ToPrettyString(4)
Msgbox(jsonString, "")
```
- ExecuteHtml** (SQL As SQL, Query As String, StringArgs() As String, Limit As Int, Clickable As Boolean) As String  
 Creates a html text that displays the data in a table.  
 The style of the table can be changed by modifying HtmlCSS variable.  
 StringArgs() - Values to replace question marks in the query. Pass Null if not needed.  
 Limit - Limits the results. Pass 0 for all results.
- GetDBVersion**(SQL As SQL) As Int  
 Gets the current version of the database.  
 If the DBVersion table does not exist it is created and the current version is set to version 1.
- SetDBVersion**(SQL As SQL, Version As Int)  
 Sets the database version to the given version number.

## 6.2 Examples

You find Erels' example in the Forum under [DBUtils - Android databases are now simple](#). This example will not be explained in this chapter.

### 6.2.1 Example program Main module

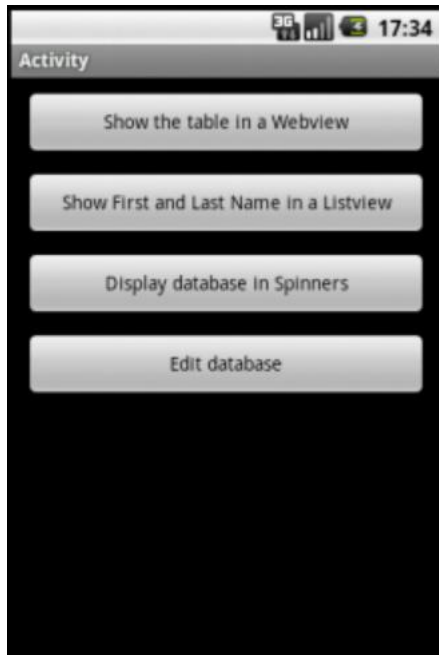
The example program in this chapter shows small examples of the DBUtils basics. The database used is persons.db from the SQLExample in the previous chapter.

The project, SQLDBUtils, is in the SourceCode folder.

The program has different activities showing each some of the DBUtils functions.

DBUtils function used :

- `DBUtils.CopyDBFromAssets`



- Show the table in a WebView
- Show the FirstName and LastName in a ListView
- Show the table in Spinners
- Edit the database

The code:

**Definition of the process global variables :** there are no global variables

```
Sub Process_Global_s
    Public DBFileName As String      : DBFileName = "persons.db"
    Public DBFileDir As String       : DBFileDir = File.DirDefaultExternal
    Public DBTableName As String     : DBTableName = "persons"

    Dim SQL1 As SQL
End Sub

Sub Global_s
End Sub
```

**Activity\_Create routine :**

```

Sub Activity_Create(FirstTime As Boolean)
  If FirstTime Then
    ' File.Delete(File.DirDefaultExternal, DBFileName) ' for testing

    If File.Exists(File.DirDefaultExternal, DBFileName) = False Then
      DBFileDir = DBUtils.CopyDBFromAssets(DBFileName)
    End If
    SQL1.Initialize(DBFileDir, DBFileName, True)
  End If

  Activity.LoadLayout("main")
End Sub

```

- If FirstTime = True, then we copy the database from the File.DirAssets directory to another folder if it doesn't already exist.  
For testing you could uncomment the line File.Delete... to delete the current database and replace it by the original one.  
By default, DBUtils tries to copy the database to File.DirDefaultExternal if this one is writable, if not then the database is copied to File.DirInternal.  
If the database already exists in the other folder nothing happens.
- Initialize the database.
- Load the `main` layout file for the main Activity.

**btnSelect routine :**

```

Sub btnSelect_Click
  Private btn As Button

  btn = Sender

  StartActivity(btn.Tag)
End Sub

```

All the Buttons have the same EventName `btnSelect` defined in the Designer.

The Button.Tag property, btn.Tag in the example, is the name of the Activity we want to start, also defined in the Designer.

The advantage is that we can add a new button to call a new Activity without the need to modify the code of the main module.

## 6.2.2 Show the table in a WebView

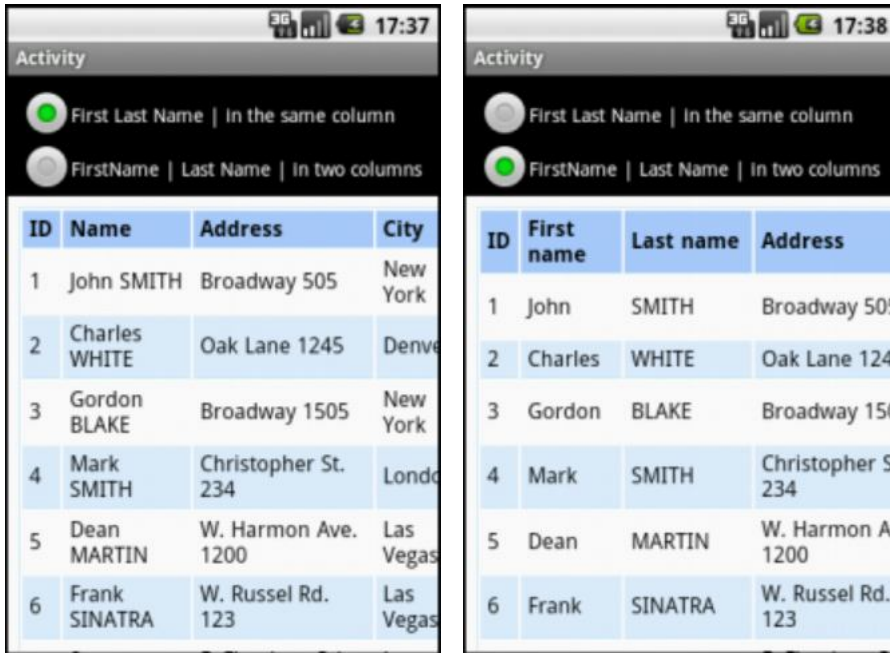
This example is the minimum needed to display a database, extracted from Erels' code.

DBUtils function used :

- `DBUtils.ExecuteHtml`

Two different displays are shown:

- First and last name in a same column with Name as column name.
- First and last name in two separate columns.



The code:

**Definition of the global variables :** There are no process global variables.

**Sub Global s**

```
Private WebView1 As WebView
Private rbtFirstNameLastName, rbtFirstNameLastName As RadioButton
End Sub
```

**Activity\_Create routine :**

```
Sub Activity_Create(FirstTime As Boolean)
    Activity.LoadLayout("webView")

    WebView1.Height=100%-(rbtFirstNameLastName.Top+ rbtFirstNameLastName.Height)
    ShowTableInWebView(0)
End Sub
```

We

- Load the activity layout.
- Set the WebView height, adapts for different height/width ratios.
- Display the database in the WebView.

**ShowTableInWebView routine :**

```

Sub ShowTableInWebView(Mode As Int)
  Private Query As String

  If Mode = 0 Then
    Query = "SELECT ID, [FirstName] || ' ' || [LastName] As Name, Address, City FROM "
    & Main.DBTableName
  Else
    Query = "SELECT ID, FirstName As [First name], LastName As [Last name], Address,
    City FROM " & Main.DBTableName
  End If
  WebView1.LoadHtml(DBUtils.ExecuteHtml(Main.SQL1, Query, Null, 0, True))
End Sub

```

This routine shows the database in a table in a WebView.

- Mode defines the display mode,  
0 = FirstName and LastName in one column.  
1 = FirstName and LastName in two separate columns.

The query texts need some explanations (second Query):

- **SELECT** SQL query keyword
- **ID** is the DB column name and the same column name is used in the header
- **FirstName As [First name]**
  - **FirstName** is the DB column name
  - **As** for alias, header name different from the column name
  - **[First name]** is the header name, the [ ] are needed because of the space between First and name
- **LastName As [Last name]**
  - **LastName** DB column name
  - **As** for alias, header name different from the column name
  - **[Last name]** header name
- **Address** DB column name and header name
- **City** DB column name and header name
- **FROM** SQL query keyword
- **DBTableName** table name, in this case a variable

Several DB columns can be combined to one table column, this is shown in this line :

```
txt = "SELECT ID, [FirstName] || ' ' || [LastName] As Name, Address As Address, City As
City FROM " & DBTableName
```

**[FirstName] || ' ' || [LastName] As Name** combines the values of the **[FirstName]** column with the value of the **[LastName]** column with a space ' ' between them and **Name** as the header name.

ID	Name	Address	City
1	John SMITH	Broadway 505	New York
2	Charles WHITE	Oak Lane 1245	Denv
3	Gordon BLAKE	Broadway 1505	New York

ID	First name	Last name	Address
1	John	SMITH	Broadway 505
2	Charles	WHITE	Oak Lane 124
3	Gordon	BLAKE	Broadway 150

First and last name in a same column.

First and last name in two separate columns.



The call of the WebView :

```
WebView1.LoadHtml (DBUtils.ExecuteHtml (SQL1, Query, Null, 0, True))
```

Where we call `DBUtils.ExecuteHtml` with :

- `SQL1` SQL object
- `Query` the SQL query
- `Null` no array
- `0` no limit
- `True` WebView clickable, if `True` the `WebView1_Overri deUrl` event will be raised.

### The WebView event routine:

This routine is called when a cell in the table is selected and returns a `ToastMessage` with the row and column index.

```
Sub WebView1_Overri deUrl (Url As String) As Boolean
    'parse the row and column numbers from the URL
    Private values() As String
    values = Regex.Split("[.]", Url.SubString(7))
    Private col, row As Int
    col = values(0)
    row = values(1)
    ToastMessageShow("User pressed on column: " & col & " and row: " & row, False)
    Return True 'Don't try to navigate to this URL
End Sub
```

The returned value of `Url` = `http://col.row.com/` Example : `http://1.3.com/`

We :

- Dim a string array `values`
- Split the `Url` string with `values = Regex.Split("[.]", Url.SubString(7))`  
`Url.SubString(7)` = `1.3.com/` substring beginning with the 7th character till the end.
- Dim `col` and `row` as Integers
- `col = values(0)` and `row = values(1)`
- Display a `ToastMessage` with the two values.
- Return `True` to consume the event, to not transmit it to the operating system.

**RadioButton CheckedChange event routine :**

```
Sub rbtSelectMode_CheckedChange(Checked As Boolean)
  Private rbt As RadioButton

  If Checked Then
    rbt = Sender
    ShowTableInWebView(rbt.Tag)
  End If
End Sub
```

The two RadioButtons have the same EventName `rbtSelectMode` defined in the Designer. The Tag properties are :

- 0 for *First and last name in the same column.*
- 1 for *First and last name in two separate columns.*

We :

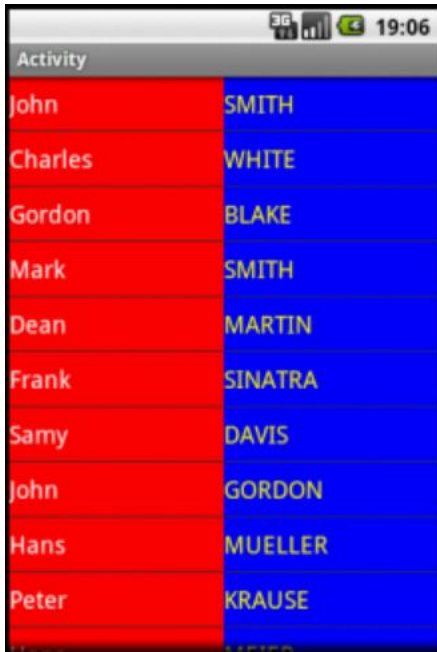
- Declare `rbt` as a local RadioButton.
- If `Checked = True` then we :
- Set `rbt = Sender`, the RadioButton that raised the event.
- Display the table according the selected display mode.

### 6.2.3 Show FirstName and LastName in a ListView

This activity shows how to display the content of two columns side by side in a ListView. The 'specialty' is that the two Labels for each item are side by side and not one on top the other.

DBUtils function used :

- `DBUtils.ExecuteListView`



The code:

**Definition of the global variables:** There are no process global variables.

**Sub Globals**

```
Private ListView1 As ListView
End Sub
```

**Activity\_Create routine :**

```
Sub Activity_Create(FirstTime As Boolean)
    Activity.LoadLayout("ListView")

    ListView1.Height = 100%

    ListView1.Init
    ListView1.Fill
End Sub
```

We

- Load the activity layout.
- Set the ListView height.
- Initialize the ListView properties.
- Fill the ListView.

**The ListViewInit routine :****Sub ListViewInit**

```

ListView1.TwoLinesLayout.ItemHeight = 40dip
ListView1.TwoLinesLayout.Label.Left = 0
ListView1.TwoLinesLayout.Label.Width = 50%x
ListView1.TwoLinesLayout.Label.Height = 40dip
ListView1.TwoLinesLayout.Label.Gravity = Gravity.CENTER_VERTICAL
ListView1.TwoLinesLayout.Label.Color = Colors.Red
ListView1.TwoLinesLayout.Label.TextSize = 18

ListView1.TwoLinesLayout.SecondLabel.Top = 0
ListView1.TwoLinesLayout.SecondLabel.Left = 50%x
ListView1.TwoLinesLayout.SecondLabel.Width = 50%x
ListView1.TwoLinesLayout.SecondLabel.Height = 40dip
ListView1.TwoLinesLayout.SecondLabel.Gravity = Gravity.CENTER_VERTICAL
ListView1.TwoLinesLayout.SecondLabel.Color = Colors.Blue
ListView1.TwoLinesLayout.SecondLabel.TextColor = Colors.Yellow
ListView1.TwoLinesLayout.SecondLabel.TextSize = 18

```

End Sub

We :

- Set the ItemHeight = 40dip
- Set the Left, Width, Height, Gravity, Color and TextSize properties for the first Label.
- Set the Top, Left, Width, Height, Gravity, Color, TextColor and TextSize properties for the second Label.

**The ListViewFill routine :****Sub ListViewFill**

```
Private Query As String
```

```
Query = "SELECT FirstName, LastName FROM " & Main.DBTableName
```

```
DBUtils.ExecuteListView(Main.SQL1, Query, Null, 0, ListView1, True)
```

End Sub

The Query :

- **SELECT** SQL keyword.
- **FirstName LastName** the two selected column names.
- **FROM** SQL keyword.
- **Main.DBTableName** the table name, in this case a process global variable

We call **DBUtils.ExecuteListView** with

- **Main.SQL1** the database
- **Query** the SQL query
- **Null** no string array
- **0** no limit
- **ListView1** the ListView view
- **True** for two lines ListView (False = one line ListView)

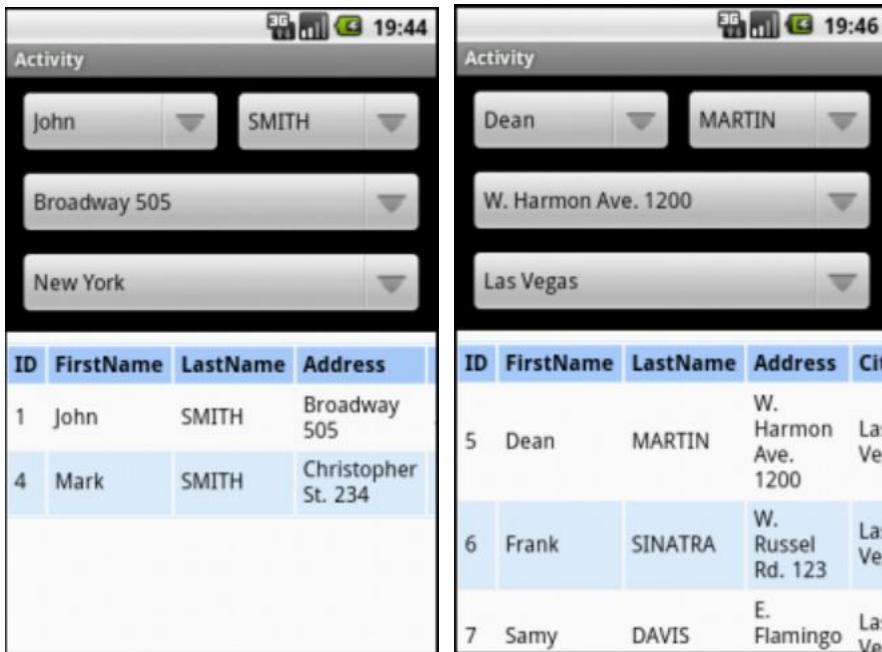
## 6.2.4 Display database in Spinners

In this activity we have one Spinner for each column.

Selecting an item in one of the Spinners sets the other Spinners to display the values of the same record and shows all records with the same selected value.

DBUtils function used :

- `DBUtils.ExecuteSpinner`
- `DBUtils.ExecuteHtml`



SMITH selected.

Las Vegas selected.

The code:

**Definition of the global variables :** There are no process global variables.

**Sub Globals**

```
Private spnFirstName, spnLastName, spnAddress, spnCity As Spinner
Private WebView1 As WebView
End Sub
```

**Activity\_Create routine :**

```
Sub Activity_Create(FirstTime As Boolean)

    Activity.LoadLayout("Spinner")

    WebView1.Height = 100% - (spnCity.Top + spnCity.Height + 8dip)
    FillSpinners
End Sub
```

We

- Load the activity layout.
- Set the WebView height, adapts for different height/width ratios.
- Fill the Spinners.

**The FillSpinners routine :****Sub FillSpinners**

```
Private Query As String
```

```
Query = "SELECT FirstName FROM " & Main.DBTableName
spnFirstName.Clear
DBUtils.ExecuteSpinner(Main.SQL1, Query, Null, 0, spnFirstName)
```

```
Query = "SELECT LastName FROM " & Main.DBTableName
spnLastName.Clear
DBUtils.ExecuteSpinner(Main.SQL1, Query, Null, 0, spnLastName)
```

```
Query = "SELECT Address FROM " & Main.DBTableName
spnAddress.Clear
DBUtils.ExecuteSpinner(Main.SQL1, Query, Null, 0, spnAddress)
```

```
Query = "SELECT City FROM " & Main.DBTableName
spnCity.Clear
DBUtils.ExecuteSpinner(Main.SQL1, Query, Null, 0, spnCity)
```

```
End Sub
```

We :

- Dim the Query variable.
- Define the Query for the `spnFirstName` Spinner.
- Call the `DBUtils.ExecuteSpinner` routine.
- `Main.SQL1` the SQL database
- `Query` the SQL query
- `Null` no string array
- `0` no limit
- `spnFirstName` the Spinner name

The Query for the FirstName Spinner :

- `SELECT` SQL keyword.
- `FirstName` the column name to read.
- `FROM` SQL keyword.
- `Main.DBTableName` the table name, in this case a process global variable.

The same principle is used for the other Spinners, the only differences are the column names and the Spinners.

## 6.2.5 Edit database

In this Activity we can modify a record and update the database.

DBUtils function used :

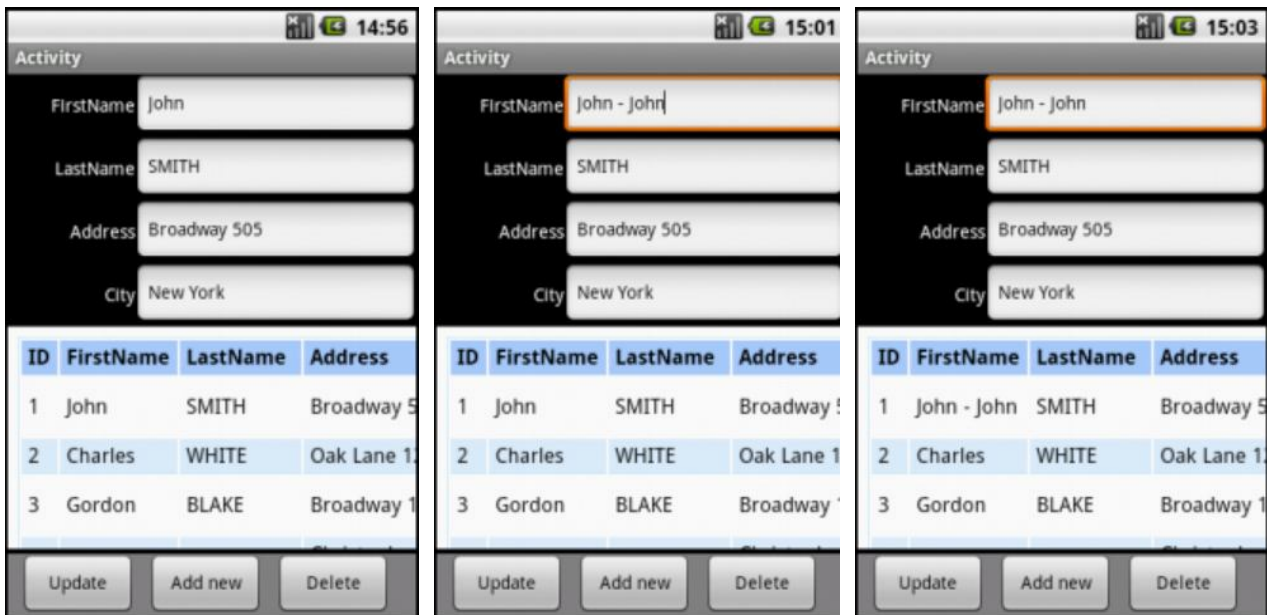
- `DBUtils.ExecuteHtml`
- `DBUtils.UpdateRecord`
- `DBUtils.ExecuteMemoryTable`
- `DBUtils.InsertMaps`
- `DBUtils.DeleteRecord`

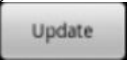
We have four EditText views for the record values, a WebView to display the database and three Buttons for updating, adding or deleting a record.

Following functions are available:

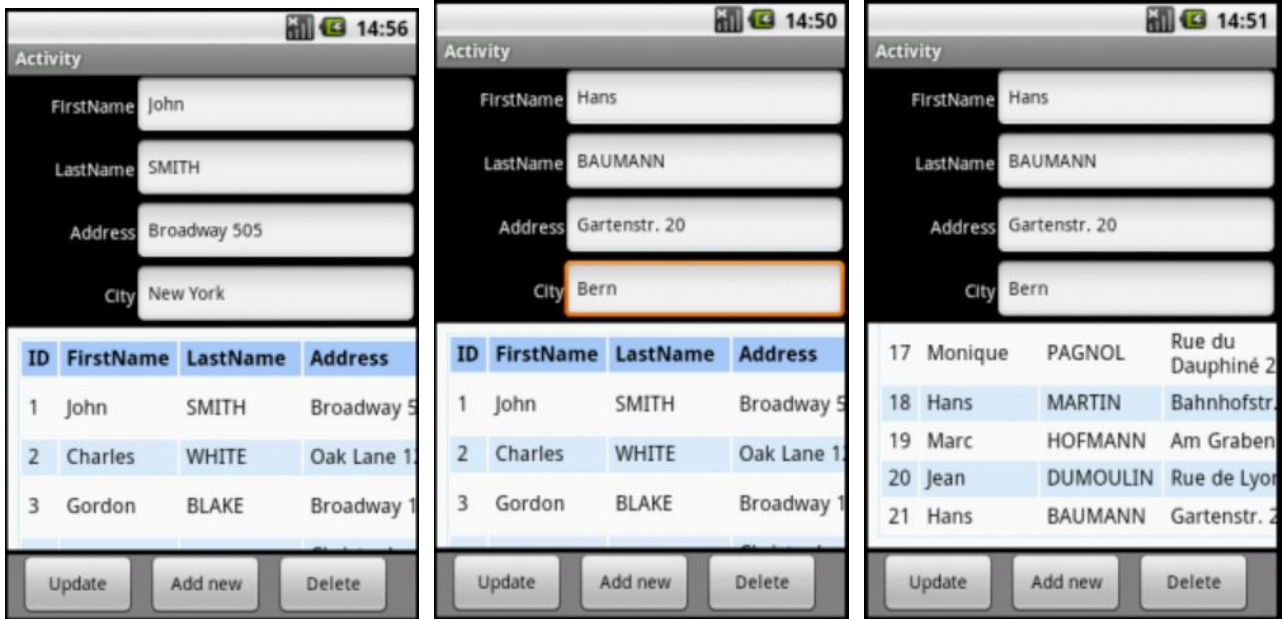
- Select a record in the WebView, the EditText views are updated.
- Update the selected record in the table with the new value in the EditText views.
- Add a new record to the table with the values in the EditText views.
- Delete the selected record.

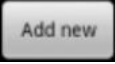
### Update a record



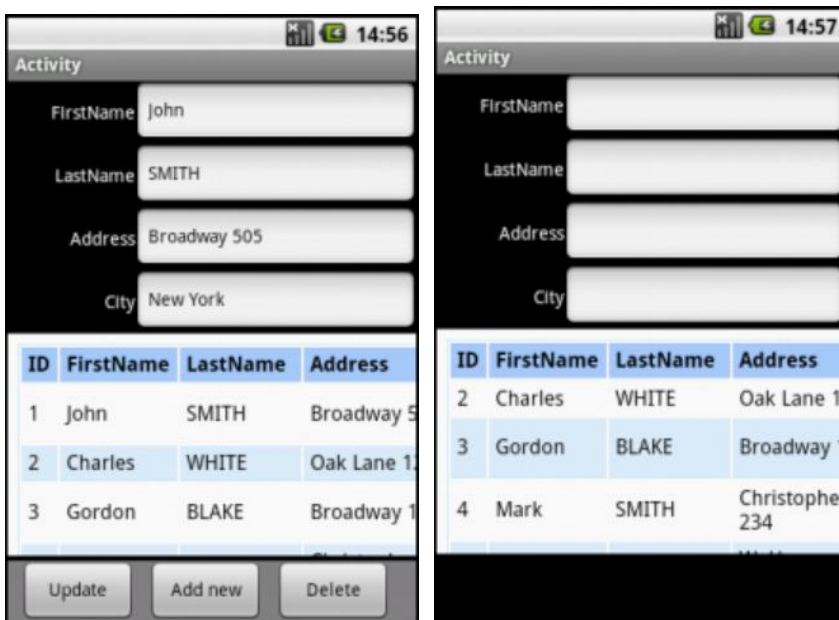
- Select a record in the WebView in the example SMITH . All the EditText views are updated.
- Change the first name from John to John-John.
- Click  to update the database, a message box asks for confirmation.
- The database in the WebView is updated.

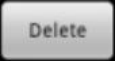
## Add a new record



- Select a line, the first one in this example
- Enter the new values in the EditText views.
- Click  to add the new record.
- The database in the WebView is updated.

## Delete a record



- Select a record in the WebView in the example SMITH .
- Click  to delete the record, a message box asks for confirmation.
- The database in the WebView is updated.
- The EditText views are empty.



The code:

**Definition of the global variables :** There are no process global variables.

**Sub Global s**

```
Private edtFirstName, edtLastName, edtAddress, edtCity As EditText
Private btnUpdate As Button
Private WebView1 As WebView
Private lstTable As List
Private pnlToolBox As Panel

Private col, row, ID As Int
End Sub
```

**Activity\_Create routine :**

```
Sub Activity_Create(FirstTime As Boolean)
    Activity.LoadLayout("edit")

    pnlToolBox.Top = 100%y - pnlToolBox.Height
    WebView1.Height = pnlToolBox.Top - (edtCity.Top + edtCity.Height + 4dip)

    FillWebView
    pnlToolBox.Visible = False
End Sub
```

We

- Load the activity layout.
- Set the btnUpdate Button Top property.
- Set the WebView height, adapts for different height/width ratios.
- Fill the WebView.
- Hide the buttons, ToolBox.

**The FillWebView routine :**

```
Sub FillWebView
    Private Query As String

    Query = "SELECT * FROM " & Main.DBTableName
    ' the line above does the same as the line below
    ' in bothe cases the WebView column names are the same as the DB column names
    ' Query = "SELECT ID, FirstName, LastName, Address, City FROM " & Main.DBTableName

    WebView1.LoadHtml(DBUtils.ExecuteHtml(Main.SQL1, Query, Null, 0, True))
    lstTable = DBUtils.ExecuteMemoryTable(Main.SQL1, Query, Null, 0)
End Sub
```

It's similar to the routine in chapter 4.2.2 Show the table in a WebView.

**The WebView event routine :**

This routine is called when a cell in the table is selected and returns a ToastMessage with the row and column index.

```
Sub WebView1_OverrideUrl (Url As String) As Boolean
    ' parse the row and column numbers from the URL
    Private values() As String
    values = Regex.Split("[.]", Url.SubString(7))
    col = values(0)
    row = values(1)

    Dim val(5) As String
    val = lstTable.Get(row)
    ID = val(0)
    edtFirstName.Text = val(1)
    edtLastName.Text = val(2)
    edtAddress.Text = val(3)
    edtCity.Text = val(4)
    pnlToolBox.Visible = True
    Return True 'Don't try to navigate to this URL
End Sub
```

The returned value of Url = http://col.row.com/ Example : http://1.3.com/

We :

- Dim a string array values
- Split the Url string with values = Regex.Split("[.]", Url.SubString(7))  
Url.SubString(7) = 1.3.com/ substring beginning with the 7th character till the end.
- Dim col and row as Integers
- col = values(0) and row = values(1)
- Display a ToastMessage with the two values.
- Return True to consume the event, to not transmit it to the operating system.

**The btnUpdate\_Click routine :**

```
Sub btnUpdate_Click
```

```
Private Answ As Int
```

```
Answ = MsgBox2("Do you really want to update this entry ?", "ATTENTION", "Yes", "", "No", Null)
```

```
If Answ = DialogResult.POSITIVE Then
```

```
Private mp As Map
```

```
mp.Initialize
```

```
mp.Put("ID", ID)
```

```
DBUtils.UpdateRecord(Main.SQL1, Main.DBTableName, "FirstName", edtFirstName.Text, mp)
```

```
DBUtils.UpdateRecord(Main.SQL1, Main.DBTableName, "LastName", edtLastName.Text, mp)
```

```
DBUtils.UpdateRecord(Main.SQL1, Main.DBTableName, "Address", edtAddress.Text, mp)
```

```
DBUtils.UpdateRecord(Main.SQL1, Main.DBTableName, "City", edtCity.Text, mp)
```

```
FillWebView
```

```
End If
```

```
End Sub
```

We :

- Ask in a MessageBox if the user really wants to update the record.
- If Yes, we Dim a Map to hold the ID of the record
- Put the ID of the record into the Map
- Update the FirstName field.
- Update the LastName field.
- Update the Address field.
- Update the City field.
- Update the WebView table

In the DBUtils.UpdateRecord we have :

- `Main.SQL1` the SQL reference of the database
- `Main.DBTableName` the table name, a variable in our case
- `FirstName` the column name
- `edtFirstName.Text` the new field value
- `mp` the Map representing the WHERE function of SQL, in our case WHERE ID = ID

**The btnAddNew\_Click routine :**

```
Sub btnAddNew_Click
    ' adds the new record
    If RecordExists = False Then
        Private maps As List
        Private mp As Map
        maps.Initialize
        mp.Initialize
        mp.Put("ID", Null)
        mp.Put("FirstName", edtFirstName.Text)
        mp.Put("LastName", edtLastName.Text)
        mp.Put("Address", edtAddress.Text)
        mp.Put("City", edtCity.Text)
        maps.Add(mp)
        DBUtils.InsertMaps(Main.SQL1, Main.DBTableName, maps)

        FillWebView
    End If
End Sub
```

We :

- Check if a record with the same data already exists.
- If No, we Dim a List of Maps maps and a Map mp.
- initialize maps and mp.
- put Null for the ID column, Null for autoincrement of the primary key.
- put edtFirstName.Text in the FirstName column.
- Same for LastName, Address and City.
- add the Map mp to the List maps.
- insert the new record in the database.
- update the WebView table.

**RecordExists routine :**

```

Sub RecordExists As Boolean
  Private Query As String
  Private curs As Cursor

  ' checks if the record already exists
  Query = "SELECT * FROM " & Main.DBTableName & _
    " WHERE FirstName='" & edtFirstName.Text & _
    "' AND LastName='" & edtLastName.Text & _
    "' AND Address='" & edtAddress.Text & _
    "' AND City='" & edtCity.Text & "'"
  curs = Main.SQL1.ExecQuery(Query)
  If curs.RowCount > 0 Then
    MsgBox("This record already exists", "A T T E N T I O N")
    Return True
  Else
    Return False
  End If
End Sub

```

We :

- Dim the Query and cursor variables.
- Define the Query to check if the values in the four EditTexts already exist.
  - SELECT \* FROM Table WHERE
  - FirstName = 'edtFirstName.Text' if 'edtFirstName.Text' exist in column FirstName.
  - LastName = 'edtLastName.Text' if 'edtLastName.Text' exist in column LastName.
  - etc.
- Execute the query.
- If the RowCount is heigher than 0 the record already exists and we Return True.
- If the RowCount is equal to 0 the record doesn't exist and we Return False.

**The btnDelete\_Click routine :****Sub btnDelete\_Click**

```
Private Answ As Int
```

```
Answ = MsgBox2("Do you really want to delete this record ?", "DELETE record", "Yes",
"", "No", Null)
```

```
If Answ = DialogResult.POSITIVE Then
```

```
Private mp As Map
```

```
Private val (5) As String
```

```
mp.Initialize
```

```
mp.Put("ID", ID)
```

```
val = lstTable.Get(row)
```

```
mp.Put("FirstName", val(1))
```

```
mp.Put("LastName", val(2))
```

```
mp.Put("Address", val(3))
```

```
mp.Put("City", val(4))
```

```
DBUtils.DeleteRecord(Main.SQL1, Main.DBTableName, mp)
```

```
FillWebView
```

```
edtFirstName.Text = ""
```

```
edtLastName.Text = ""
```

```
edtAddress.Text = ""
```

```
edtCity.Text = ""
```

```
row = -1
```

```
col = -1
```

```
pnlToolBox.Visible = False
```

```
End If
```

```
End Sub
```

We :

- Dim the Answ variable
- Ask the user if he really wants to delete the record.
- If the answer is yes (`DialogResponse.POSITIVE`) then :
- Dim mp as a Map and val (5) as String.
- Initialize the Map mp .
- Put the ID value to the Map.
- Set the values of the current row `lstTable.Get(row)` to val .  
The content of the current row is the string array `lstTable.Get(row)`.
- Put the values of the columns to the Map.
- Execute the Query `DBUtils.DeleteRecord(Main.SQL1, Main.DBTableName, mp)`.
- Run the `FillWebView` routine to update the table.
- Set the content of the EditText views to empty.
- Set row and col to -1 no row nor column selected.
- Hide the button toolbox, no function available.

## 7 GPS

The GPS library has three objects:

- GPS
- GPSSatellite
- Location

The example program will show several functions of the GPS library and has following functions.

- Connecting the GPS
- Getting and displaying GPS information
- Saving a GPS path
- Display the available satellites
- Showing Google maps
- Show a GPS path on the map

### 7.1 GPS Library

The GPS Library is part of the basic Basic4Android language.

#### 7.1.1 GPS Object

The GPS object has:

- Members
  - GPSEnabled as Boolean [read only]  
Tests whether the user has enabled the GPS or not
  - Initialize (eventName As String)  
Initializes the GPS with its eventName
  - LocationSettingsIntent As android.content.Intent [read only]  
Returns the intent that is used to show the global location settings.
  - Start(minimumTime As Long, minimumDistance As Float)  
Starts listening for events.
 

minimumTime	The shortest period (in milliseconds) between events. Pass 0 for highest frequency
minimumDistance	The shortest change in distance (in meters) for which to raise events. Pass 0 for highest frequency.
  - Stop  
Stops listening to the GPS. You will usually want to call Stop inside Sub Activity\_Pause.
- Events
  - LocationChanged (location1 As Location)  
Raised when a new 'fix' is ready.
  - UserEnabled (enabled As Boolean)
  - GpsStatus (satellites As List)
  - NMEA (National Marine Electronics Association) returns [NMEA data sentences](#).

### 7.1.2 GPS Satellite

The GPSSatellite object holds various information about a GPS satellite. A List with the available satellites is passed to the GpsStatus event.

Satellite data:

- Azimuth      0 - 360 degrees
- Elevation    0 - 90 degrees
- Prn (Pseudo random number)
- Snr (Signal / noise ratio)
- UsedInFix    True if the satellite is used to determine the current fix.

### 7.1.3 GPS Location

A Location object holds various information about a specific GPS fix (position).

In most cases you will work with locations that are passed to the GPS LocationChanged event.

The location object can also be used to calculate distance and bearing to other locations.

The most useful properties.

- Location1.Latitude      latitude of the fix      in [°]
- Location1.Longitude    longitude of the fix    in [°]
- Location1.Altitude      altitude of the fix      in [m]  
This is the altitude above the WGS 84 reference ellipsoid not the altitude above sea level.
- Location1.Bearing      bearing of the fix      in [°]
- Location1.Speed        speed of the fix        in [m/s]
- Location1.Time         time of the fix in      in [ticks]

The most useful methods.

- Location1.Initialize  
Initializes an empty location.
- Location1.Initialize2 (Latitude As String, Longitude As String)  
Initializes a location with the two given values, all the other properties are 0.
- Location1.DistanceTo (TargetLocation As Location)      in [m]  
Location1.DistanceTo(Location2)  
calculates the distance between Location1 and Location2.
- Location1.BearingTo (TargetLocation As Location)      in [°]  
Location1.BearingTo(Location2)  
calculates the bearing from Location1 to Location2.



## 7.1.4 NMEA data sentences

Unfortunately the altitude given in the Location object in the LocationChanged event is not the altitude above sea level but the altitude above the WGS 84 reference ellipsoid.

The difference can be up to 50 meters.

To get the altitude above sea level we can use the NMEA event.

Sub GPS1\_NMEA (TimeStamp As Long, Sentence As String)

TimeStamp is

Sentence represents different data sentences returned by the event.

More detailed information about the different sentences in this [LINK](#).

In the [GGA sentence \(Fix information\)](#) we get following information shown with a sentence example:

```
$GPGGA, 123519, 4807.038, N, 01131.000, E, 1, 08, 0.9, 545.4, M, 46.9, M, , *47
```

Where:

GGA	Global Positioning System Fix Data
123519	Fix taken at 12:35:19 UTC
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
1	Fix quality: 0 = invalid
	1 = GPS fix (SPS)
	2 = DGPS fix
	3 = PPS fix
	4 = Real Time Kinematic
	5 = Float RTK
	6 = estimated (dead reckoning) (2.3 feature)
	7 = Manual input mode
	8 = Simulation mode
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9,M	Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number
*47	the checksum data, always begins with *

You can use the code below to get the altitude above sea level.

```
Sub GPS1_NMEA (TimeStamp As Long, Sentence As String)
  If Sentence.SubString2(0, 6) = "$GPGGA" Then
    Private vals() As String
    vals = Regex.Split(",", Sentence)
    GPSAltitudeSeaLevel = vals(9)
  End If
End Sub
```

GPSAltitudeSeaLevel is a global variable : Dim GPSAltitudeSeaLevel As Double

## 7.2 GPS Program

The GPS example program shows several possibilities of the GPS library and has the following functions :

- Connect the GPS
- Get and display GPS information
- Save a GPS path
- Display the available satellites
- Display Google maps
- Display a GPS path on the map

Google map functions, user settable :

- Display zoom control
- Display scale control
- Display map type control
- Display a path
- Display markers
- Move a marker
- Display coordinates (touch the screen)
- Move the map / Display coordinates (touch the screen and move)

It is designed for smartphones and works only in portrait mode.

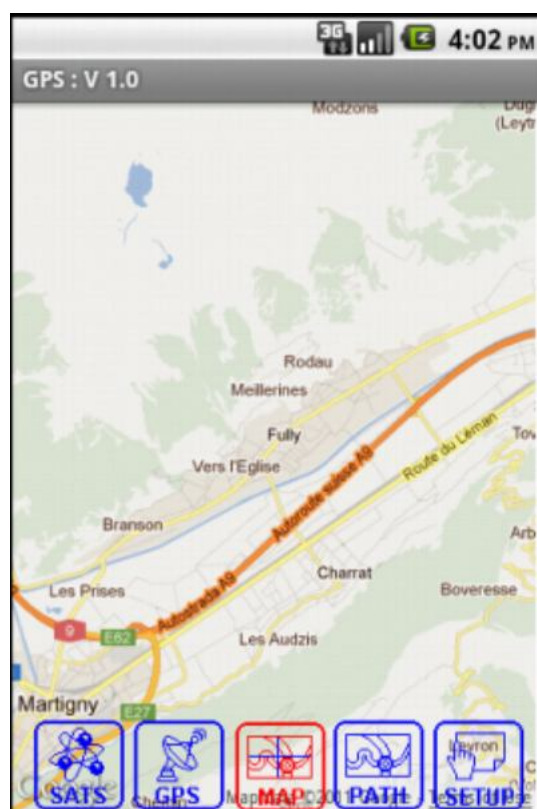
It is only available for users who bought Basic4Android, the program takes advantage of libraries which are not part of the trial version.

**The source code of this program is available in the users forum in the [GPSExample](#) project.** It is not in the SourceCode directory of this guide to guarantee the latest version of the program.

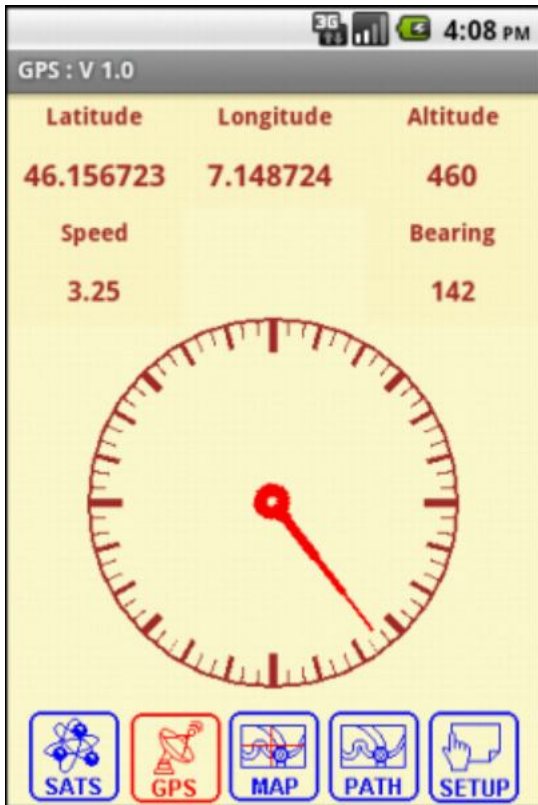
Main screen



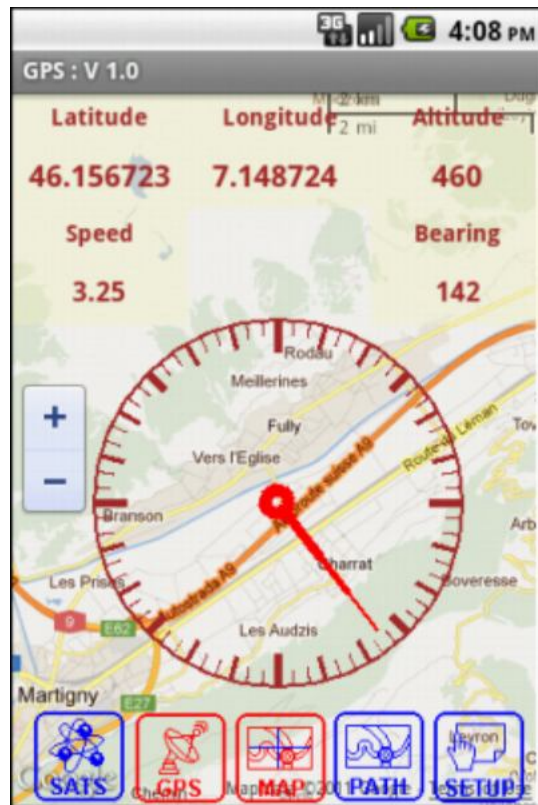
Google Maps



GPS display



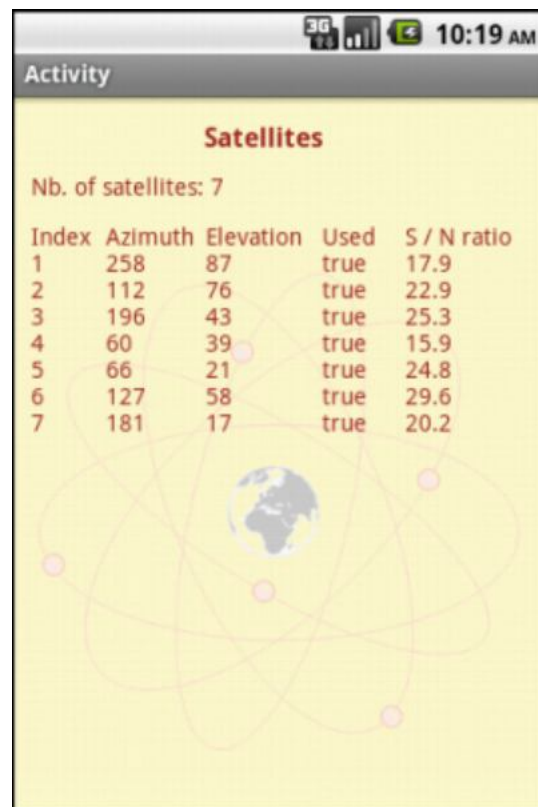
GPS display plus map



Setup



Satellites



GPS path data

GPS path display

File Test1.GPP 13 pnts

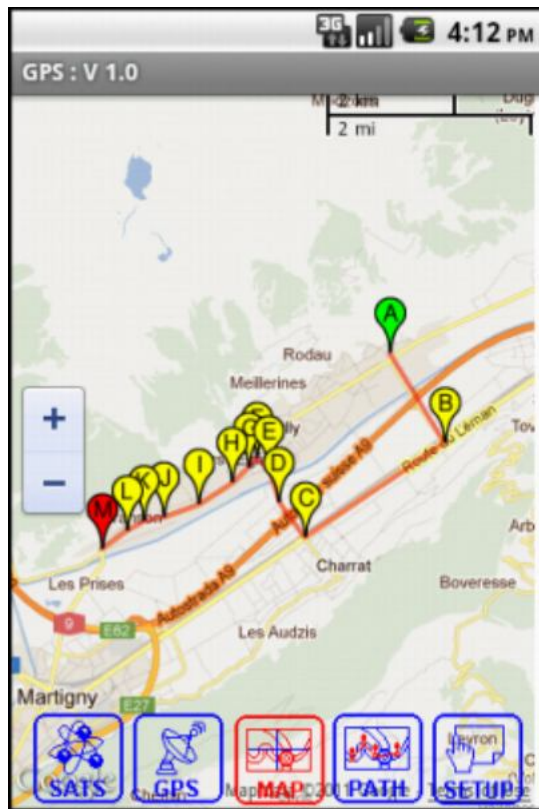
Comment GPS default path test

Date / Time 2011.12.10 12:00:56

ID	Latitude [°]	Longitude [°]	Altitude [m]	Time [s]
0	46.153322	7.145578	460.00	0.0
1	46.140693	7.156977	459.00	120.0
2	46.126944	7.128119	458.00	320.0
3	46.132111	7.122914	457.00 </td <td>330.0</td>	330.0
4	46.136458	7.120715	456.00	340.0
5	46.138468	7.118115	456.00	350.0
6	46.13704	7.116468	456.00	360.0

DEL LOAD SAVE FILTER

Map with GPS path



## 7.2.1 General explanations

Main screen buttons:



Displays Google maps centered on the default coordinates (defined in the setup).  
If there is a GPS path, this one is displayed, centered and zoomed to show the whole path.



Displays the setup screen.



Activates the GPS and memorizes the fixes depending on the setup.



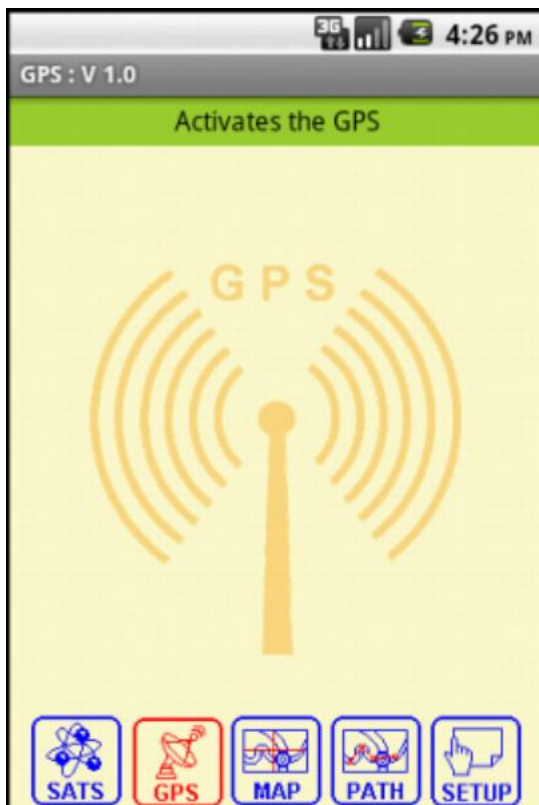
Shows the available satellites.



Displays the data of a GPS path.



For certain functions, the button color is red when they are activated



Tooltips for the buttons.

When you touch a button the tooltip is displayed on top of the screen.

When you release it, the tooltip is hidden.

The function is executed when you release the button inside the buttons area.

If you release the button outside its area the function is not executed, this allows the user to look at the buttons function without executing it.

## 7.2.2 Setup



The setup screen allows to define setup parameters for the program.



### GPS

- min time between two fixes.
- min distance between two fixes.

Enable display of:

- Speed
- Bearing

- Windrose

GPS path units, for the display of the

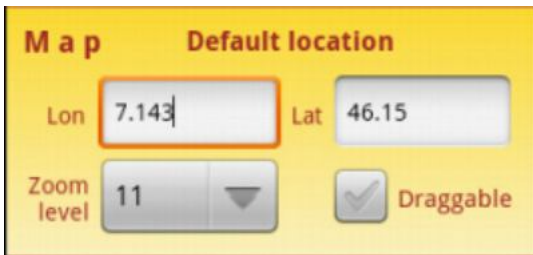
- unit for Altitude m and ft
- unit for Speed m/s, km/h and



values.

mile/h

- unit for Distance m, km and mile



### Map Default location

- Latitude
- Longitude
- Zoom level

- Draggable Checked means: the map can be moved  
Unchecked means: show the coordinates.



### GPS

- Show the current GPS location on the map.
- Save the GPS path when the GPS is stopped.

- Draw the

GPS path online (not yet implemented)

Map show different controls, marker/line

- Display center marker
- Display GPS path markers
- Display the GPS path polyline

- Scale control

- Type control, map ROADMAP or

- Zoom control DEFAULT, SMALL,



properties

SATELLITE

LARGE



### GPS Polyline properties

- Width in pixels
- Color
- Opacity 0 = transparent

properties



GPS Marker

- Marker clickable (not yet implemented)
- Marker draggable

### 7.2.3 GPS display



GPS display.

When activated, the GPS displays following parameters:

- Latitude
  - Longitude
  - Altitude
  - Speed
  - Bearing
  - Windrose
- user selectable in the setup
- user selectable in the setup
- user selectable in the setup

The minimum time and minimum distance to raise a fix change can be set in the setup screen. Values of 0 for both parameters give the fastest acquisition frequency.

The number of memorized fixes is displayed in the titlebar.



The map can be displayed at the same time with the current GPS location.

After stopping the GPS the user is asked to save the path giving a file name and a comment.



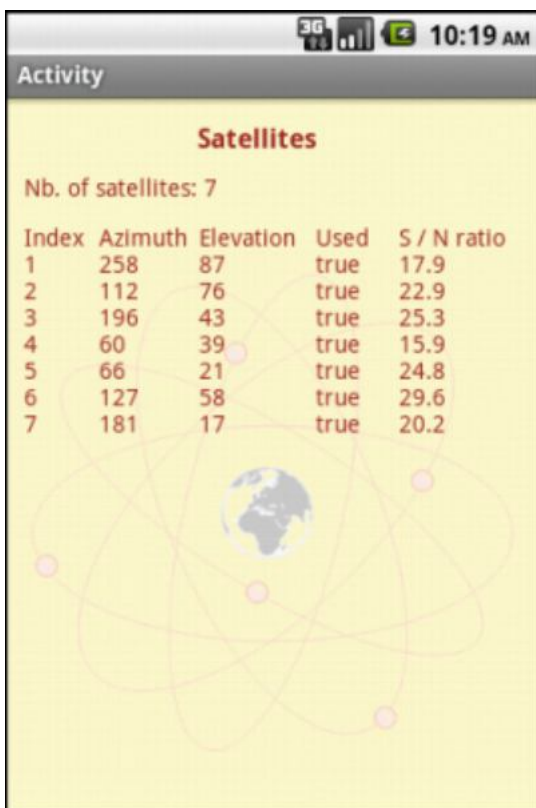
## 7.2.4 Satellites



The Satellites screen displays the information about the satellites currently received by the GPS.

The displayed data are:

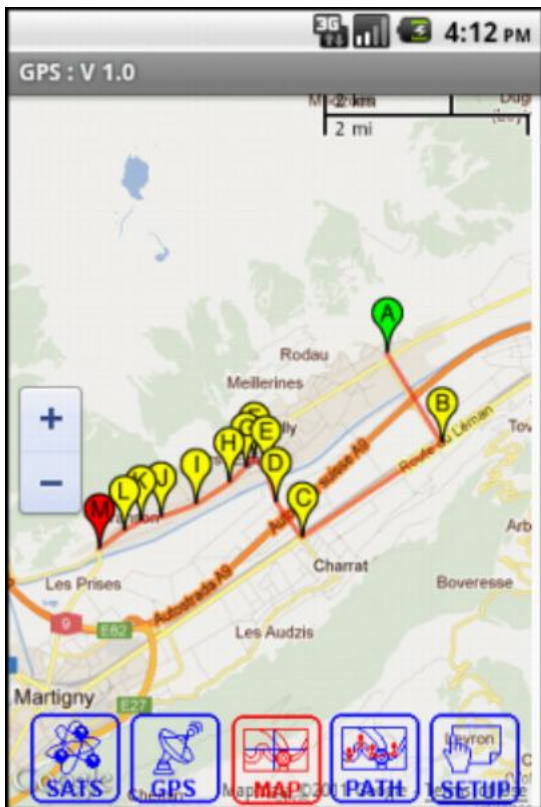
- Azimuth
- Elevation
- Used
- Signal/Noise ratio



## 7.2.5 Map display



The Google map can be displayed on the main screen.



with markers



without markers

The following elements can either be displayed or hidden, set in the Setup screen :

- Type control           MAP or SATELLITE
- Scale control
- Zoom control
- GPS path (polyline)
- Markers



The map can be zoomed with a 'double click'.  
Or with the zoom button.



If the map is 'Draggable' then :

- touching and moving (dragging) moves the map.



If the map is NOT 'Draggable' then

- touching the screen shows the coordinates of that position.
- touching and moving (dragging) shows the coordinates



Changing settings directly when the map is displayed:

Touching the MAP button shows a second button on top of it.

Moving up into the area of the second button, this one becomes red, and releasing it changes the map mode

and versa.

from draggable to coordinates and vice

Touching the PATH button shows three buttons on top of it.

Moving on one of these buttons allows to either:

- Polyline of the path with the markers.
- Polyline of the path without the markers.
- No polyline and no markers.



more

choose

**7.2.6 GPS path**



The data from the GPS can be memorized and saved in files.

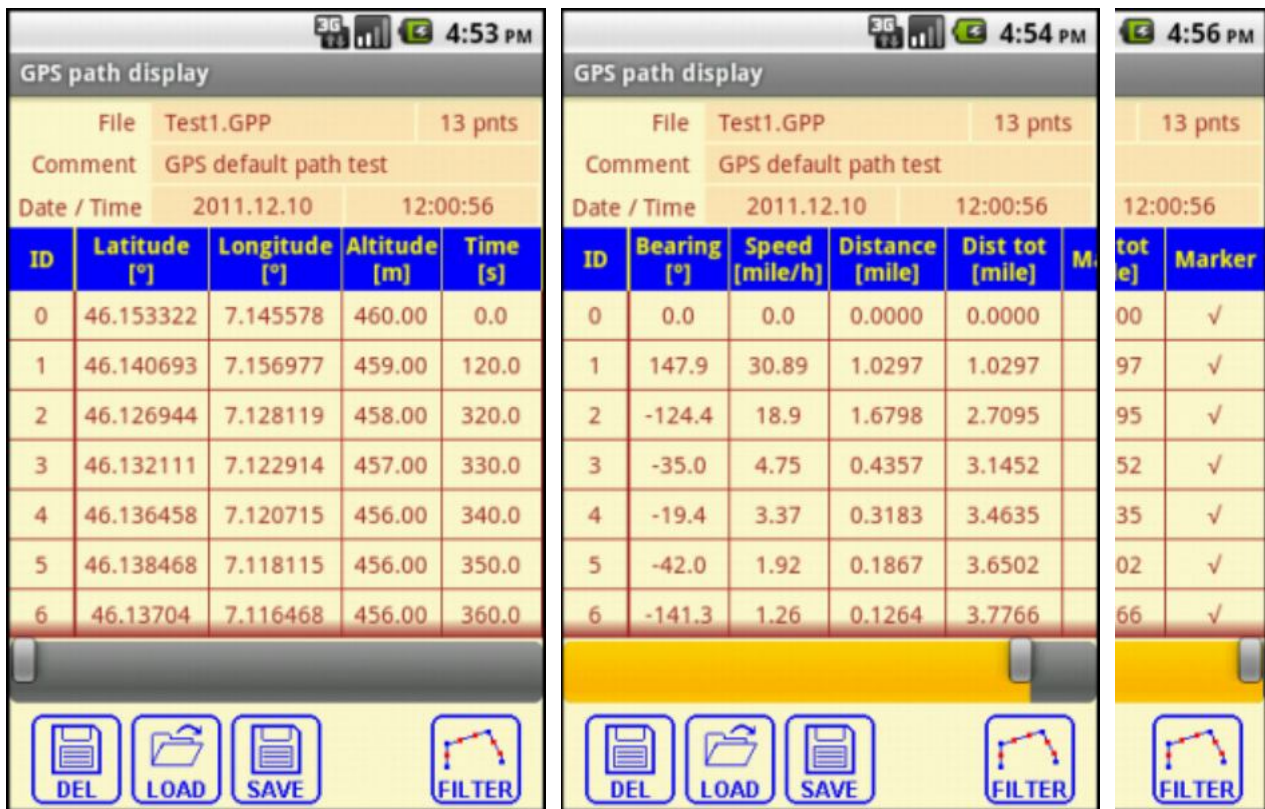
General data for the file:

- The filename
- The number of points in the file
- A comment for the path

The variables of a GPS fix are:

- Latitude the latitude of the fix [°]
- Longitude the longitude of the fix [°]
- Altitude the altitude of the fix [m]
- Time the time when the fix was taken [tick]
- Bearing the bearing from the previous fix [°]
- Speed the speed between the two fixes [m/s]
- Distance the distance between the two fixes [m]
- Dist tot the total distance from the first fix [m]
- Marker flag if a marker should be shown [-]

The GPS path screen shows the data of the selected GPS path.



The data can be scrolled vertically, normal ScrollView scrolling, and horizontally with the slider. The left column with the ID remains always visible.

You can:



Delete the selected file.



Load a GPS path file.



Save the GPS path file.



Delete the selected row.



Filter a path

Clicking on a row selects it or unselects it.

When a row is selected :

- it is highlighted in red.
- a Delete button is displayed allowing to delete this fix.  
When a fix is deleted the Speed, Bearing, Distance and Dist Tot values are updated.

Clicking on one of the headers below changes the unit of the displayed values.

Altitude		Speed			Distance / Dist tot				
m	ft	m/s	km/h	mile/h	m	km	mile	Marker	Marker
Altitude [m]	Altitude [ft]	Speed [m/s]	Speed [km/h]	Speed [mile/h]	Distance [m]	Distance [km]	Distance [mile]		
460.00	1509.2	0.00	0.00	0.00	0.0	0.0000	0.0000	✓	✓
459.00	1505.9	13.81	49.71	30.89	1657.1	1.6571	1.0297	✓	✓
458.00	1502.6	8.45	30.41	18.90	2703.5	2.7035	1.6798	✓	-
457.00	1499.3	2.12	7.65	4.75	701.2	0.7012	0.4357	✓	-
456.00	1496.1	1.51	5.42	3.37	512.2	0.5122	0.3183	✓	-
456.00	1496.1	0.86	3.09	1.92	300.5	0.3005	0.1867	✓	✓
456.00	1496.1	0.57	2.03	1.26	203.4	0.2034	0.1264	✓	✓

check / uncheck  
Marker

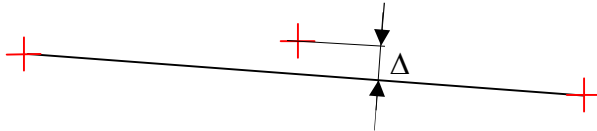
Clicking on a marker cell changes between checked and unchecked.

**GPS path filtering.**

When memorizing GPS paths it often happens that there are some point aligned along a straight line. These points can be removed.

The principle of the program is the following.

- The program looks at 3 successive points.
- Calculates the distance  $\Delta$  of the midpoint out of the line between the the two outer points.
- If this distance  $\Delta$  is higher than the predefined value the point is selected and can be deleted.



GPS path display

Filter Min. distance [m] 5.0

Nb. of points found 46 / 78

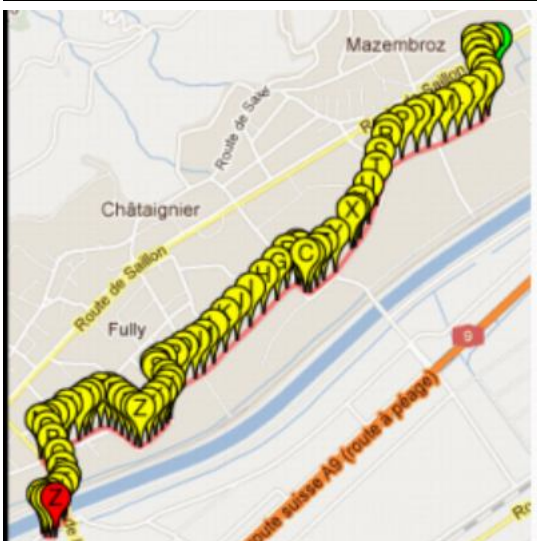
ID	Latitude [°]	Longitude [°]	Altitude [m]	Time [s]
3	46.150439	7.14267	451.60	13.0
4	46.150229	7.14273	455.60	19.0
5	46.149974	7.142851	455.40	23.0
6	46.149701	7.142991	458.80	29.0
7	46.149414	7.143186	459.10	34.0
8	46.149182	7.143292	461.40	39.0
9	46.14891	7.142964	462.90	45.0

DEL FILTER

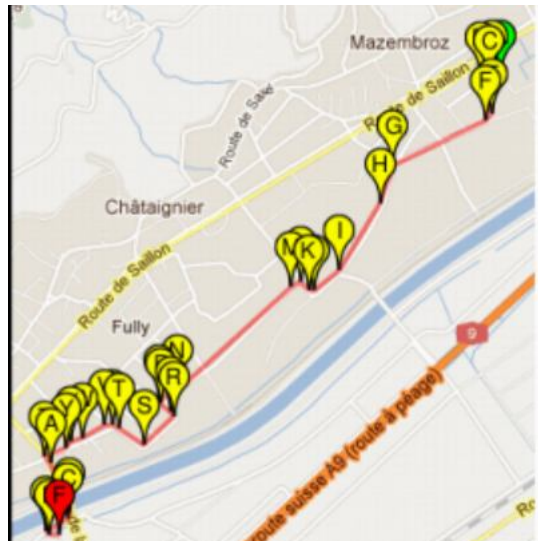
With the file added by default Test2.GPP you can test it yourself.

In this example, Test2.GPP, the number of original points is 78.

The number of points to filter (delete), with a distance  $\Delta$  of 5 m, is 46.





before filtering

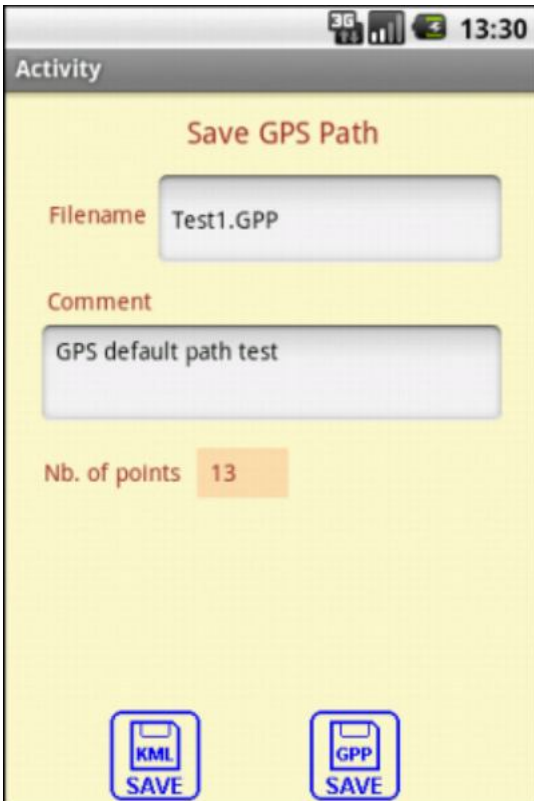


after filtering

## 7.2.7 Save GPS path file / KML file

A GPS path can be save either :

-  in the program specific GPP format (GPS Path)
-  in the Google Earth KML format.



Activity

3G 13:30

Save GPS Path

Filename Test1.GPP

Comment  
GPS default path test

Nb. of points 13

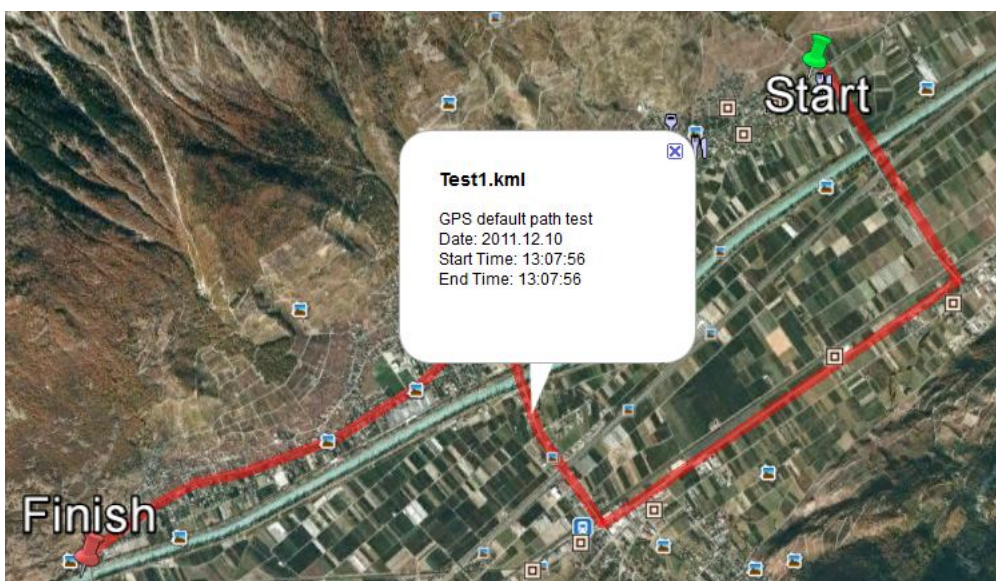
KML SAVE GPP SAVE

Enter the filename.  
The program adds automatically the correct suffix.

Enter a comment.

Reminder of the number of points.

The drawback of the kml format is that you loose the time and speed information for each fix.  
The advantage is, you can display the path in Google Earth.



[KML documentation.](#)

## 7.3 GPS Program Code

Not all the code of the program will be explained in this chapter but only some more special features. Many functions are almost the same as explained in other examples.

**The line numbers in the code snippets in this chapter can be somewhat different from the latest project.**

The program has following modules:

- Main activity module  
main screen, GPS screen and GoogleMaps screen.
- Satellites activity module  
display of the satellites data.
- Setup activity module  
setup screen using a ScrollView with a panel higher than the screen.
- GPSPaths activity module  
displays the GPS path data plus loading, saving and editing.
- GPSSave activity module  
screen for saving a GPS path file with file name and comment entry.
- GPSModule code module  
GPS code used in several activity modules



### 7.3.1 Initialization of the GPS

Android doesn't allow a program to start the GPS automatically for security reasons, only the user can enable it.

If the GPS is disabled, the program must ask the user to enable it, this is done with the following code (in the btnGPS\_Touch routine in the Main module) :

```

If GPS1.GPSEnabled = False Then
  ToastMessageShow("Please enable the GPS device." & CRLF & "And press the BACK button",
True)
  StartActivity(GPS1.LocationSettingsIntent)
End If

```

Here we check if the GPS is enabled.

If no, we show a ToastMessage asking the user to enable it and activate the LocationSettings screen where the user must check the GPS.



Then we can start the GPS with:

```
GPS1.Start(GPSMinTime, GPSMinDistance)
```

Where:

- GPSMinTime = the minimum time before the next fix.
- GPSMinDistance = the minimum distance before the next fix.

To have the quickest sampling of the GPS enter zeros for both parameters : GPS1.Start(0, 0)

### 7.3.2 Button with tooltip

The buttons of the program show a tooltip on top of the screen when they are touched. The views used for this are not Buttons but Panels, because Button don't have the Touch event and Panels have it.

**btnGPS button** (panel), in the Main module:

This button acts as a toggle button, GPS ON or OFF with a color change.

```
Sub btnGPS_Touch(Action As Int, x As Float, y As Float)
    Private bmd As BitmapDrawable

    Select Action
    Case Activity.ACTION_DOWN
        If GPS_On = False Then
            ToolTip("Activates the GPS")
            bmd.Initialize(LoadBitmap(File.DirAssets, "btngps1.png"))
        Else
            ToolTip("Stops the GPS")
            bmd.Initialize(LoadBitmap(File.DirAssets, "btngps0.png"))
        End If
    btnGPS.Background = bmd
```

- First we dim a BitmapDrawable object that will contain the bitmap to display in the button.
- Select ACTION\_DOWN,
  - Check if GPS is OFF (GPS\_On = False) or ON (GPS\_On = True)
  - Load the corresponding bitmap and set it to the buttons background.

```
Case Activity.ACTION_UP
    If x > 0 AND x < btnGPS.Width AND y > 0 AND y < btnGPS.Height Then
        GPS_On = Not(GPS_On)
        If GPS_On = False Then
            bmd.Initialize(LoadBitmap(File.DirAssets, "btngps0.png"))
            GPS1.Stop
            PhoneAwake.ReleaseKeepAlive
            If GPSPath.Size > 0 Then
                cvsMap.DrawRect(rectMapPos, Colors.Transparent, True, 1)
                pnlMap.Invalidate2(rectMapPos)

                If SaveGPSPath = True Then
                    StartActivity(GPSSave)
                End If
            Else
                MsgBox("There are no waypoints", "GPS path saving")
                GPSModule.LoadPath
                MapZoomLevel = MapZoomLevelOld
                MapCenter = MapOldCenter
                MapShow
            End If
```

- Select ACTION\_UP
- Check if the touch coordinates are within the button area. If yes we execute the function.
  - Change the GPS\_On variable
  - Check if GPS\_On = False (GPS disabled) we
    - Load the corresponding bitmap (blue image)
    - Stop the GPS
    - Release the phone keep alive function

- Check if there are GPS path data `GPSPath.Size > 0`
- Check if saving GPS path data is selected
  - Start the GPS path data saving Activity
- If not
  - Display a MessageBox
  - Load the previous GPS path
  - Show the GoogleMap

```

Else
  bmd.Initialize(LoadBitmap(File.DirAssets, "btngps1.png"))
  If GPS1.GPSEnabled = False Then
    ToastMessageShow("Please enable the GPS device." & CRLF & "And press the BACK
button", True)
    StartActivity(GPS1.LocationSettingsIntent)
  End If
  GPSPath.Initialize
  If Map_On Then
    MapZoomLevelOld = MapZoomLevel
    MapOldCenter = MapCenter
    MapZoomLevel = MapDefaultZoomLevel
    MapCenter.Latitude = MapDefaultLat
    MapCenter.Longitude = MapDefaultLng
    MapShow
  End If
  PhoneAwake.KeepAlive(False)
  lblLatitude.Text = "- - -"
  lblLongitude.Text = "- - -"
  lblAltitude.Text = "- - -"
  lblBearing.Text = "- - -"
  lblSpeed.Text = "- - -"

  GPS1.Start(GPSMinTime, GPSMinDistance)
End If

```

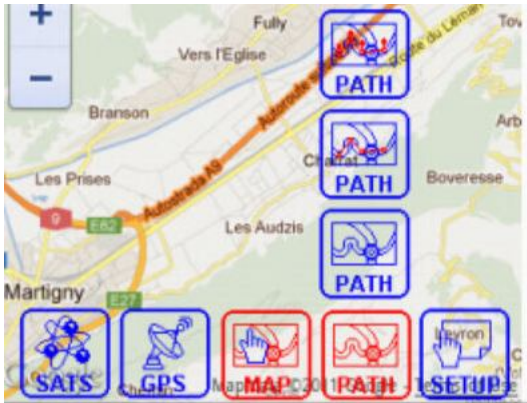
- If `GPS_On = True` (GPS enabled) we
  - Load the corresponding bitmap (red image)
  - Check if the GPS is not enabled on the phone
    - Display a ToastMessage inviting the user to enable the GPS
    - Call the phone setup to let the user enable the GPS
  - Initialize the GPSPath data list
  - Check if the `Map_On = True`, map displayed
    - Set the Zoom level, centre lat and long to the default values
    - Update the map the new parameters
  - Set the PhoneAwake function, the False parameter indicates no bright screen
  - set the different display labels to "- - -", no value
  - Start the GPS, GPSMinTime and GPSMinDistance are defined in the setup screen.

```
    pnlDi spGPSLatLng.Visible = GPS_On
    pnlDi spGPSAltitude.Visible = GPS_On
    pnlDi spGPSSpeed.Visible = GPS_On AND GPSDi spSpeed
    pnlDi spGPSBearing.Visible = GPS_On AND GPSDi spBearing
    pnlDi spGPSWindrose.Visible = GPS_On AND GPSDi spWindrose
    pnlMainBackground.Visible = Not(pnlDi spGPSWindrose.Visible)
    btnGPS.Background = bmd
End If
    ToolTip("")
End Select
End Sub
```

- Then show the different display labels according to setup settings.
  - Set the background image
- Hide the tooltip.

### 7.3.3 Button with tooltip and additional buttons

**btnGPSPath** button (panel), in the Main module:



This button starts the GPSPaths activity, or if the map is displayed shows three more buttons that allow to change the following map setup parameters:

- Display the markers and the polyline of the GPS path on the map
- Display only the polyline, no markers, of the GPS path on the map
- Display only the map

The three upper buttons are on a Panel, `pnlGPSPathToolbox`.

```
Sub btnGPSPath_Touch(Action As Int, x As Float, y As Float)
  Private bmd, bmd1, bmd2, bmd3 As BitmapDrawable
```

```
  Select Action
  Case Activity.ACTION_DOWN
    ToolTip("Shows the GPS path points")
    bmd.Initialize(LoadBitmap(File.DirAssets, "btngpspath1.png"))
    btnGPSPath.Background = bmd
    If Map_On = True Then
      pnlGPSPathToolbox.Visible = True
    End If
```

- First we dim four `BitmapDrawable` objects for the background images.
- Select Activity `ACTION_DOWN`
  - Show the tooltip
  - Load the red image bitmap
  - Set the button background image
  - Check if the map is displayed
    - If yes, we show the `pnlGPSPathToolbox` panel with the three supplementary buttons.

Case Activity.ACTION\_MOVE

```

If x > 0 AND x < btnGPSPath.Width AND y > -3 * btnGPSPath.Height AND y < -2 *
btnGPSPath.Height Then
    bmd3.Initialize(LoadBitmap(File.DirAssets, "btngpspathmarker1.png"))
    bmd2.Initialize(LoadBitmap(File.DirAssets, "btngpspathline0.png"))
    bmd1.Initialize(LoadBitmap(File.DirAssets, "btngpspath0.png"))
    ToolTip("Shows the polyline and the markers")
Else If x > 0 AND x < btnGPSPath.Width AND y > -2 * btnGPSPath.Height AND y < -
btnGPSPath.Height Then
    bmd3.Initialize(LoadBitmap(File.DirAssets, "btngpspathmarker0.png"))
    bmd2.Initialize(LoadBitmap(File.DirAssets, "btngpspathline1.png"))
    bmd1.Initialize(LoadBitmap(File.DirAssets, "btngpspath0.png"))
    ToolTip("Shows the polyline without the markers")
Else If x > 0 AND x < btnGPSPath.Width AND y > -btnGPSPath.Height AND y < 0 Then
    bmd3.Initialize(LoadBitmap(File.DirAssets, "btngpspathmarker0.png"))
    bmd2.Initialize(LoadBitmap(File.DirAssets, "btngpspathline0.png"))
    bmd1.Initialize(LoadBitmap(File.DirAssets, "btngpspath1.png"))
    ToolTip("Doesn't shows the polyline nor the markers")
Else
    bmd3.Initialize(LoadBitmap(File.DirAssets, "btngpspathmarker0.png"))
    bmd2.Initialize(LoadBitmap(File.DirAssets, "btngpspathline0.png"))
    bmd1.Initialize(LoadBitmap(File.DirAssets, "btngpspath0.png"))
    ToolTip("Shows the GPS path points")
End If
btnGPSPath3.Background = bmd3
btnGPSPath2.Background = bmd2
btnGPSPath1.Background = bmd1

```

In this part we check if the move coordinates are in the area of a button and change the button images, red in in the area and blue outsides.

- Select Activity ACTION\_MOVE

- Check if the move coordinates are in the area of the top most button, if yes
  - Load the corresponding images for the four buttons.
- Check if the move coordinates are in the area of the second button from top, if yes
  - Load the corresponding images for the four buttons.
- Check if the move coordinates are in the area of the third button from top, if yes
  - Load the corresponding images for the four buttons.
- Check if the move coordinates are in the area of the lower button, if yes
  - Load the corresponding images for the four buttons.
- Set the images for the three top buttons

```

Case Activity.ACTION_UP
  If x > 0 AND x < btnGPSPath.Width Then
    If y > -3 * btnGPSPath.Height AND y < -2 * btnGPSPath.Height Then
      DispMapMarkers = True
      DispMapPolyline = True
      MapShow
    Else If y > -2 * btnGPSPath.Height AND y < -btnGPSPath.Height Then
      DispMapMarkers = False
      DispMapPolyline = True
      MapShow
    Else If y > -btnGPSPath.Height AND y < 0 Then
      DispMapMarkers = False
      DispMapPolyline = False
      MapShow
    Else If y > 0 AND y < btnGPSPath.Height Then
      StartActivity("GPSPaths")
    End If
  End If
End If
If DispMapMarkers = True Then
  bmd.Initialize(LoadBitmap(File.DirAssets, "btngpspathmarker0.png"))
Else If DispMapPolyline = True Then
  bmd.Initialize(LoadBitmap(File.DirAssets, "btngpspathline0.png"))
Else
  bmd.Initialize(LoadBitmap(File.DirAssets, "btngpspath0.png"))
End If

ToolTip("")
pnlGPSPathToolbox.Visible = False
btnGPSPath.Background = bmd
End Select
End Sub

```

In this part we check in what button area the UP coordinates are in and execute, or not, the corresponding functions.

#### - Select Activity ACTION\_UP

- Check if the move coordinates are in the area of the top most button, if yes
  - Set the setup variables the given values, DispMapMarkers = True DispMapPolyline = True.
  - Update the map.
- Check if the move coordinates are in the area of the second button from top, if yes
  - Set DispMapMarkers = False DispMapPolyline = True.
  - Update the map.
- Check if the move coordinates are in the area of the third button from top, if yes
  - Set DispMapMarkers = False DispMapPolyline = False.
  - Update the map.
- Check if the move coordinates are in the area of the lower button, if yes
  - Start the GPSPaths activity.
- Depending on the setup variables we load the correct bitmap for the btnGPSPath button.
- Hide the tooltip.
- Hide the toolbox of the three upper buttons.
- Set the correct bitmap to btnGPSPath.

### 7.3.4 GPS Calculate distance scales

The two routines below calculate the latitude X and longitude Y coordinates, in km, from coordinate 0, 0 (equator and Greenwich meridian) to the given lat and lng coordinates.

- lat and lng are in degrees, we need to transform them to radians with  $\text{lng} / 180 * \text{cPI}$  and  $\text{lat} / 180 * \text{cPI}$ .
- for Y (lat) we multiply the angle (in radians) by the earth radius (6371 km)
- for X (lng) we multiply the angle (in radians) by the earth radius (6371 km) and multiply by  $\text{CosD}(\text{lat})$ .

The EarthRadius variable is defined in the Process\_Globals routine in the GPSPaths module. The EarthRadius value used, 6371 km, is a mean value. In reality, the [EarthRadius](#) varies with the latitude but for our calculations the assumption of a mean radius is enough accurate.

```
Sub GPSCal cX(lat As Double) As Double
    ' calculates the longitude distance in km from coordinates 0, 0
    Return lat * EarthRadius / 180 * cPI
End Sub
```

It's 'simply' the Earthradius multiplied by the angle lat, but the angle is in degrees so we need to transform it into radians.

```
Sub GPSCal cY(lat As Double, lng As Double) As Double
    ' calculates the latitude distance in km from coordinates 0, 0
    Return lng * EarthRadius / 180 * cPI * CosD(lat)
End Sub
```

The calculation for the lng coordinate is similar to the lat calculation. But, the radius of a [circle of latitude](#) depends on the latitude, so we need to multiply the result by the cosine of lat  $\text{CosD}(\text{lat})$ . We use  $\text{CosD}$  because the angle is in degrees.



### 7.3.5 Drawing GPS position



The current GPS position is drawn on the map when:

the GPS is active,



a map is displayed.



'Show GPS on map' is selected in the setup.

The GoogleMap is drawn on the WebView MapViewer.

The position is drawn on the transparent panel pnlMap which is on top of the Activity and the MapViewer WebView. The code below is in:

```
Sub GPS1_LocationChanged (Location1 As Location)
.
.
  If ShowGPSOnMap = True Then
    Private xc, yc As Float
    xc = (Location1.Longitude - MapCenter.Longitude) / MapScaleLng + MapViewer.Width / 2
    yc = (MapCenter.Latitude - Location1.Latitude) / MapScaleLat + MapViewer.Height / 2
    If xc < 10%x OR xc > 90%x OR yc < 20%y OR yc > 80%y Then
      MapCenter.Latitude = Location1.Latitude
      MapCenter.Longitude = Location1.Longitude
      MapShow
    End If
    DrawGPSPosition(xc, yc)
  End If
End Sub
```

Where:

- Location1.Longitude, Location1.Latitude are the current location coordinates in degrees.
- MapCenter.Longitude, MapCenter.Latitude are the current map center coordinates in degrees
- MapScaleLng, MapScaleLat are the scales, degrees/pixel, of the current map.
- xc, yc are the coordinates of the current location in pixels.

Location1.Longitude and Location1.Latitude, the coordinates of the current position, are given by the GPS.

In the equation.

$$xc = (Location1.Longitude - MapCenter.Longitude) / MapScaleLng + MapViewer.Width / 2$$

$$(Location1.Longitude - MapCenter.Longitude)$$

Is the x distance, in degrees, from the map center to the current location.

$$(Location1.Longitude - MapCenter.Longitude) / MapScaleLng$$

Is the x distance, in pixels, from the the map center to the current location.

$$(Location1.Longitude - MapCenter.Longitude) / MapScaleLng + MapViewer.Width / 2$$

Is the x distance, in pixels, from the left border to the current location.

The equation for yc is similar.

```

If xc < 10%x OR xc > 90%x OR yc < 20%y OR yc > 80%y Then
  MapCenter.Latitude = Location1.Latitude
  MapCenter.Longitude = Location1.Longitude
  MapShow
End If

```

Here we check if the current location reaches one of the maps borders, and if true, we set the map centre coordinates to the current location coordinates and redraw the map.

DrawGPSPosition(xc, yc) Draws the location on the map.

Complementary calculation routines:

CalcMapScales: calculates the map pixel scales

```

Sub CalcMapScales
  ' Calculates the map scales
  MapScaleLng = 360 / Power(2, MapZoomLevel) / TileSize
  MapScaleLat = MapScaleLng * CosD(MapCenter.Latitude)
End Sub

```

Where:

- MapScaleLng = Longitude scale in degrees/pixel
- MapZoomLevel = Current map zoom level
- TileSize = Map tile size in pixels
- MapScaleLat = Latitude scale in degrees/pixel
- MapCenter.Latitude = Center lat coordinate of the current map in degrees.

360 is the earth circumference in degrees.

Power(2, MapZoomLevel) is the number of tiles for the given zoom level.

The default map tile size is 256 pixels defined in the Main module.

```
Private TileSize As Int : TileSize = 256
```

But this tile size must be changed according to the device's density, because on different devices with almost the same physical dimensions, but different densities, the size of the map is the same. That means that the tile size is proportional to the device density.

```

Private Iv As LayoutValues
Iv = GetDeviceLayoutValues
TileSize = TileSize * Iv.Scale

```

Drawing of the GPS position on the map :



```
Sub DrawGPSPosition(xc As Float, yc As Float)
```

```
Private x1, y1, x2, y2 As Float
```

```
Private dd1, dd2, r As Float
```

```
cv$Map.DrawRect(rectMapPos, Colors.Transparent, True, 1)
```

```
pnlMap.Invalidate2(rectMapPos)
```

```
dd1 = 20dip
```

```
dd2 = 20dip
```

```
r = 10dip
```

```
x1 = xc - dd1
```

```
y1 = yc - dd1
```

```
x2 = xc + dd2
```

```
y2 = yc + dd2
```

```
rectMapPos.Initialize(x1, y1, x2 + 1, y2 + 1)
```

```
cv$Map.DrawLine(x1, yc, x2, yc, Colors.Red, 1dip)
```

```
cv$Map.DrawLine(xc, y1, xc, y2, Colors.Red, 1dip)
```

```
cv$Map.DrawCircle(xc, yc, r, Colors.Red, False, 3dip)
```

```
pnlMap.Invalidate2(rectMapPos)
```

```
End Sub
```

We

- define local variables
- draw a transparent rectangle to erase the previous position
- update the drawing
- set the variables for the drawing
- define the new surrounding rectangle
- draw a horizontal line
- draw a vertical line
- draw a circle
- update the drawing

If there are code sections you would like to be developed here, please post the questions and suggestions in the [GPSExample](#) thread on the users forum.

## 8 Widgets, home screen widgets

This chapter is a copy of Erels' two tutorials in the users forum.

[Android home screen widgets tutorial - part I](#)

[Android home screen widgets tutorial - part II](#)

### 8.1 Widgets Part I

Basic4android v1.6 adds support for home screen widgets. This tutorial will explain how to implement your own home screen widgets (also named App Widgets).

It is important to understand that the widgets are created and managed in another process, different than the process that your application is running in. The home screen application is hosting your widgets.

This means that it is not possible to directly access the widgets views. Instead we are using a special object named RemoteViews which gives us indirect access to the widget views.

Widgets do not support all views types. The following views are supported:

- Button (default drawable)
- Label (ColorDrawable or GradientDrawable)
- Panel (ColorDrawable or GradientDrawable)
- ImageView
- ProgressBar (both modes)

All views support the Click event and no other event.

The widget layout and configuration must be defined with XML files. During compilation Basic4android reads the layout file created with the designer and generates the required XML files.

Each widget is tied to a Service module. The widget is created and updated through this module.

#### Creating a widget - step by step guide

- Add a Service module. Note that the service module handling the widget is a standard service.
- Design the widget layout with the designer. First add a Panel and then add the other views to this Panel.

The widget layout will be made from this panel.

- Add code similar to the following code the service module:

```

Sub Process_Global s
  Public rv As RemoteViews
End Sub

Sub Service_Create
  rv = ConfigureHomeWidget("LayoutFile", "rv", 0, "Widget Name")
End Sub

Sub Service_Start (StartingIntent As Intent)
  If rv.HandleWidgetEvents(StartingIntent) Then Return
End Sub

Sub rv_RequestUpdate
  rv.UpdateWidget
End Sub

Sub rv_Disabled
  StopService("")
End Sub

Sub Service_Destroy

End Sub

```

- Compile and run your application. Go to the home screen, long press on the screen and you will see your widget listed on the widgets list.

**ConfigureHomeWidget** is a special keyword. At runtime it creates the RemoteViews object from the layout and sets the events. At compile time the compiler generates the required files based on the arguments of this keyword.

The four parameters are: layout file, event name, update interval and the widget name.

Event name sets the subs that will handle the RequestUpdate and Disabled events.

The widget can be configured to update itself automatically. The interval, measured in minutes, defines how often will the widget request to update itself. Set to 0 to disable automatic updates. Updating the widget too often will have a bad impact on the battery. The minimum value is 30 minutes.

Widget name - the name that will appear in the widgets list.

As these arguments are read by the compiler, only strings or numbers are accepted.

### Events:

```

Sub Service_Start (StartingIntent As Intent)
  If rv.HandleWidgetEvents(StartingIntent) Then Return
End Sub

```

The above code checks the Intent message that caused this service to start and is responsible for raising the events related to the widget. It returns true if an event was raised.

The widget raises two events. RequestUpdate is raised when the widget needs to update itself. It will fire after adding the widget to the screen, after the device has booted, based on the scheduled updating interval (if set) or after the application was updated.

The Disabled event is raised when the last instance of our widget is removed from the screen.

As mentioned above all views support the Click event. All that needs to be done in order to handle the click event of a button named Button1 is to add a sub named Button1\_Click (the sub name should actually match the EventName property which is by default the same as the name). For example if you want to show the main Activity when the user presses on Button1 you can use this code:

```
Sub Button1_Click
    StartActivity(Main)
End Sub
```

### Modifying the widget:

It is not possible to directly access the widget views. Instead we need to use one of the RemoteView.Set methods.

If we want to change the text of a label named Label1 then we need to write the following code:

```
rv.SetText("Label1", "This is the new text.")
'do more changes if needed
rv.UpdateWidget
```

After writing all the changes we call rv.UpdateWidget to send the updates to the widget.

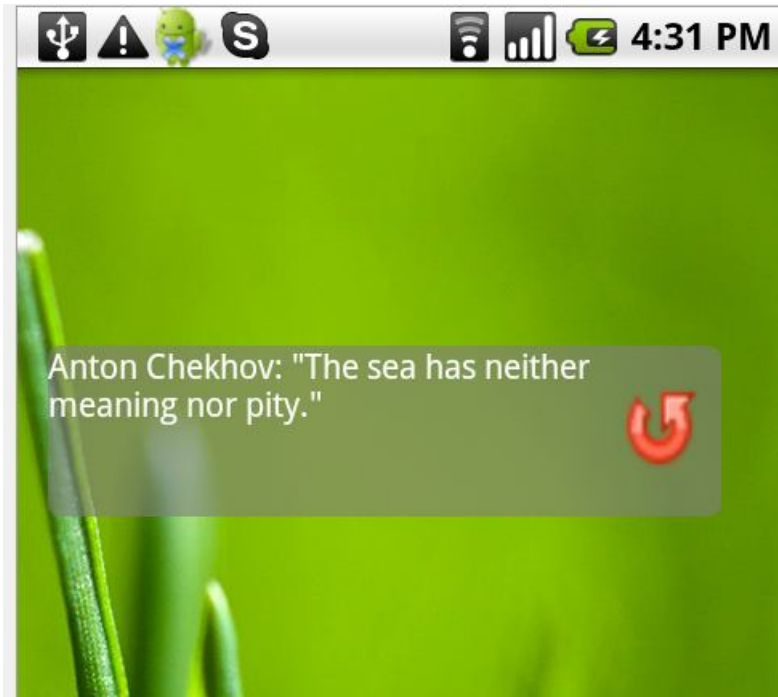
A simple example is available in the forum : [HomeWidgets](#).

The example adds a simple widget. The widget doesn't do anything useful.



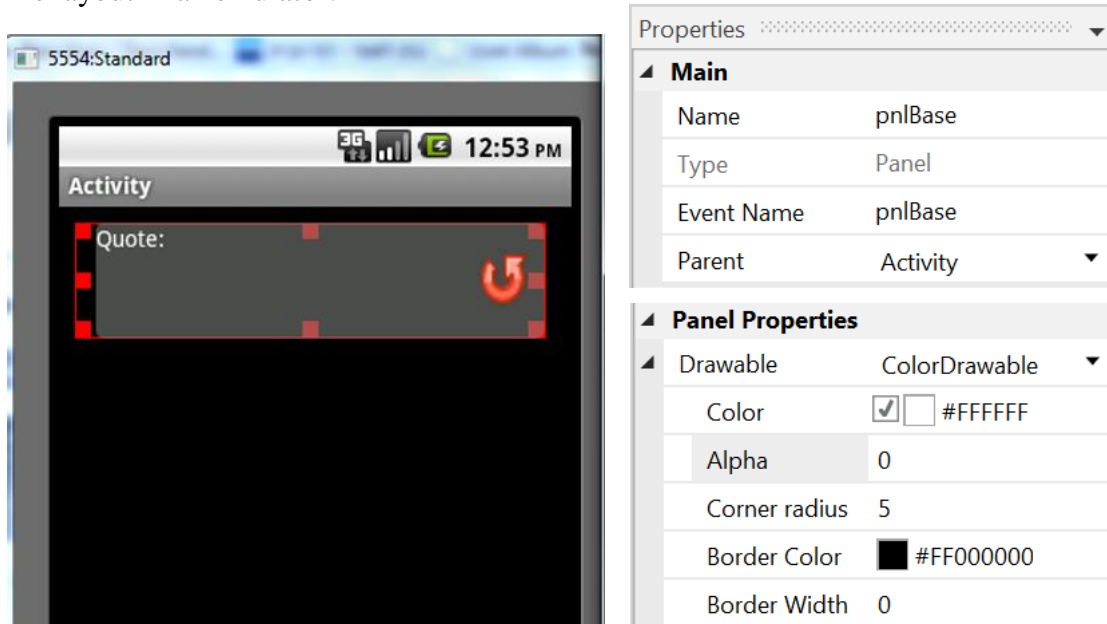
## 8.2 Widgets Part II

In this part we will build a "quote of the day" widget.



**We will start with the layout.** The widget is made of a Label for the text and an ImageView for the arrow button.

The layout in an emulator:



You can see in the above picture that we use two panels. The base panel named `pnlBase` is a transparent panel (`Alpha=0`). The base panel contains another panel which is the grey panel. The purpose of the transparent panel is to add some padding to the other views. The widget size is determined by the base panel. Without the transparent panel there will be no margin between the visible widget and the screen left edge.

Horizontal Ancho	LEFT
Vertical Anchor	TOP
Left	10
Top	10
Width	294
Height	72

We are setting the base panel size to 294x72.

This is the recommended size for a 4x1 cells widget.

Tip: in some cases when you change the layout and there is already an existing widget in the device, the widget doesn't get updated. Remove the widget and add it again to see the change.

### Now for the program logic.

Once a day the program fetches 20 quotes from 5 feeds available from [Famous Quotes at BrainyQuote](#)

Then the first quote is displayed. Each time the user presses on the arrow button the next quote is displayed.

While getting the quotes the first quote of each feed is added to the beginning of the quotes list.

Only the first quote on each feed is new, and we want to start with the new quotes.

Downloading the feeds is done with HttpUtils and parsing them is done with XmlSax library. See the code for more information.

The widget is configured to be updated automatically every 24 hours. This is done in this line:

```
'configure the widget and set it to update every 24 hours (1440 minutes).
rv = ConfigureWidget("WidgetLayout", "rv", 1440, "Quote of the day")
```

After 24 hours or when the widget is first added or after a boot the RequestUpdate event is raised.

#### Sub rv\_RequestUpdate

```
quotes.Clear
currentQuote = -1
HttpUtils.DownloadList("Quotes", Array As
String("http://feeds.feedburner.com/brainyquote/QUOTEBR", _
"http://feeds.feedburner.com/brainyquote/QUOTEAR", _
"http://feeds.feedburner.com/brainyquote/QUOTEFU", _
"http://feeds.feedburner.com/brainyquote/QUOTELO", _
"http://feeds.feedburner.com/brainyquote/QUOTENA"))
End Sub
```

First we clear the current quotes and then we fetch the new ones. Note that if the device was sleeping at this time then the calls are likely to fail as most devices turn off the wifi while sleeping.

In this case new quotes will arrive when the user presses on the arrow button.

In cases like this you should not count on the automatic update to succeed and make sure that there is an alternative way to update the widget.

**Persisting the data.** The process running our widget code will not stay alive forever. It will be killed by the OS at some point.

Therefore we cannot rely on global variables to store our data.

All of the "state" variables must be written to a file.

RandomAccessFile.WriteObject and ReadObject are very useful for such tasks.

Each time that the widget sends a request to our application, Service Start is called.

Not much is done in this sub:

```
Sub Service_Start (StartingIntent As Intent)
If rv.HandleWidgetEvents(StartingIntent) Then Return
End Sub
```



However if our process is not alive yet then Service\_Create will be called before. Service\_Create is an important point, as it allows us to read the previously saved state to memory:

#### Sub Service\_Create

```
'configure the widget and set it to update every 24 hours (1440 minutes).
rv = ConfigureHomeWidget("WidgetLayout", "rv", 1440, "Quote of the day")
HttpUtils.CallBackActivity = "WidgetService"
HttpUtils.CallBackUrlDoneSub = "Url Done"
HttpUtils.CallBackJobDoneSub = "JobDone"
parser.Initialize

'Load previous data if such is available.
'This is relevant in case our process was killed and now the user pressed on the
widget.
If File.Exists(File.DirectoryInternalCache, QUOTES_FILE) Then
    raf.Initialize(File.DirectoryInternalCache, QUOTES_FILE, True)
    quotes = raf.ReadObject(0)
    raf.Close
Else
    quotes.Initialize
End If
If File.Exists(File.DirectoryInternalCache, CURRENTQUOTE_FILE) Then
    currentQuote = File.ReadString(File.DirectoryInternalCache, CURRENTQUOTE_FILE)
End If
End Sub
```

The source code is available in the forum : [Quotes](#).

## 9 OkHttpUtils2

OkHttpUtils2 is a standard library that helps with communicating with web services (Http servers). It replaces the previous HttpUtils2 library.

The previous examples in the HttpUtilsExamples folder have been modified and saved in the OkHttpUtilsExamples folder.

The only thing I had to do to adapt the programs to the new library was to replace the 'old' libraries HTTP and HttpUtils2 by the new ones OkHTTP and OkHttpUtils2 in the IDE Libraries Manager Tab.

### 9.1 OkHttpUtils2 Objects

The OkHttpUtils2 library contains three objects:

- **HttpJob.**  
How to define a job.  
Example:  

```
Dim Job1 As HttpJob
Job1.Initialize("Job1", Me)
Job1.Download(b4a)
```
- **HttpUtils2Service**  
Is used internally by HttpJob.
- **MultipartFileData**

### 9.2 HttpJob Functions

The HttpJob object has following methods:

- **Complete**(id As Int)  
Called by the service when job completes.
- **Download**(Link As String)  
Submits a HTTP GET request.  
Consider using Download2 if the parameters should be escaped.
- **Download2**(JobName As String, Parameters As String)  
Submits a HTTP GET request.  
Encodes illegal parameter characters.  
Example:  

```
job.Download2("http://www.example.com", Array As String("key1", _  
"value1", "key2", "value2"))
```
- **GetBitmap** As Bitmap  
Returns the response as a bitmap
- **GetBitmapSample** As Bitmap  
Returns the response as a bitmap loaded with LoadBitmapSample.

- **GetInputStream** *As InputStream*  
Returns an `InputStream`
- **GetRequest** *As InputStream*  
Returns an `OkHttpRequest`
- **GetString** *As String*  
Returns the response as a string encoded with UTF8.
- **GetString2** (Encoding *As String*) *As String*  
Returns the response as a string with the specified encoding.
- **Initialize** (Name *As String*, TargetModule *As Object*)  
Initializes the Job.  
Name - The job's name. Note that the name doesn't need to be unique.  
TargetModule - The activity or service that will handle the JobDone event.
- **IsInitialized** *As Boolean*  
Tests whether the object has been initialized.
- **JobName**  
Field containing the job name.  
Example : `If job1.JobName = "Job1") Then`
- **Password**  
Field containing a string with the job password.  
Example : `If job1.Password = "PassWord") Then`
- **PostBytes**(Link *As String*, Data *As Bytes()*)  
Sends a POST request with the given Byte array as the post data.
- **PostFile**(Link *As String*, Dir *As String*, FileName *As String*)  
Sends a POST request with the given file as the post data.  
This method doesn't work with assets files.
- **PostMultipart**(Link *As String*, NameValues *As Map*, Files *As List*)  
Sends a multipart POST request..  
NameValues - A map with the keys and values. Pass Null if not needed.  
Files - List of `MultipartFileData` items. Pass Null if not needed.
- **PostString**(Link *As String*, Text *As String*)  
Sends a POST request with the given string as the post data.
- **PutBytes**(Link *As String*, Data *As Byte()*)  
Sends a POST request with the Byte array as the post data.
- **PutString**(Link *As String*, Text *As String*)  
Sends a POST request with the given string as the post data.
- **Release**  
Should be called to free resources held by this job.

- **Success**  
Field containing a Boolean.  
Example : If job1.Password = "PassWord") Then
- **Tag**  
Field containing the job Tag.  
Example : If job1.Tag = "something") Then
- **UserName**  
Field containing a string with the job username.  
Example : If job1.UserName = "UserName") Then

The calling module must contain a JobDone routine where you handle the results of the different jobs.

Example :

```
Sub JobDone(Job As HttpJob)
    If Job.Success = True Then
        Dim s As String
        s = Job.GetString
        Log(s)
    End If
End Sub
```

Look at Example2 for more than one job.

### 9.3 OkHttpUtils2 Example1

A simple example of downloading a page and returning the page as string:

Source code : OkHttpUtilsExamples\OkHttpUtilsExample1.b4a

```
Sub Global s
  Dim b4a As String
  b4a = "http://www.b4x.com"
End Sub

Sub Activity_Create(FirstTime As Boolean)
  Dim Job1 As HttpJob
  Job1.Initialize("Job1", Me)
  Job1.Download(b4a)
End Sub

Sub JobDone(Job As HttpJob)
  If Job.Success = True Then
    Dim s As String
    s = Job.GetString
    Log(s)
  End If
End Sub
```

First we initialize the job.

"Job1" = job name

Me = current calling module

Then we call Job1.Download. This call submits a **job** request to HttpUtils.

HttpUtils raises an event `Sub JobDone(Job As HttpJob)` when the job is finished.

We have four ways to access a downloaded resource:

- `HttpJob.GetString` - Returns the resource as string
- `HttpJob.GetString2(Encoding As String)` - Returns the resource as string with a specific encoding.
- `HttpJob.GetBitmap` - Returns the resource as bitmap
- `HttpJob.GetInputStream` - Returns an `InputStream` which allows you to manually read the downloaded resource.

These four methods should only be called in the `JobDone` routine.

Inside the `JobDone` event sub you should check `Job.Success = True` to ensure that the job request was successful.

## 9.4 OkHttpUtils2 Example2

Source code: OkHttpUtilsExamples\OkHttpUtilsExample2.b4a

In this example we first download an image and set it as the activity background. Then we download another two Urls and print them as string.

```

Sub Activity_Create(FirstTime As Boolean)
    Dim job1, job2, job3 As HttpJob
    job1.Initialize("Job1", Me)

    ' Send a GET request
    job1.Download2("http://www.b4x.com/print.php", _
        Array As String("first key", "first value :)", "second key", "value 2"))

    ' Send a POST request
    job2.Initialize("Job2", Me)
    job2.PostString("http://www.b4x.com/print.php", "first key=first value&key2=value2")

    ' Send a GET request
    job3.Initialize("Job3", Me)
    job3.Download("http://www.b4x.com/forum/images/categories/android.png")
End Sub

Sub JobDone (Job As HttpJob)
    Log("JobName = " & Job.JobName & ", Success = " & Job.Success)
    If Job.Success = True Then
        Select Job.JobName
            Case "Job1", "Job2"
                'print the result to the logs
                Log(Job.GetString)
            Case "Job3"
                'show the downloaded image
                Activity.SetBackgroundImage(Job.GetBitmap)
        End Select
    Else
        Log("Error: " & Job.ErrorMessage)
        ToastMessageShow("Error: " & Job.ErrorMessage, True)
    End If
    Job.Release
End Sub

```

In Activity\_Create we :

- Dim the three jobs.
- send a Get request in job1
- send a Post request in job2
- send a Get request in job3

In JobDone we :

- Test if the current job was successful
- If yes, check the current job name and execute its code.
- If no, display a ToastMessage

## 9.5 The Flickr Viewer example

The FlickrViewer example uses the OkHttpUtils2 library.

Source code : `okHttpUtilsExamples\FlickrViewer.b4a`

In this example we first go to the "main" page of this site  
<http://www.flickr.com/explore/interesting/7days/>.

In this page we find 9 links to 9 images. We submit a second job with all these links.

We show each image as soon as it is ready in the by calling `ImagesJobDone` from the `JobDone` event.

Clicking on an image shows in a second activity as the background image.



## 10 Network / AsyncStreams

The Network library allows you to communicate over TCP/IP with other computers or devices.

The Network library contains two objects. Socket and ServerSocket.

The Socket object is the communication endpoint. Reading and writing are done with Socket.InputStream and Socket.OutputStream.

ServerSocket is an object that listens for incoming connections. Once a connection is established an event is raised and a socket object is passed to the event sub. This socket will be used to handle the new client.

### Client application

Steps required:

- Create and initialize a Socket object.
- Call Socket.Connect with the server address.
- Connection is done in the background. The Connected event is raised when the connection is ready or if it failed.
- Communicate with the other machine using Socket.InputStream to read data and Socket.OutputStream to write data.

### Server application

Steps required:

- Create and initialize a ServerSocket object.
- Call ServerSocket.Listen to listen for incoming connections. This happens in the background.
- Once a connection is established the NewConnection event is raised and a Socket object is passed.
- Call ServerSocket.Listen if you want to accept more connections.
- Using the Socket object received, communicate with the client.



We will see two examples.

The first example connects to a time server and displays the current date and time as received from the server.

```
Sub Process_Global s
    Dim Socket1 As Socket
End Sub

Sub Global s

End Sub

Sub Activity_Create(FirstTime As Boolean)
    Socket1.Initialize("Socket1")
    Socket1.Connect("nist1-ny.ustiming.org" , 13, 20000)
End Sub

Sub Socket1_Connected (Successful As Boolean)
    If Successful = False Then
        MsgBox(LastException.Message, "Error connecting")
        Return
    End If
    Dim tr As TextReader
    tr.Initialize(Socket1.InputStream)
    Dim sb As StringBuilder
    sb.Initialize
    sb.Append(tr.ReadLine) 'read at least one line
    Do While tr.Ready
        sb.Append(CRLF).Append(tr.ReadLine)
    Loop
    MsgBox("Time received: " & CRLF & sb.ToString, "")
    Socket1.Close
End Sub
```

We are creating a new socket and trying to connect to the server which is listening on port 13.

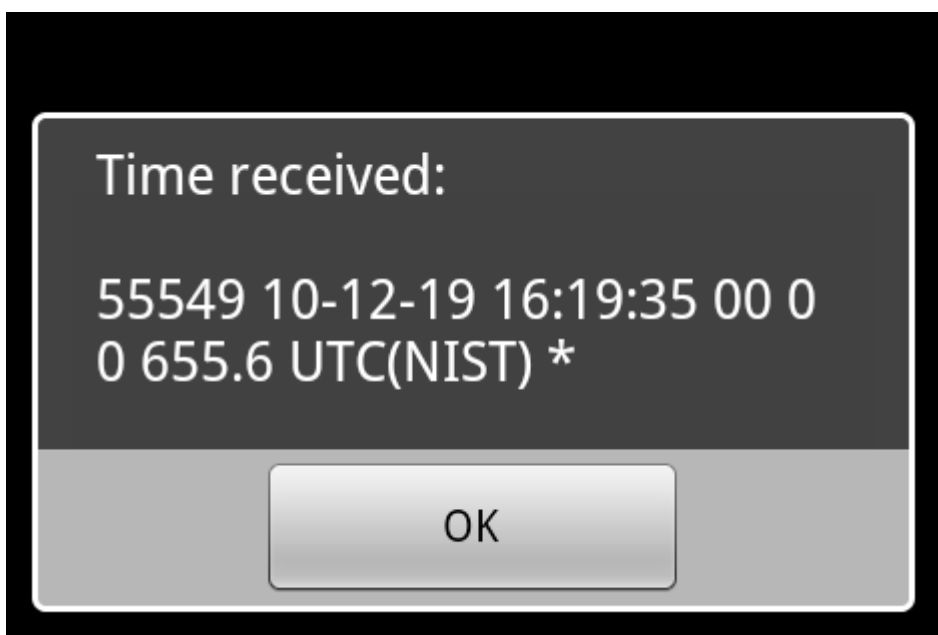
The next step is to wait for the Connected event.

If the connection is successful we create a TextReader object and initialize it with

Socket1.InputStream. In this case we want to read characters and not bytes so a TextReader is used.

Calling tr.ReadLine may block. However we want to read at least a single line so it is fine.

Then we read all the other available lines (tr.Ready means that there is data in the buffer).



In the second application we will create a file transfer application, that will copy files from the desktop to the device.

The device will use a ServerSocket to listen to incoming connections.

Once a connection has been made, we will enable a timer. This timer checks every 200ms whether there is any data waiting to be read.

The file is sent in a specific protocol. First the file name is sent and then the actual file.

We are using a RandomAccessFile object to convert the bytes read to numeric values.

RandomAccessFile can work with files or arrays of bytes, we are using the later in this case.

RandomAccessFile can be set to use little endian byte order. This is important here as the desktop uses this byte order as well.

The desktop example application was written with Basic4ppc.

Once connected the user selects a file and the file is sent to the device which saves it under /sdcard/android.

Both applications are attached.

Some notes about the code:

- The server is set to listen on port 2222.

The server displays its IP when it starts. The desktop client should use this IP address when connecting to a real device (this IP will not work with the emulator).

However if you work with the emulator or if your device is connected to the computer in debug mode you can use 'adb' to forward a desktop localhost port to the device.

This is done by issuing "adb forward tcp:5007 tcp:2222"

Now in the client code we should connect to the localhost ip with port 5007.

```
Client.Connect("127.0.0.1", 5007)
```

Again if you are testing this application in the emulator you must first run this adb command. Adb is part of the Android SDK.

- Listening to connections :

```
Sub Activity_Resume
```

```
    ServerSocket1.Listen
```

```
End Sub
```

```
Sub Activity_Pause (UserClosed As Boolean)
```

```
    If UserClosed Then
```

```
        Timer1.Enabled = False
```

```
        Socket1.Close
```

```
        ServerSocket1.Close 'stop listening
```

```
    End If
```

```
End Sub
```

```
Sub ServerSocket1_NewConnection (Successful As Boolean, NewSocket As Socket)
```

```
    Dim InputStream1, OutputStream1 As AsyncStreams
```

```
    If Successful Then
```

```
        Socket1 = NewSocket
```

```
        Timer1.Enabled = True
```

```
        InputStream1 = Socket1.InputStream
```

```
        OutputStream1 = Socket1.OutputStream
```

```
        ToastMessageShow("Connected", True)
```

```
    Else
```

```
        MsgBox(LastException.Message, "Error connecting")
```

```
    End If
```

```
    ServerSocket1.Listen 'Continue listening to new incoming connections
```

```
End Sub
```

In Sub `Activity_Resume` (which also called right after `Activity_Create`) we call `ServerSocket.Listen` and start listening to connections. Note that you can call this method multiple times safely.

In Sub `Activity_Pause` we close the active connection (if there is such a connection) and also stop listening. This only happens if the user pressed on the back key (`UserClosed = True`).

The `ServerSocket` will later be initialized in `Activity_Create`.

The server side application can handle new connections. It will just replace the previous connection with the new one.

The desktop client example application doesn't handle broken connections. You will need to restart it to reconnect.

## 11 Advanced drawings

Three main chapters for advanced drawing:

- View Drawables
- Layers with Panels / ImageViews / Images
- Diagrams / Charts

### 11.1 View Drawables

The views have default backgrounds when they are defined either in the Designer or by code. There do exist three other background objects.

- ColorDrawables
- GradientDrawables
- BitmapDrawables
- StateListDrawable

These can be defined in the Designer, but when we want to modify them in the code it needs some code.

The source code is in the Background folder.

#### 11.1.1 ColorDrawable

The ColorDrawable object has a solid single color, the corners can rounded or not. The code below sets a ColorDrawable background to a panel.

```
Dim pnlColor As Panel

pnlColor.Initialize("")
Activity.AddView(pnlColor, 10%x, 40di p, 80%x, 80di p)
Dim cdwColor As ColorDrawable
cdwColor.Initialize(Colors.Red, 5di p)
pnlColor.Background = cdwColor
```



```
cdwColor.Initialize(Colors.Red, 5di p)
```

The Initialize method of the ColorDrawable object needs two properties :

- Color `Colors.Red`
- CornerRadius `5di p`

### 11.1.2 GradientDrawable

The GradientDrawable object has two colors with a gradient change from the first to the second color.

The code below sets a GradientDrawable background to a panel.

```
Dim pnlGradient As Panel
```

```
pnlGradient.Initialize("")
Activity.AddView(pnlGradient, 10%x, 140dip, 80%x, 80dip)
Dim gdwGradient As GradientDrawable
Dim Colors(2) As Int
Colors(0) = Colors.Blue
Colors(1) = Colors.White
gdwGradient.Initialize("TOP_BOTTOM", Colors)
gdwGradient.CornerRadius = 10dip
pnlGradient.Background = gdwGradient
```



```
gdwGradient.Initialize("TOP_BOTTOM", Colors)
```

The GradientDrawable Initialize method needs two parameters :

- a string with the orientation "TOP\_BOTTOM"
- a color array with two colors Colors

The possible orientations are:

- TOP\_BOTTOM
- TR\_BL (Top - Right to Bottom - Left)
- RIGHT\_LEFT
- BR\_TL (Bottom - Right to Top - Left)
- BOTTOM\_TOP
- BL\_TR (Bottom - Left to Top - Left)
- LEFT\_RIGHT
- TL\_BR (Top - Left to Bottom - Right)

The CornerRadius is another separate property.

```
gdwGradient.CornerRadius = 10dip
```

### 11.1.3 BitmapDrawable

The BitmapDrawable object has two properties a Bitmap and a Gravity property.

The BitmapDrawable object has no rounded corner property, if you want rounded corners these must be part of the bitmap.

The code below sets a BitmapDrawable background to a panel.

```
Dim pnlBitmap As Panel
```

```
pnlBitmap.Initialize("")
```

```
Activity.AddView(pnlBitmap, 10%x, 250dip, 80%x, 80dip)
```

```
Dim bdwBitmap As BitmapDrawable
```

```
bdwBitmap.Initialize(LoadBitmap(File.DirAssets, "background.png"))
```

```
bdwBitmap.Gravity = Gravity.FILL
```

```
pnlBitmap.Background = bdwBitmap
```



Aletsch glacier, picture taken from the Jungfrauoch.

```
bdwBitmap.Initialize(LoadBitmap(File.DirAssets, "background.png"))
```

Sets the bitmap property, in this case a file loaded from the File.DirAssets folder.

But it could be any bitmap.

It is also possible to draw onto the panels background with a Canvas which has this Panel as the target. The Canvas.Bitmap property points to the `bdwBitmap.Bitmap` property.

```
bdwBitmap.Gravity = Gravity.FILL
```

Sets the Gravity property.

The Gravity property values can be:

- BOTTOM
- CENTER
- CENTER\_HORIZONTAL
- CENTER\_VERTICAL
- FILL
- LEFT
- NO\_GRAVITY
- BOTTOM
- TOP

The Gravity property can be a combination of above values.

Examples : `bdwBitmap.Gravity = Gravity.TOP + Gravity.LEFT`

`bdwBitmap.Gravity = Gravity.BOTTOM + Gravity.CENTER_HORIZONTAL`

In the Designer there are only three values available: Fill, Center and Top-Left.

### 11.1.4 StateListDrawable

The StateListDrawable is a drawable that holds other drawables and chooses the current one based on the view's state.

The Background property of Buttons is a StateListDrawable, it can be defined either in the Designer (see chapter 3 in the Beginner's Guide) or in the code.

In the Designer there are two options:

- DefaultDrawable      default colors      set by default
- StateListDrawable      custom colors

The button StateListDrawable has three states.

- Enabled Drawable
- Disabled Drawable
- Pressed Drawable

Each state has its own Drawable, that could be one of the three ColorDrawable, GradientDrawable or BitmapDrawable.

Example code for a Button with a ColorDrawable :

The source code is the in the ButtonStateDrawables folder.

```
btnColor.Initialize("btnColor")
Activity.AddView(btnColor, 20dip, 100dip, 100dip, 60dip)
btnColor.Text = "Color"

' Define a color for Enabled state
Dim cdwGreenColorEnabled As ColorDrawable
cdwGreenColorEnabled.Initialize(Colors.Green, 10)

' Define a color for Pessed state
Dim cdwGreenColorPressed As ColorDrawable
cdwGreenColorPressed.Initialize(Colors.RGB(255, 182, 18), 10)

' Define a StateListDrawable
Dim stdGreenColor As StateListDrawable
stdGreenColor.Initialize
Dim states(2) As Int
states(0) = stdGreenColor.state_enabled
states(1) = -stdGreenColor.state_pressed
stdGreenColor.AddState2(states, cdwGreenColorEnabled)
Dim states(1) As Int
states(0) = stdGreenColor.state_pressed
stdGreenColor.AddState2(states, cdwGreenColorPressed)

' Set stdGreenColor to button background
btnColor.Background = stdGreenColor
```

Example code for a Button with a GradientDrawable :

```
btnGradient.Initialize("btnGradient")
Activity.AddView(btnGradient, 20dip, 180dip, 100dip, 60dip)
btnGradient.Text = "Gradient"

' Define two gradient colors for Enabled state
Dim colsEnabled(2) As Int
colsEnabled(0) = Colors.RGB(255, 196, 196)
colsEnabled(1) = Colors.RGB(255, 25, 25)
' Define a GradientDrawable for Enabled state
Dim gdwEnabled As GradientDrawable
gdwEnabled.Initialize("TOP_BOTTOM", colsEnabled)
gdwEnabled.CornerRadius = 5
' Define two gradient colors for Pressed state
Dim colsPressed(2) As Int
colsPressed(0) = Colors.RGB(25, 255, 25)
colsPressed(1) = Colors.RGB(255, 255, 255)
' Define a GradientDrawable for Pressed state
Dim gdwPressed As GradientDrawable
gdwPressed.Initialize("TOP_BOTTOM", colsPressed)
gdwPressed.CornerRadius = 5
' Define a StateListDrawable
Dim stdGradient As StateListDrawable
stdGradient.Initialize
Dim states(2) As Int
states(0) = stdGradient.state_enabled
states(1) = -stdGradient.state_pressed
stdGradient.addState2(states, gdwEnabled)
Dim states(1) As Int
states(0) = stdGradient.state_pressed
stdGradient.addState2(states, gdwPressed)
' Set stdRedGradient to button background
btnGradient.Background = stdGradient
```



Example code for a Button with a BitmapDrawable :

```
btnBitmap.Initialize("btnBitmap")
Activity.AddView(btnBitmap, 40dp, 260dp, 60dp, 60dp)

' Define a bitmap for Enabled state
Dim bdwEnabled As BitmapDrawable
bdwEnabled.Initialize(LoadBitmap(File.DirAssets, "btnArrowDown0.png"))
' Define a bitmap for Pressed state
Dim bdwPressed As BitmapDrawable
bdwPressed.Initialize(LoadBitmap(File.DirAssets, "btnArrowDown1.png"))
' Define a StateListDrawable
Dim stdBitmap As StateListDrawable
stdBitmap.Initialize
Dim states(2) As Int
states(0) = stdBitmap.state_enabled
states(1) = -stdBitmap.state_pressed
stdBitmap.addState2(states, bdwEnabled)
Dim states(1) As Int
states(0) = stdBitmap.state_enabled
stdBitmap.addState2(states, bdwPressed)
' Set stdBitmap to button btnBitmap
btnBitmap.Background = stdBitmap
```

### 11.1.5 NinePatchDrawable

This is a copy of Erel's tutorial in the forum.

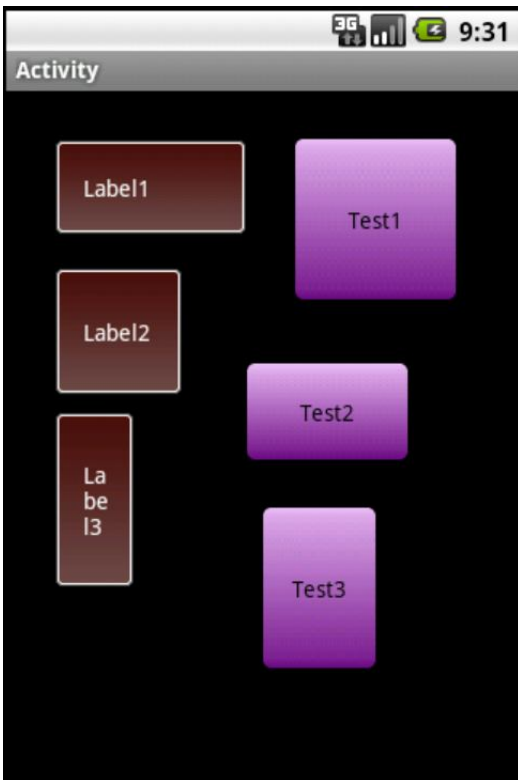
The example code is NinePatchExample in the SourceCode folder.

Android supports a special format of PNG images that can be resized by replicating specific parts of the image.

These images also include padding information.

These images are named nine-patch images.

You can read more about this format here: [Canvas and Drawables | Android Developers](#)



In the example three labels use the same background nine-patch image and three button using another nine-patch image.

Android SDK includes a tool named draw9patch.bat that can help you with building and modifying such images.

This tool is available under: <android>\Tools

You can read more about it here :

[Draw 9-patch | Android Developers](#)

The following steps are required to use a nine patch image as a view background:

- Copy the image to <project folder>\Objects\res\drawable
- Set the image to be read-only (otherwise it will be deleted during compilation).
- Add the following sub to your code (requires Reflection library):

```
Sub SetNinePatchDrawable(Control As View, ImageName As String)
    Dim r As Reflector
    Dim package As String
    Dim id As Int
    package = r.GetStaticField("anywheresoftware.b4a.BA", "packageName")
    id = r.GetStaticField(package & ".R.drawable", ImageName)
    r.Target = r.GetContext
    r.Target = r.RunMethod("getResources")
    Control.Background = r.RunMethod2("getDrawable", id, "java.lang.int")
End Sub
```

For buttons you should use this sub which creates a StateListDrawable from two nine-patch images:

```
Sub SetNinePatchButton(Btn As Button, DefaultImage As String, PressedImage As String)
    Dim r As Reflector
    Dim package As String
    Dim idDefault, idPressed As Int
    package = r.GetStaticField("anywheresoftware.b4a.BA", "packageName")
    idDefault = r.GetStaticField(package & ".R.drawable", DefaultImage)
    idPressed = r.GetStaticField(package & ".R.drawable", PressedImage)
    r.Target = r.GetContext
    r.Target = r.RunMethod("getResources")
    Dim sd As StateListDrawable
    sd.Initialize
    sd.AddState(sd.State_Pressed, r.RunMethod2("getDrawable", idPressed, "java.lang.int"))
    sd.AddCatchAllState(r.RunMethod2("getDrawable", idDefault, "java.lang.int"))
    Btn.Background = sd
End Sub
```

Now you should use this sub to set the views backgrounds:

```
Sub Activity_Create(FirstTime As Boolean)
    Activity.LoadLayout("1")
    SetNinePatchDrawable(Label1, "label_bg")
    SetNinePatchDrawable(Label2, "label_bg")
    SetNinePatchDrawable(Label3, "label_bg")
End Sub
```

### Tips

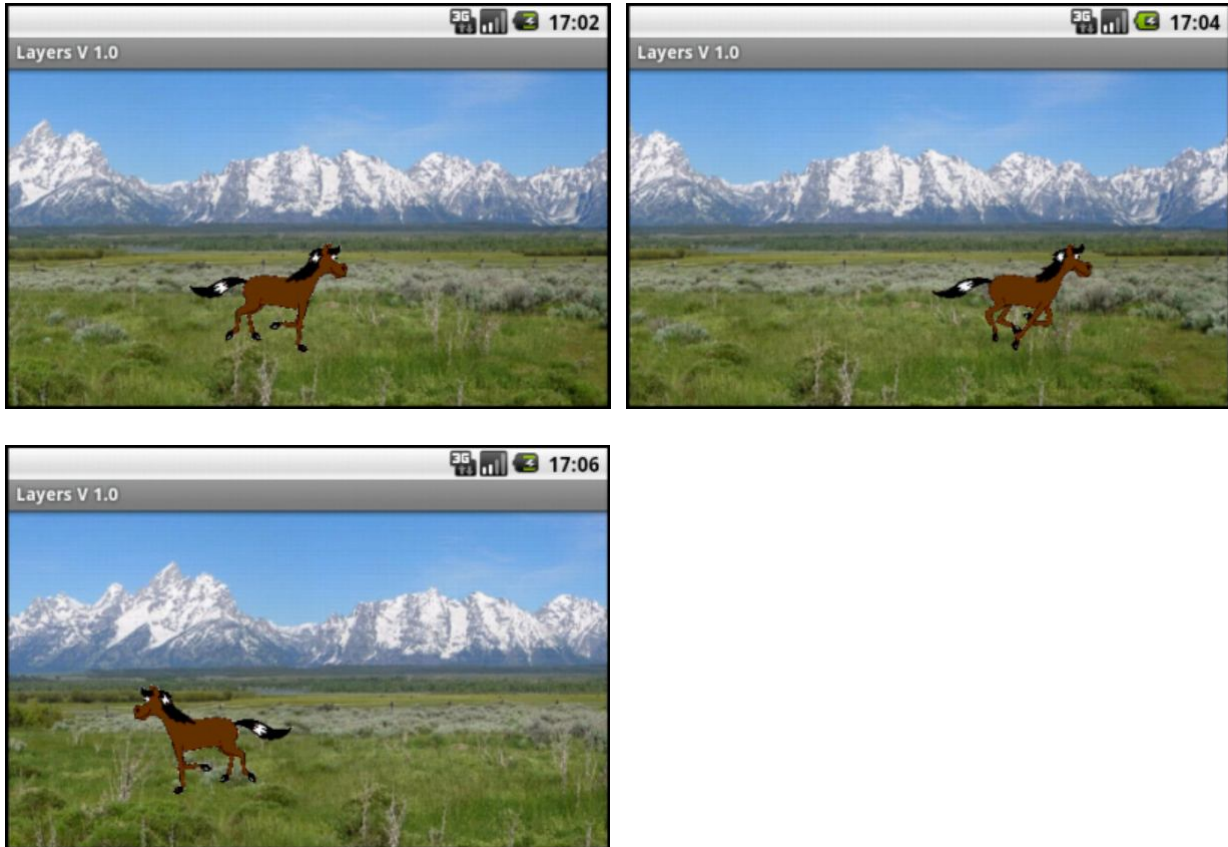
- Don't modify the image files located under res\drawable directly with the draw9patch tool. It removes the read-only attribute and then the image will be deleted.
- The image name must be lower case (allowed characters a - z, 0 - 9, ., \_).
- After adding a new image you should clean the project by choosing Tools - Clean Project. This causes a generated file (R.java) to be recreated and include the new resources.

## 11.2 Layers with Panels / ImageViews / Images

Let's make an example with a movable background, and an image in the foreground.

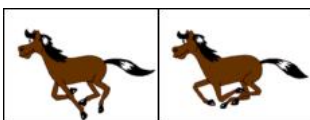
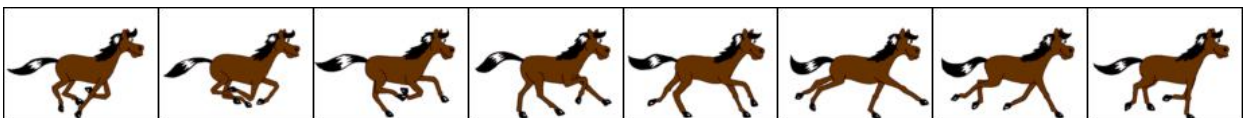
The source code is in the Layers folder.

The background is a landscape and the foreground is galloping horse.



We use:

- Two ImageViews, we could also have used Panels.  
One with the background and the second transparent one for the galloping horse.
- Two Timers.  
One to move the background and one to move the horse image.
- One image for the background.
- Two sets of 8 images for the horse, one set for galloping to the right and the second set for galloping to the left.



etc.

## 11.2.1 Source code

### Definition of process global variables :

```

Sub Process_Global s
    Dim ProgName As String           : ProgName = "Layers"
    Dim ProgVersion As String       : ProgVersion = "V 1.0"
    Dim TimerHorse As Timer
    Dim TimerBackground As Timer
    Dim TimerInterval As Long
End Sub

```

### Definition of global variables :

```

Sub Global s
    Dim Image1 As Int                ' index of the current horse image
    Dim ImageDir As Int              ' direction of the horse image
    Dim ImageNumber As Int           ' number of horse images
        ImageNumber = 8
    Dim imgHorse(2, ImageNumber) As Bitmap ' array horse image bitmaps
    Dim imvBackground As ImageView   ' background ImageView
    Dim imvForeground As ImageView   ' foreground ImageView
    Dim cvsForeground As Canvas      ' canvas for the foreground image
    Dim rectHorse As Rect            ' rectangle of the horse image
    Dim HorseWidth As Int            ' horse image width
    Dim HorseHeight As Int           ' horse image height
    Dim HorseTop As Int              ' horse image top
    Dim HorseLeft As Float           ' current left position of the horse image
    Dim HorseDelta As Float          ' horse move per timer tick
    Dim BackgroundLeft As Float      ' current left position of the background
    Dim BackgroundDelta As Float     ' background move per timer tick
    Dim Scale As Float              ' device scale
End Sub

```

**Initialization of the views and different variables :**

```

Sub Activity_Create(FirstTime As Boolean)
    Dim i As Int

    ' get the device scale
    Dim Lv As LayoutValues
    Lv = GetDeviceLayoutValues
    Scale = Lv.Scale

    ' load the horse images
    ' first index = 0 for galloping to the right
    ' first index = 1 for galloping to the left
    For i = 0 To ImageNumber - 1
        imgHorse(0, i).Initialize(File.DirAssets, "horse0" & i & ".png")
        imgHorse(1, i).Initialize(File.DirAssets, "horse1" & i & ".png")
    Next

    ' initialize variables depending on the device orientation
    If Activity.Width > Activity.Height Then
        HorseDelta = 4dip
        HorseHeight = 40%y
        TimerInterval = 50
    Else
        HorseDelta = 2dip
        HorseHeight = 25%y
        TimerInterval = 80
    End If

    ' initialize the background timer
    TimerBackground.Initialize("TimerBackground", TimerInterval)

    ' initialize the horse timer
    ' we use two times the background timer interval
    TimerHorse.Initialize("TimerHorse", TimerInterval * 2)

    ' calculate the horse images size and their vertical position
    HorseWidth = HorseHeight / imgHorse(0, 0).Height * imgHorse(0, 0).Width
    HorseTop = 65%y - HorseHeight / 2
    rectHorse.Initialize(0, HorseTop, HorseWidth, HorseTop + HorseHeight)

    ' initialize the background
    imgBackground.Initialize("")
    Activity.AddView(imgBackground, 0, 0, 400%y, 100%y)
    imgBackground.Gravity = Gravity.FILL
    imgBackground.Bitmap = LoadBitmap(File.DirAssets, "Wyoming.jpg")
    imgBackground.Left = 0

    ' calculate BackgroundDelta
    ' to have the same number of steps as for the horse
    i = (100%x - HorseWidth) / HorseDelta
    BackgroundDelta = -(imgBackground.Width - 100%x) / 2 / i

    ' initialize the foreground
    imgForeground.Initialize("")
    Activity.AddView(imgForeground, 0, 0, 100%x, 100%y)

    ' initialize the foreground canvas
    cvsForeground.Initialize(imgForeground)

    ' set the foreground to transparent
    Dim rect1 As Rect
    rect1.Initialize(0, 0, imgForeground.Width, imgForeground.Height)
    cvsForeground.DrawRect(rect1, Colors.Transparent, True, 1)
End Sub

```

**Initialization of different variables :****Sub Activity\_Resume**

```

Activity.Title = ProgName & " " & ProgVersion

' initialize the timers
TimerHorse.Enabled = True
TimerBackground.Enabled = True

' set the initial values
HorseLeft = 0
BackgroundLeft = 0
ImageI = 0
ImageDir = 0

' draw the first horse image
DrawHorse(ImageI, 10)
End Sub

```

**Horse timer :****Sub TimerHorse\_Tick**

```

' increase the horse left position
HorseLeft = HorseLeft + HorseDelta

' test if the horse reaches the right or left border
If HorseLeft >= 100%x - HorseWidth - HorseDelta OR HorseLeft <= 0 Then
    BackgroundDelta = - BackgroundDelta
    HorseDelta = - HorseDelta
    HorseLeft = HorseLeft + HorseDelta
    If ImageDir = 0 Then
        ImageDir = 1
    Else
        ImageDir = 0
        ImageBackground.Left = 0
    End If
End If

' update the horse image index
ImageI = ImageI + 1
' reset the image index
If ImageI = ImageNumber Then
    ImageI = 0
End If

' draw the new horse image
DrawHorse(ImageI, HorseLeft)
End Sub

```

**Background timer :****Sub TimerBackground\_Tick**

```

' set the background left position
BackgroundLeft = BackgroundLeft + BackgroundDelta
ImageBackground.Left = BackgroundLeft
End Sub

```

**Drawing routine for the horse images :**

```
Sub DrawHorse(i As Int, x As Float)
    ' drawing routine for the horse image

    ' erase the current horse image, draw a transparent rectangle
    cvsForeground.DrawRect(rectHorse, Colors.Transparent, True, 1)

    ' set the new horse image position
    rectHorse.Left = x
    rectHorse.Right = x + HorseWidth

    ' draw the new horse image
    cvsForeground.DrawBitmap(imgHorse(ImageDir, i), Null, rectHorse)

    ' invalidate (update) the foreground image
    mvForeground.Invalidate2(rectHorse)
End Sub
```

**Stop of the timers when the Activity is paused :**

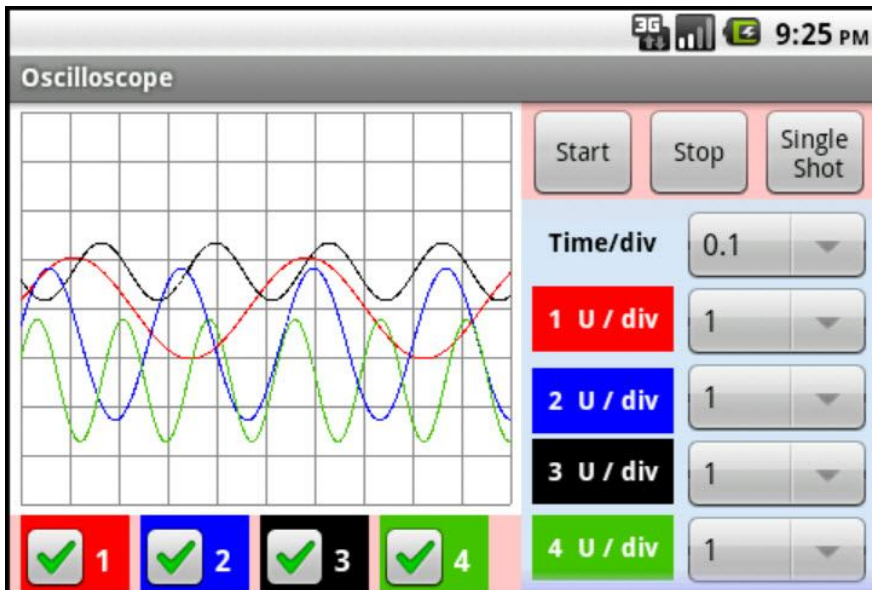
```
Sub Activity_Pause (UserClosed As Boolean)
    ' stop the timers
    TimerHorse.Enabled = True
    TimerBackground.Enabled = True
End Sub
```



## 11.3 Diagrams / Charts

In the first chapter we will draw diagrams to show curves.

In the second we analyze the Charts Framework.



One example program for drawing curves can be found in the forum : the [Oscilloscope](#) project.

### 11.3.1 Diagrams / Graph example program

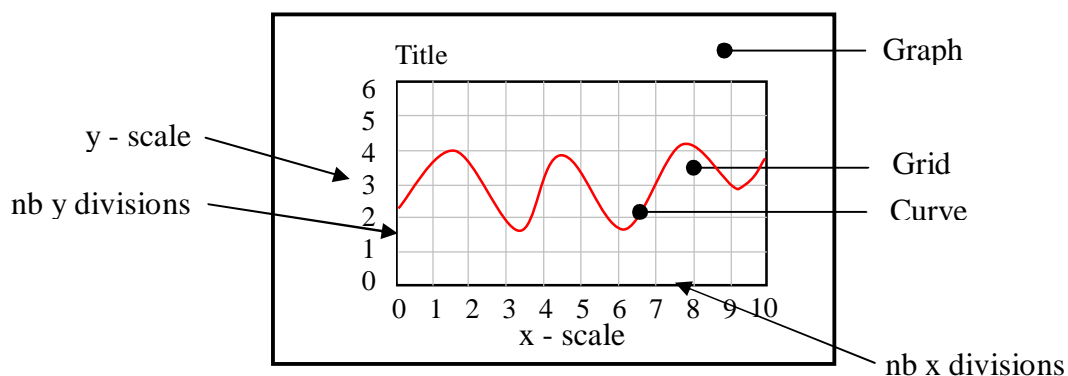
Two dimensional data can be drawn on a plane with a Cartesian coordinate system.

A Cartesian coordinate system in two dimensions (also called a rectangular coordinate system) is defined by an ordered pair of perpendicular lines (axes), a single unit of length for both axes, and an orientation for each axis. The point where the axes meet, is taken as the origin for both, thus turning each axis into a number line.

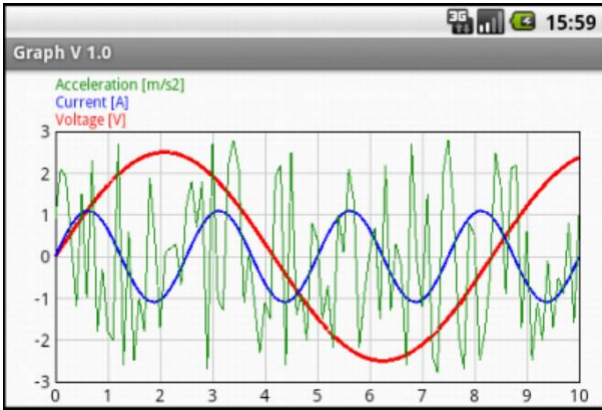
In mathematical illustrations of two-dimensional Cartesian systems, the first coordinate (traditionally called the abscissa or x - axis) is measured along a horizontal axis, oriented from left to right. The second coordinate (the ordinate or y - axis) is then measured along a vertical axis, usually oriented from bottom to top (source Wikipedia).

**The source code of this example program, *Graph*, is joined in the SourceCode folder.**

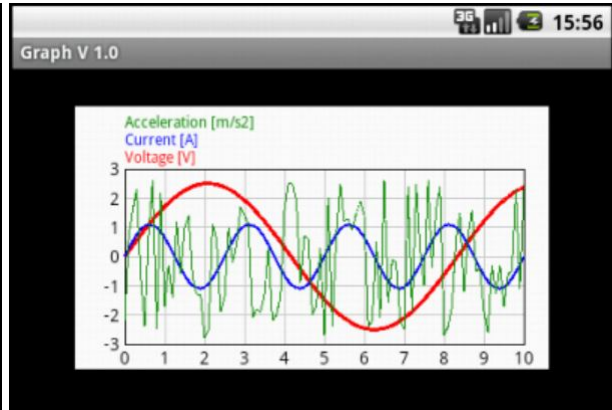
We use a Panel (we could also have used an ImageView) to draw the graphics, we call it the *graph*. On the *graph* we have the *grid*, the surface where the curve is drawn.



The panel can be of different sizes :

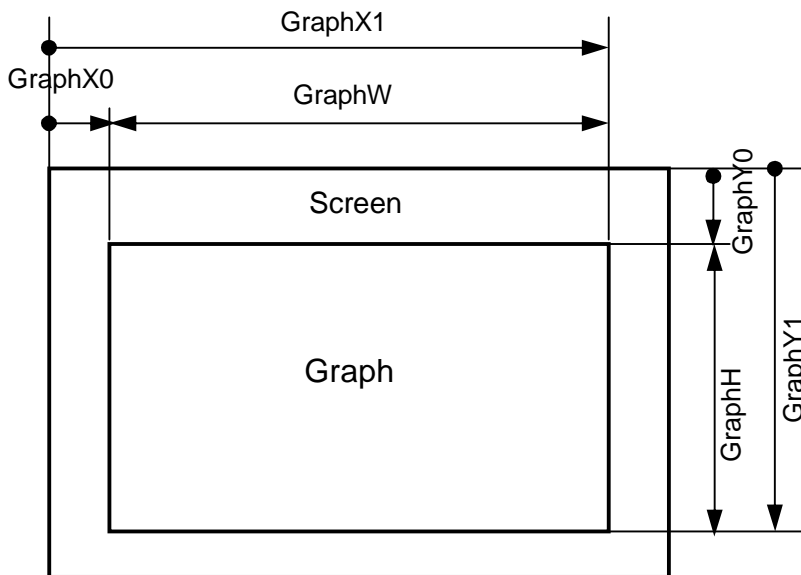


Full screen

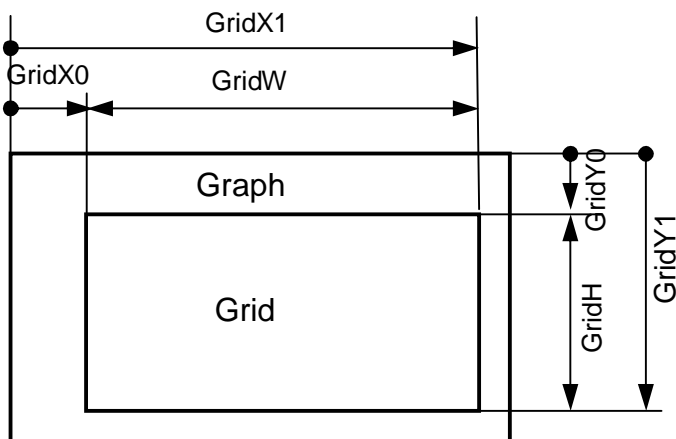


Smaller

**Definition of the Graph variables :**



**Definition of the Grid variables :**



**Source code.**

The code is commented and I think (hope) enough self explanatory.  
It should not be considered as a project for itself but more as a demonstrator.

The code could seem complicated but it really is not. You'll probably notice a lot of variables, but I prefer working with variables rather than directly with numeric values. It's much easier to maintain or modify a code.

**Activity create.**

```
Sub Activity_Create(FirstTime As Boolean)
    GraphInit
    CheckDeviceType

    pnlGraph.Initialize("pnlGraph")
    Activity.AddView(pnlGraph, GraphX0, GraphY0, GraphW, GraphH)

    cvsGraph.Initialize(pnlGraph)
    CurveInit
    GridInit
    If FirstTime Then
        CurveValInit
    End If
    ScaleInit
End Sub
```

**Activity resume.**

```
Sub Activity_Resume
    Private i As Int

    Activity.Title = ProgName & " " & ProgVersion
    GraphClear
    GridDraw
    For i = 0 To CurveNb - 1
        CurveDraw(i)
    Next
End Sub
```

The different dimensions for the graph are all expressed in percentage of the Activity height `d%y`, to fit the different device sizes.

**Initialization of the Graph panel dimensions.** (called from Activity\_Create)

```
Sub GraphInit
    ' initialize the Graph variables
    ' all dimensions are expressed in % of height
    GraphX0 = 3%y
    GraphW = 100%x - 2 * GraphX0
    GraphX1 = GraphX0 + GraphW

    GraphY0 = 3%y
    GraphH = 100%y - 2 * GraphY0
    GraphY1 = GraphY0 + GraphH

    rectGraph.Initialize(0, 0, GraphW, GraphH)
    GraphColor = Colors.White
End Sub
```

The text sizes are also adapted to the device sizes.

I have considered four different device sizes :

- Smartphone 3.5      smartphone with a 3.5 inch screen size
- Smartphone 5      smartphone with a 5 inch screen size
- Tablet 7      tablet with a 7 inch screen size
- Tablet 10      tablet with a 10 inch screen size

**Code to check the device type.** (called from Activity\_Create)

**Sub CheckDeviceType**

```
' check device type, used to define text sizes  
' should be completed if necessary
```

```
Private Iv As LayoutValues
```

```
Iv = GetDeviceLayoutValues
```

```
Select Iv.Scale
```

```
Case 2
```

```
    DeviceType = "Smartphone 5"
```

```
Case 1.5
```

```
    If Iv.Width > 1100 Then
```

```
        DeviceType = "Tablet 7"
```

```
    Else
```

```
        DeviceType = "Smartphone 3.5"
```

```
    End If
```

```
Case 1
```

```
    If Iv.Width > 1100 Then
```

```
        DeviceType = "Tablet 10"
```

```
    Else If Iv.Width < 600 Then
```

```
        DeviceType = "Smartphone 3.5"
```

```
    Else
```

```
        DeviceType = "Tablet 7"
```

```
    End If
```

```
End Select
```

```
End Sub
```

**Initialization of the curves parameters.** (called from Activity\_Create)**Sub CurveInit**

```
' set curve line color
CurveLineColor(0) = Colors.Red
CurveLineColor(1) = Colors.Blue
CurveLineColor(2) = Colors.RGB(10, 140, 0)

' set curve line wisth (stroke)
CurveLineStroke(0) = 3dip
CurveLineStroke(1) = 2dip
CurveLineStroke(2) = 1dip

' set curve text size according to the device type
CurveTextSize = 14
Select DeviceType
Case "Smarphone 3.5"
    CurveTextSize = 14
Case "Smarphone 5"
    CurveTextSize = 22
Case "Tablet 7"
    CurveTextSize = 28
Case "Tablet 10"
    CurveTextSize = 42
End Select

' get the text height
CurveTextHeight = cvsGraph.MeasureStringHeight("Ag", Typeface.DEFAULT, CurveTextSize)
+ 2dip
End Sub
```

**Initialization of the grid.** (called from Activity\_Create)**Sub GridInit**

```

' initialize the Grid variables
' all dimensions are expressed proportional to the curves text height

' horizontal dimensions
GridX0 = 2 * CurveTextHeight
GridW = GraphW - GridX0 - 1.2 * CurveTextHeight
GridX1 = GridX0 + GridW

' vertical dimensions
GridY0 = 3.2 * CurveTextHeight
GridH = GraphH - GridY0 - 1.2 * CurveTextHeight
GridY1 = GridY0 + GridH

' define the number of divisions for each axis
If Iv.Width > Iv.Height Then
    GridNbDi vX = 10
    GridNbDi vY = 6
Else
    GridNbDi vX = 5
    GridNbDi vY = 12
End If

' calculate the division dimensions in pixels
GridDeltaX = GridW / GridNbDi vX
GridDeltaY = GridH / GridNbDi vY

' assign the grid rectangle
rectGrid.Initialize(GridX0, GridY0, GridX1, GridY1)

' set the different colors
GridColor = Colors.White
GridLineColor = Colors.LightGray
GridFrameColor = Colors.Black

ScaleTextColor = Colors.Black

' set scale text size according to the device type
ScaleTextSize = 12
Select DeviceType
Case "Smartphone 3.5"
    ScaleTextSize = 12
Case "Smartphone 5"
    ScaleTextSize = 18
Case "Tablet 7"
    ScaleTextSize = 24
Case "Tablet 10"
    ScaleTextSize = 36
End Select
ScaleTextHeight = cvsGraph.MeasureStringHeight("Ag", Typeface.DEFAULT, ScaleTextSize)
+ 2dip

End Sub

```

**Initialization of the curve values.** (called from Activity\_Create)

These are sample curves, this routine can be adapted to each application.

This routine is called only when `FirstTime = True`.

**Sub CurveVal Init**

```

Private i, n As Int
Private t As Double
Private Amplitude(CurveNb) As Double
Private Offset(CurveNb) As Double
Private Omega(CurveNb) As Double

' set curve amplitude
Amplitude(0) = 2.5
Amplitude(1) = 1.5
Amplitude(2) = .02

' set curve offset
Offset(0) = 0
Offset(1) = 1
Offset(2) = -1

' set curve omega
Omega(0) = 2.4 * cPI
Omega(1) = 8 * cPI

' calculate curve point values
For i = 0 To CurveNb - 1
  For n = 0 To CurveNbPoints
    t = n / 100
    If i = 2 Then
      Curve(i, n) = Offset(i) + Amplitude(i) * Rnd(-100, 100)
    Else
      Curve(i, n) = Offset(i) + Amplitude(i) * Sin(Omega(i) * t)
    End If
  Next n
Next i

' set curve names and units
CurveName(0) = "Voltage"
CurveUnit(0) = "[V]"
CurveName(1) = "Current"
CurveUnit(1) = "[A]"
CurveName(2) = "Acceleration"
CurveUnit(2) = "[m/s2]"

' set scale values
ScaleXMax = CurveNbPoints / 10
ScaleXMin = 0

ScaleYMax = 3
ScaleYMin = -3
End Sub

```

**Initialization of the scales according to the device orientation.** (called from Activity\_Create)**Sub ScaleInit**

```

' initialize the scales according to the grid dimensions.
ScaleXDelta = ScaleXMax / GridNbDi vX
ScaleX = GridW / (ScaleXMax - ScaleXMin)

ScaleYDelta = (ScaleYMax - ScaleYMin) / GridNbDi vY
ScaleY = GridH / (ScaleYMax - ScaleYMin)
End Sub

```

**Drawing of the grid.** (called from Activity\_Resume)**Sub GridDraw**

```

' draw the Grid
Private i As Int
Private x0, y0 As Float

' draw vertical lines
For i = 1 To GridNbDivX - 1
    x0 = GridX0 + i * GridDeltaX
    cvsGraph.DrawLine(x0, GridY0, x0, GridY1, GridLineColor, 1)
Next

' draw horizontal lines
For i = 1 To GridNbDivY - 1
    y0 = GridY0 + i * GridDeltaY
    cvsGraph.DrawLine(GridX0, y0, GridX1, y0, GridLineColor, 1)
Next

' draw the frame
cvsGraph.DrawRect(rectGrid, GridFrameColor, False, 1)

' draw the scales
ScaleYDraw
ScaleXDraw

' invalidate (update) the Graph
Activity.Invalidate
End Sub

```

**Drawing the X scale.** (called from GridDraw)**Sub ScaleXDraw**

```

' draw X scale
Private i As Int
Private txt As String
Private x, y As Float

y = GridY1 + ScaleTextHeight
For i = 0 To GridNbDivX
    txt = (ScaleMin + i * ScaleDelta)
    x = GridX0 + i * GridDeltaX
    cvsGraph.DrawText(txt, x, y, Typeface.DEFAULT, ScaleTextSize, ScaleTextColor,
"CENTER")
Next
End Sub

```



**Drawing of the Y scale.** (called from GridDraw)**Sub ScaleYDraw**

```

' draw Y scale
Private i As Int
Private txt As String
Private x, y As Float

x = GridX0 - ScaleTextHeight / 3
For i = 0 To GridNbDivY
    txt = (ScaleYMax - i * ScaleYDelta)
    y = GridY0 + ScaleTextHeight/3 + i * GridDeltaY
    cvsGraph.DrawText(txt, x, y, Typeface.DEFAULT, ScaleTextSize, ScaleTextColor,
"RIGHT")
Next
End Sub

```

**Drawing the curves.** (called from Activity\_Resume)**Sub CurveDraw(i As Int)**

```

' draw the curve of index i
Private n As Int
Private d, th, x0, y0, x1, y1 As Float
Private TextHeight As Float

' draw the curve
x0 = GridX0
y0 = GridY0 + (ScaleYMax - Curve(i, 0)) * ScaleY
d = GridW / CurveNbPoints
For n = 1 To CurveNbPoints
    x1 = GridX0 + n * d
    y1 = GridY0 + (ScaleYMax - Curve(i, n)) * ScaleY
    cvsGraph.DrawLine(x0, y0, x1, y1, CurveLineColor(i), CurveLineStyle(i))
    x0 = x1
    y0 = y1
Next

' get the text height
TextHeight = cvsGraph.MeasureStringHeight("Ag", Typeface.DEFAULT, CurveTextSize) +
2dip

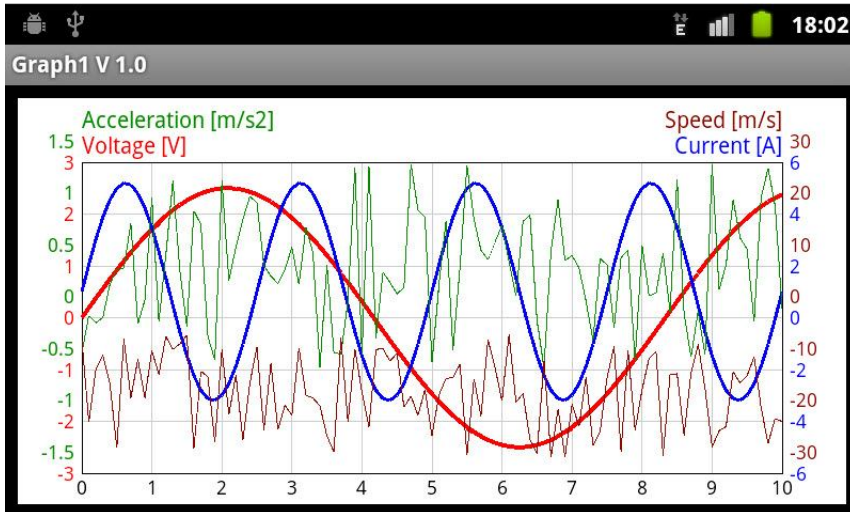
' draw curve name
y0 = GridY0 - TextHeight / 3
cvsGraph.DrawText(CurveName(i) & " " & CurveUnit(i), GridX0, y0 - i * TextHeight,
Typeface.DEFAULT, CurveTextSize, CurveLineColor(i), "LEFT")
End Sub

```

### 11.3.2 Second Graph program

Graph1 is a second diagram example program, it's an evolution of the previous one, that uses a different scale for each curve.

The source code of this example program, *Graph1*, is joined in the SourceCode folder.



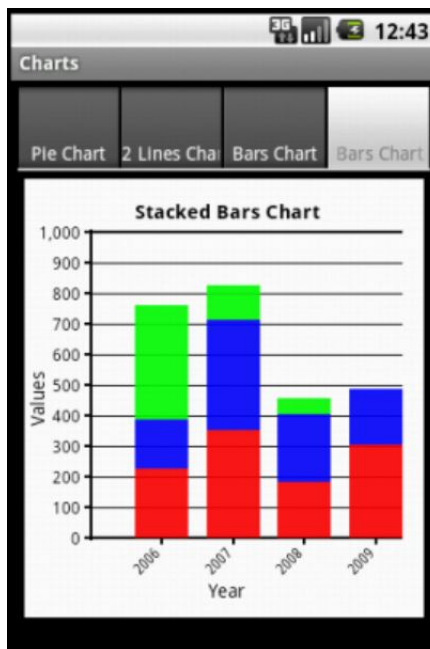
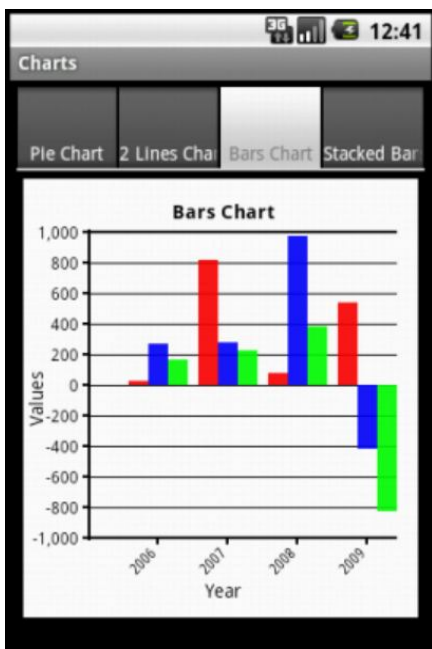
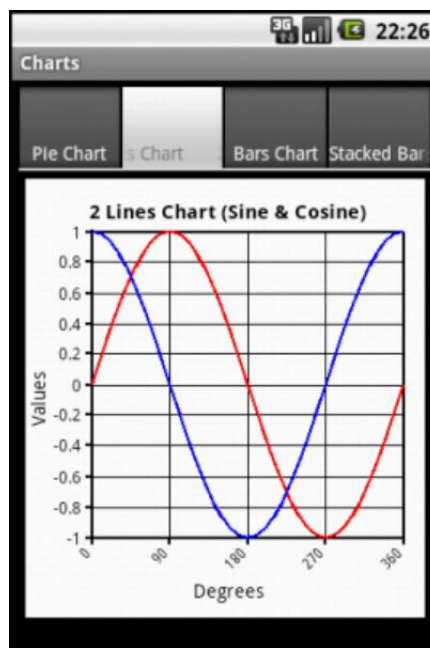
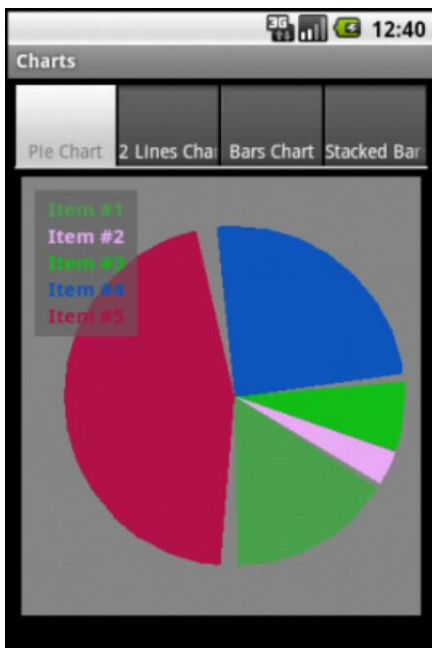
### 11.3.3 Charts Framework

The Charts Framework module allows to draw several types of diagrams:

- Pie charts
- Bar charts
- Stacked Bar charts
- Curves

It can be downloaded here : [Charts Framework](#)

Attention : The Charts Framework is a code module and not a library.



The goal of this chapter is to show how to use the Charts Framework and not to explain how its code is build. The joined source codes allows to play with each kind of chart type.

### 11.3.3.1 Pie Chart

Source code in the Charts\PieChart folder.

#### Sub Globals

```
Private pnlPie As Panel
```

```
End Sub
```

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

```
CreatePieTab
```

```
End Sub
```

#### Sub CreatePieTab

```
' initialize the panel to display the pie chart
```

```
pnlPie.Initialize("pnlPie")
```

```
Activity.AddView(pnlPie, 10%x, 10%y, 80%x, 80%y)
```

```
' initialize the pie data
```

```
Private PD As PieData
```

```
PD.Initialize
```

```
PD.Target = pnlPie ' Set the target view
```

```
' Add the items.
```

```
' The last parameter is the color. Passing 0 will make it a random color.
```

```
Charts.AddPieItem(PD, "Item #1", 120, 0)
```

```
Charts.AddPieItem(PD, "Item #2", 25, 0)
```

```
Charts.AddPieItem(PD, "Item #3", 50, 0)
```

```
Charts.AddPieItem(PD, "Item #4", 190, 0)
```

```
Charts.AddPieItem(PD, "Item #5", 350, 0)
```

```
' Total size of gaps between slices. Set to 0 for no gaps.
```

```
PD.GapDegrees = 20
```

```
' The background color of the legend bitmap.
```

```
PD.LegendBackColor = Colors.ARGB(150, 100, 100, 100)
```

```
' PD.LegendBackColor = Colors.White
```

```
' This call draws the pie.
```

```
' PD - The pie data
```

```
' Colors.Gray - The view's background color
```

```
' True - Create a legend bitmap.
```

```
Dim Legend As Bitmap
```

```
Legend = Charts.DrawPie(PD, Colors.Gray, True)
```

```
' Legend = Charts.DrawPie(PD, Colors.LightGray, True)
```

```
' Legend = Charts.DrawPie(PD, Colors.White, True)
```

```
If Legend.IsInitialized Then
```

```
' Initialize the legend ImageView
```

```
Private ImageView1 As ImageView
```

```
ImageView1.Initialize("")
```

```
ImageView1.SetBackgroundImage(Legend)
```

```
' Add the legend ImageView to the Pie Panel
```

```
pnlPie.AddView(ImageView1, 10dip, 10dip, Legend.Width, Legend.Height)
```

```
End If
```

```
End Sub
```

**Initialize the Panel for the Pie Chart :**

```
' initialize the panel to display the pie chart
pnlPie.Initialize("pnlPie")
Activity.AddView(pnlPie, 10%x, 10%y, 80%x, 80%y)
```

10%x, 10%y, 80%x, 80%y are the position and dimensions of the pie panel.

**Initialize the Pie data :**

```
' initialize the pie data
Dim PD As PieData
PD.Initialize
PD.Target = pnlPie ' Set the target view
```

**Add the pie chart data :**

```
' Add the items.
' The last parameter is the color. Passing 0 will make it a random color.
Charts.AddPieItem(PD, "Item #1", 120, 0)
Charts.AddPieItem(PD, "Item #2", 25, 0)
Charts.AddPieItem(PD, "Item #3", 50, 0)
Charts.AddPieItem(PD, "Item #4", 190, 0)
Charts.AddPieItem(PD, "Item #5", 350, 0)
```

Drawing routine : `Charts.AddPieItem(PD, "Item #1", 120, 0)`

Where :

- PD is the PieData object
- "Item #1" is the item title
- 120 is the item value
- 0 is the color, enter 0 for random color

**Set the gap value :**

```
' Total size of gaps between slices. Set to 0 for no gaps.
PD.GapDegrees = 20
```

This is the total gap between the pies in degrees. Enter 0 for no gap.

**Set the legend background color :**

```
' The background color of the legend bitmap.
PD.LegendBackColor = Colors.ARGB(150, 100, 100, 100)
' PD.LegendBackColor = Colors.White
```

You can play with different colors, the original color is partially transparent to see the chart behind the legend.

**Calls the pie drawing routine :**

```
' This call draws the pie.
' PD - The pie data
' Colors.Gray - The view's background color
' True - Create a legend bitmap.
Dim Legend As Bitmap
Legend = Charts.DrawPie(PD, Colors.Gray, True)
```

Drawing routine: `Legend = Charts.DrawPie(PD, Colors.Gray, True)`

Where :

- LegendBitmap of the legend image returned by the routine
- PD PieData object
- Colors.Gray Background color of the pie chart.
- True Boolean variable defining whether a legend image should be drawn.

**Check if the legend bitmap does exist :**

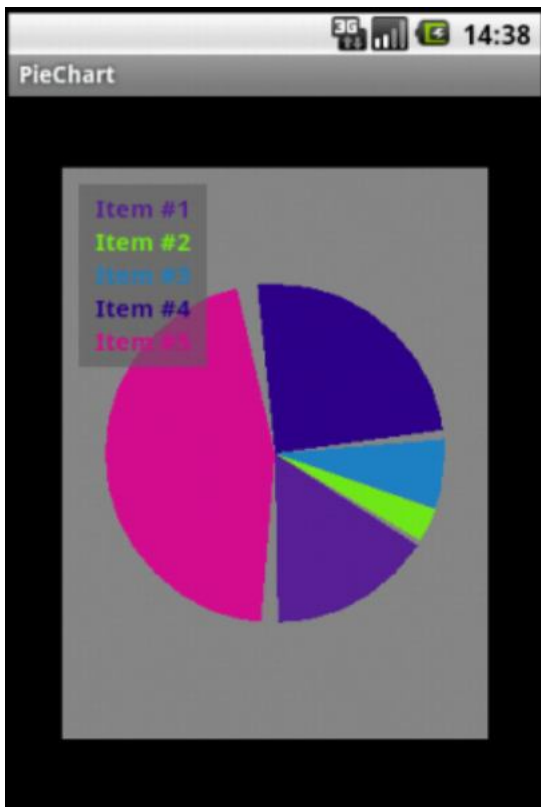
```
If Legend.IsInitialized Then
```

If True then:

**Initialize the legend ImageView, set it's background image and add it to the pie panel :**

```
' Initialize the Legend ImageView
Dim ImageView1 As ImageView
ImageView1.Initialize("")
ImageView1.SetBackgroundImage(Legend)

' Add the Legend ImageView to the Pie Panel
pnlPie.AddView(ImageView1, 10dip, 10dip, Legend.Width, Legend.Height)
End If
End Sub
```



### 11.3.3.2 Bar Chart

Source code in the Charts\BarChart folder.

#### Sub Globals

```
Private pnlBars As Panel
End Sub
```

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

```
CreateBarsTab
End Sub
```

#### Sub CreateBarsTab

```
' initialize the panel to display the bar chart
pnlBars.Initialize("pnlBars")
Activity.AddView(pnlBars, 10%x, 10%y, 80%x, 80%y)

' initialize the bar data
Private BD As BarData
BD.Initialize
BD.Target = pnlBars
BD.BarsWidth = 15dip
BD.Stacked = False

' set the bar colors
Charts.AddBarColor(BD, MakeTransparent(Colors.Red, 230)) ' First bar color
Charts.AddBarColor(BD, MakeTransparent(Colors.Blue, 230))
Charts.AddBarColor(BD, MakeTransparent(Colors.Green, 230))

' Add the items.
For i = 1 To 4
    Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(-1000, 1000), Rnd(-1000, 1000),
Rnd(-1000, 1000)))
'    Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(0, 1000), Rnd(0, 1000), Rnd(0,
1000)))
Next

' Initialize the graph object
' Set the bar chart parameters
Private G As Graph
G.Initialize
G.Title = "Bars Chart"
G.XAxis = "Year"
G.YAxis = "Values"
G.YStart = -1000 ' min vertical scale
' G.YStart = 0
G.YEnd = 1000 ' max vertical scale
G.YInterval = 200 ' vertical scale divisions
G.AxisColor = Colors.Black
Charts.DrawBarsChart(G, BD, Colors.White)
End Sub

Sub MakeTransparent(Color As Int, Alpha As Int) As Int
Return Bit.And(Color, Bit.Or(0x0FFFFFFF, Bit.ShiftLeft(Alpha, 24)))
End Sub
```

**Initialize the Panel for the Bar Chart :**

```
' initialize the panel to display the bar chart
pnl Bars.Initialize("pnl Bars")
Activity.AddView(pnl Bars, 10%x, 10%y, 80%x, 80%y)
```

10%x, 10%y, 80%x, 80%y are the position and dimensions of the bar panel.

**Initialize the Bar data :**

```
' initialize the bar data
Private BD As BarData
BD.Initialize
BD.Target = pnl Bars
BD.BarsWidth = 15dip
BD.Stacked = False
```

BD.Stacked = False indicates that it is a BarChart and not a StackedBarChart.

**Set the bar colors :**

```
' set the bar colors
Charts.AddBarColor(BD, MakeTransparent(Colors.Red, 230)) 'First bar color
Charts.AddBarColor(BD, MakeTransparent(Colors.Blue, 230))
Charts.AddBarColor(BD, MakeTransparent(Colors.Green, 230))
```

**Add the bar data :**

```
' Add the items.
For i = 1 To 4
    Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(-1000, 1000), Rnd(-1000, 1000),
    Rnd(-1000, 1000)))
'    Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(0, 1000), Rnd(0, 1000), Rnd(0,
    1000)))
Next
```

A bar chart can have  $n$  sets of  $m$  values.

Loop to add the sets, in the example  $n = 4$ .

```
For i = 1 To 4
```

Routine adding the bar values, in the example  $m = 3$  :

```
Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(-1000, 1000), Rnd(-1000, 1000),
    Rnd(-1000, 1000)))
```

Where :

- BD BarData object
- 2005 + i Horizontal axis tag
- Array As Float Array of values the  $m$  values per set.

The bar chart can draw negative and positive values.

This is not the case for stacked bar charts.



**Initialize the graph object, set the bar chart parameters and draw the chart :**

```

' Initialize the graph object
' Set the bar chart parameters and draw the chart
Private G As Graph
G.Initialize
G.Title = "Bars Chart"
G.XAxis = "Year"
G.YAxis = "Values"
G.YStart = -1000           ' min vertical scale
' G.YStart = 0
G.YEnd = 1000             ' max vertical scale
G.YInterval = 200        ' vertical scale divisions
G.AxisColor = Colors.Black
Charts.DrawBarsChart(G, BD, Colors.White)
End Sub

```

Drawing routine : `Charts.DrawBarsChart(G, BD, Colors.White)`

Where:

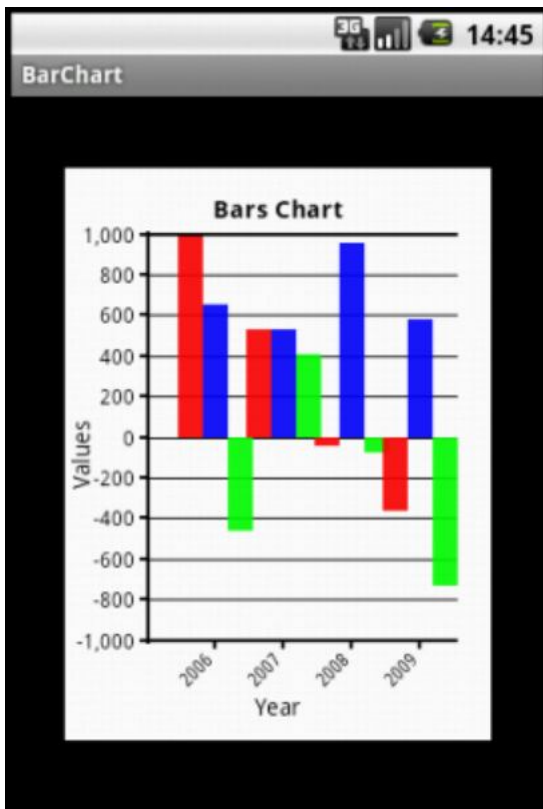
- G Graph object
- BD BarData object
- `Colors.White` Background color of the bar chart panel

**Routine to set a color partially transparent :**

```

Sub MakeTransparent(Color As Int, Alpha As Int) As Int
Return Bit.And(Color, Bit.Or(0x00FFFFFF, Bit.ShiftLeft(Alpha, 24)))
End Sub

```



### 11.3.3.3 Stacked Bar Chart

Source code in the Charts\StackedBarChart folder.

For details, look at the Bar Chart chapter.

#### Sub Globals

```
Private pnlStackedBars As Panel
End Sub
```

```
Sub Activity_Create(FirstTime As Boolean)
    CreateStackedBarsTab
End Sub
```

#### Sub CreateStackedBarsTab

```
' initialize the panel to display the stacked bar chart
pnlStackedBars.Initialize("pnlStackedBars")
Activity.AddView(pnlStackedBars, 10%x, 10%y, 80%x, 80%y)

' initialize the bar data
Private BD As BarData
BD.Initialize
BD.Target = pnlStackedBars
BD.BarsWidth = 40dip
BD.Stacked = True 'Makes it a stacked bars chart

' set the bar colors
Charts.AddBarColor(BD, MakeTransparent(Colors.Red, 230)) 'First bar color
Charts.AddBarColor(BD, MakeTransparent(Colors.Blue, 230))
Charts.AddBarColor(BD, MakeTransparent(Colors.Green, 230))

' Add the items.
For i = 1 To 4
    Charts.AddBarPoint(BD, 2005 + i, Array As Float(Rnd(0, 400), Rnd(0, 400), Rnd(0,
400)))
Next

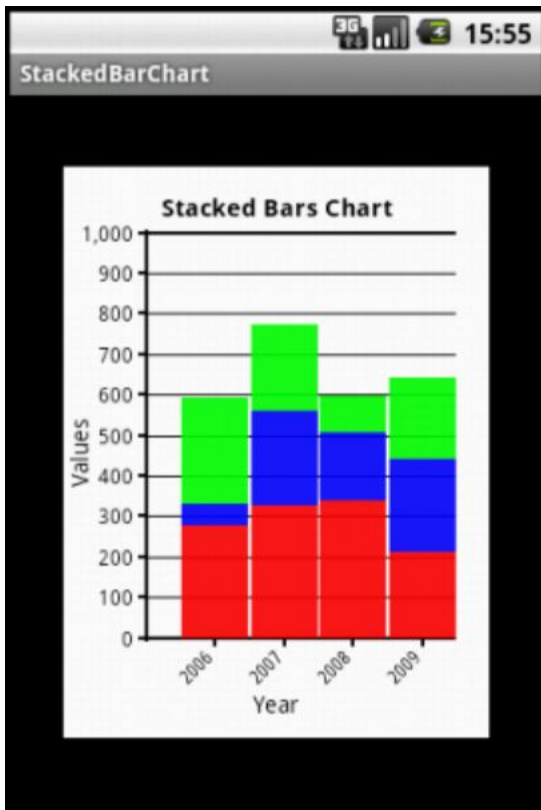
' Initialize the graph object
' Set the bar chart parameters
Private G As Graph
G.Initialize
G.Title = "Stacked Bars Chart"
G.XAxis = "Year"
G.YAxis = "Values"
G.YStart = 0
G.YEnd = 1000
G.YInterval = 100
G.AxisColor = Colors.Black
Charts.DrawBarsChart(G, BD, Colors.White)
End Sub
```

```
Sub MakeTransparent(Color As Int, Alpha As Int) As Int
    Return Bit.And(Color, Bit.Or(0x00FFFFFF, Bit.ShiftLeft(Alpha, 24)))
End Sub
```

The routine is exactly the same as for Bar Charts.

The only difference is the bar with that is wider and the BD.Stacked parameter is True.

```
BD.BarsWidth = 40dip
BD.Stacked = True 'Makes it a stacked bars chart
```



### 11.3.3.4 Lines Chart

Source code in the Charts\LinesChart folder.

The routine can draw one or more lines, the difference is explained below.

#### Sub Globals

```
Private pnlLines As Panel
End Sub
```

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

```
CreateLinesTab
End Sub
```

#### Sub CreateLinesTab

```
' Initialize the panel to display the lines chart
pnlLines.Initialize("pnlLines")
Activity.AddView(pnlLines, 10%x, 10%y, 80%x, 80%y)

' Initialize the line data
Dim LD As LineData
LD.Initialize
LD.Target = pnlLines

' Set the line colors
Charts.AddLineColor(LD, Colors.Red) 'First line color
Charts.AddLineColor(LD, Colors.Blue) 'Second line color

' Add the line points.
For i = 0 To 360 Step 10
    ' In the case of 2 lines or more we are adding an array of values.
    ' One for each line.
    ' Make sure to create an array for each point.
    ' You cannot reuse a single array for all points.
    Charts.AddLineMultiplePoints(LD, i, Array As Float(SinD(i), CosD(i)), i Mod 90 = 0)
Next

' Initialize the graph object
' Set the line chart parameters and draw the line chart
Private G As Graph
G.Initialize
G.Title = "2 Lines Chart (Sine & Cosine)"
G.XAxis = "Degrees"
G.YAxis = "Values"
G.YStart = -1
G.YEnd = 1
G.YInterval = 0.2
G.AxisColor = Colors.Black
Charts.DrawLineChart(G, LD, Colors.White)
End Sub
```

**Initialize the Panel for the Lines Chart :**

```
' Initialize the panel to display the lines chart
pnlLines.Initialize("pnlLines")
Activity.AddView(pnlLines, 10%x, 10%y, 80%x, 80%y)
```

10%x, 10%y, 80%x, 80%y are the position and dimensions of the lines panel.

**Initialize the Lines data :**

```
' Initialize the line data
Private LD As LineData
LD.Initialize
LD.Target = pnlLines
```

**Set the line colors :**

```
' Set the line colors
Charts.AddLineColor(LD, Colors.Red) 'First line color
Charts.AddLineColor(LD, Colors.Blue) 'Second line color
```

**Add the lines data :**

```
' Add the line points.
For i = 0 To 360 Step 10
' In this case we are adding an array of two values. One for each line.
' Make sure to create an array for each point.
' You cannot reuse a single Array For all points.
Charts.AddLineMultiPoints(LD, i, Array As Float(SinD(i), CosD(i)), i Mod 90 = 0)
Next
```

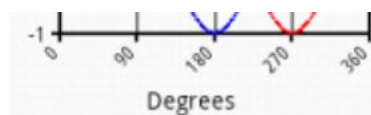
In this part of the routine we generate the values for each line at each point.

Routine adding the line values :

```
Charts.AddLineMultiPoints(LD, i, Array As Float(SinD(i), CosD(i)), i Mod 90 = 0)
```

Where :

- LD LineData object
- i X value of the horizontal axis, can be different than  $i$ .
- `Array As Float(SinD(i), CosD(i))`  
Array of Y values for index  $i$   
In the example there are 2 lines : `SinD(i)` and `CosD(i)`  
For a single line chart we could have this statement :  
`Charts.AddLineMultiPoints(LD, i, SinD(i), i Mod 90 = 0)`  
For a three line chart we could have this statement :  
`Charts.AddLineMultiPoints(LD, i, Array As Float(Val 1, Val 2, Val 3), i Mod 90 = 0)`
- `i Mod 90 = 0` When true then a horizontal tag, with the value of  $i$ , is displayed on the horizontal axis.  
`i Mod 90 = 0` is equal to the remainder of  $i / 90$ ,  
the remainder is true only when  $i = 0, 90, 180, 270$  or  $360$ .



The general routine would look like this:

```
Charts.AddLineMultiPoints(LD, x, Array As Float(Val 1, Val 2, ... , Val N), TagFunction)
```

TagFunction is a function that is true when we want to have a tag on the horizontal axis.

**Initialize the graph object, set the lines chart parameters and draw the chart :**

```

' Initialize the graph object
' Set the line chart parameters and draw the line chart
Private G As Graph
G.Initialize
G.Title = "2 Lines Chart (Sine & Cosine)"
G.XAxis = "Degrees"
G.YAxis = "Values"
G.YStart = -1
G.YEnd = 1
G.YInterval = 0.2
G.AxisColor = Colors.Black
Charts.DrawLineChart(G, LD, Colors.White)
End Sub

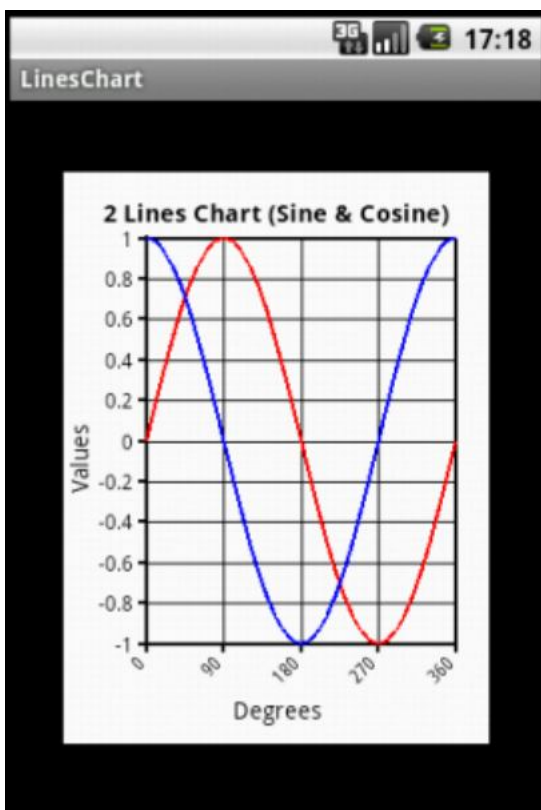
```

Drawing routine : `Charts.DrawLineChart(G, LD, Colors.White)`

Where :

- G                                      Graph object
- LD                                      LineData object
- `Colors.White` Chart background color

There is only one text and one scale for the vertical axis.



Other example with 3 lines.

Source code LinesChart1.

### Sub CreateLinesTab

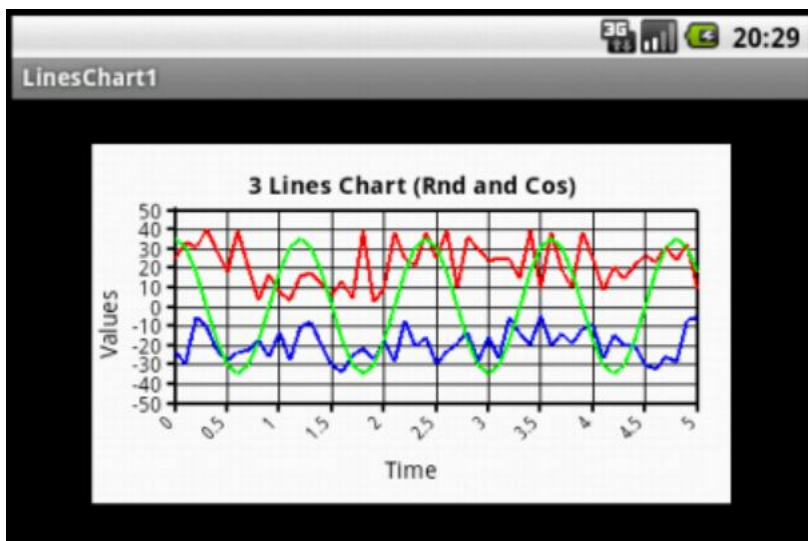
```
' Initialize the panel to display the lines chart
pnlLines.Initialize("pnlLines")
Activity.AddView(pnlLines, 10%x, 10%y, 80%x, 80%y)

' Initialize the line data
Private LD As LineData
LD.Initialize
LD.Target = pnlLines

' Set the line colors
Charts.AddLineColor(LD, Colors.Red) 'First line color
Charts.AddLineColor(LD, Colors.Blue) 'Second line color
Charts.AddLineColor(LD, Colors.Green) 'Third line color

' Add the line points.
Private x As Float
For i = 0 To 500 Step 10
    ' In the case of 2 lines or more we are adding an array of values.
    ' One for each line.
    ' Make sure to create an array for each point.
    ' You cannot reuse a single array for all points.
    x = i / 100
    Charts.AddLineMultiplePoints(LD, x, Array As Float(Rnd(-20,21) + 20, Rnd(-15,16) -
20, CosD(3 * i) * 35), i Mod 50 = 0)
Next

' Initialize the graph object
' Set the line chart parameters and draw the line chart
Private G As Graph
G.Initialize
G.Title = "3 Lines Chart (Rnd and Cos)"
G.XAxis = "Time"
G.YAxis = "Values"
G.YStart = -50
G.YEnd = 50
G.YInterval = 10
G.AxisColor = Colors.Black
Charts.DrawLineChart(G, LD, Colors.White)
End Sub
```



Horizontal tags every 50 increments :

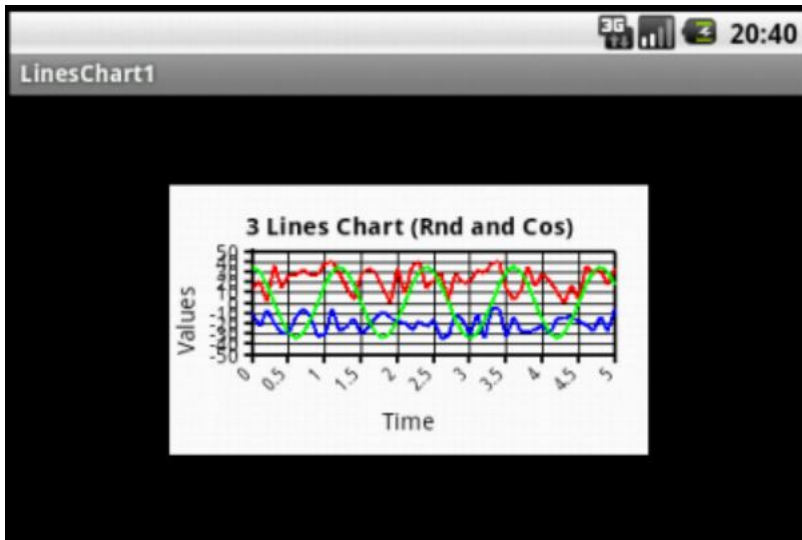
`i Mod 50 = 0`

The x values are ( $x = i / 100$ ):  
0, 0.5, 1, 1.5, 2 etc.

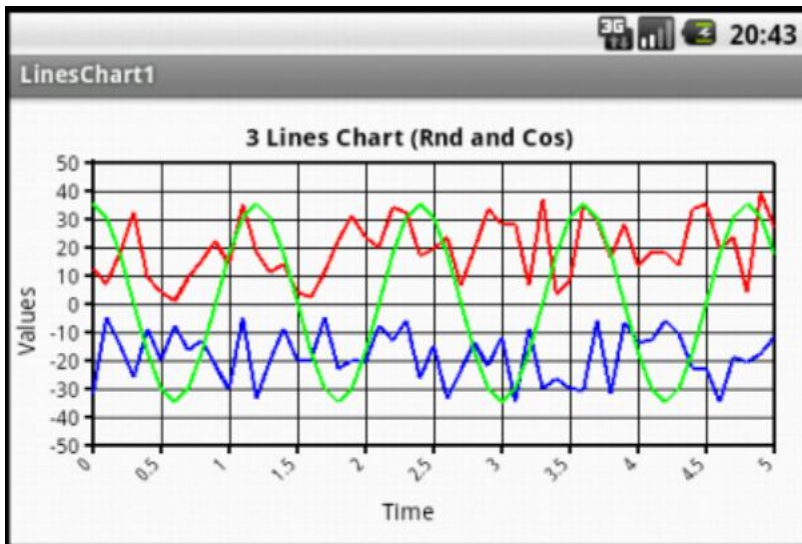
Size :

`10%x, 10%y, 80%x, 80%y`

The charts can be of different sizes, but it's up to you to adapt the scales. In the example below the vertical scale tags are overlapping.



Size :  
20%x, 20%y, 60%x, 60%y



Size :  
0, 0, 100%x, 100%y

The margins and the text sizes remain the same, independent of the chart size. If really you want a small size chart you could change the margins and the text sizes in the Charts code, but be careful because in that case your code module becomes a custom one.



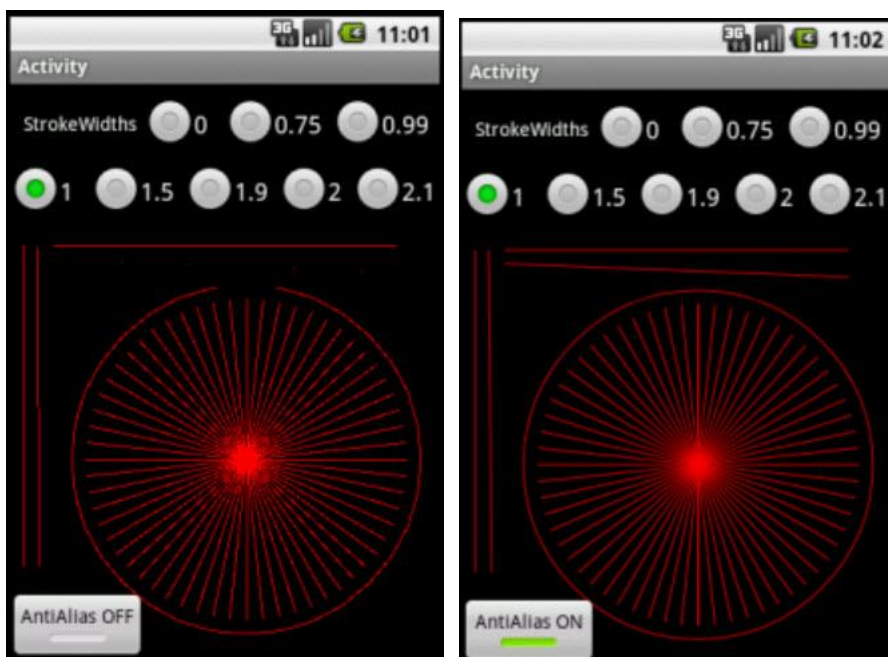
## 11.4 Antialiasing filter

Erel posted the code to set the antialiasing filter in a thread where a user asked for that possibility.

Here is the code:

```
Sub SetAntiAlias (c As Canvas, Active As Int)
    ' Active = 0 filter OFF
    ' Active = 1 filter ON
    ' Active = 2 filter ON for Bitmaps
    Private r As Reflector
    Dim NativeCanvas As Object
    r.Target = c
    NativeCanvas = r.GetField("canvas")
    Private PaintFlagsDrawFilter As Object
    PaintFlagsDrawFilter = r.CreateObject2("android.graphics.PaintFlagsDrawFilter", _
        Array As Object(0, Active), Array As String("java.lang.int", "java.lang.int"))
    r.Target = NativeCanvas
    r.RunMethod4("setDrawFilter", Array As Object(PaintFlagsDrawFilter), _
        Array As String("android.graphics.DrawFilter"))
End Sub
```

An example program to test it with simple lines can be found on the forum : [DrawLines](#)



## 12 Class modules

### 12.1 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's start with an example, the source code: *Persons* in the Classes/ Persons 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 activity or service that called the 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.

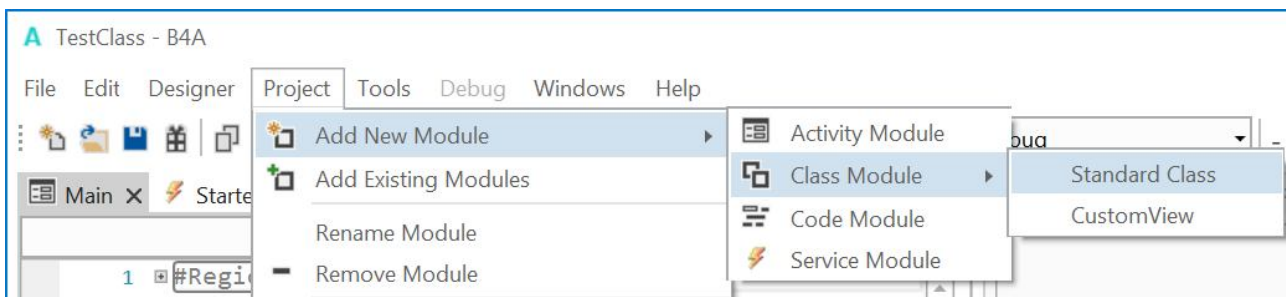
Classes store a reference to the context of the activity or service module that called the Initialize sub. This means that classes objects share the same life cycle as the service or activity that initialized them.

Code modules are somewhat similar to singleton or static classes.

### 12.1.1 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](#)

## 12.1.2 Polymorphism

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

Basic4android 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 Classes/Draw folder.

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

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

Sub Draw(c As Canvas)
    Private r As Rect
    r.Initialize(mx, my, mx + mLength, my + mLength)
    c.DrawRect(r, Colors.White, 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 (x As Int, y As Int, radius As Int)
    mx = x
    my = y
    mRadius = radius
End Sub

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

In the main module we create a list 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 sq1, sq2 As Square
    Private circle1 As Circle
    shapes.Initialize
    sq1.Initialize(shapes, 100dip, 100dip, 50dip)
    sq2.Initialize(shapes, 2dip, 2dip, 100dip)
    circle1.Initialize(shapes, 50%x, 50%y, 100dip)
    DrawAllShapes
End Sub

Sub DrawAllShapes
    For i = 0 To shapes.Size - 1
        Log(shapes.Get(i))
        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.

### 12.1.3 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 could 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

```

### 12.1.4 Activity object

This point is related to the 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.

## 12.2 Standard class structure

Default layout of a standard class:

```

Main Starter ClsStandard X ClsCustomView
Initialize
1
2 Sub Class_Globals
3
4 End Sub
5
6 'Initializes the object. You can add pa
7 Public Sub Initialize
8
9 End Sub

```

Only two routines are predefined:

**Sub Class\_Global s** - This sub is similar to the activity Globals sub. These variables will be the class global variables (sometimes referred to instance variables or instance members).

**Sub Initialize** - A class object should 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 activity or service).

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

Example:

```

'Class Person module
Sub Class_Global s
    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
    Dim diff As Long
    diff = Date - BirthDate
    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.Parse("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)
```



## 12.3 Classes from the forum

A certain number of examples exist in the forum.

Below a non exhaustive list with comments from the developers.

- [TableView - Supports tables of any size](#) Erel  
This class is a much improved version of the ScrollView based Table.
- [CustomListView](#) Erel  
A flexible list based on ScrollView
- [CameraEx](#) Erel  
CameraEx class wraps the Camera object and using reflection and other code it extends its functionality.
- [CheckList](#) Informatix  
I created a class to manage a list with checkboxes (but not only, you can use it for any kind of list with rows). The layout is highly customizable and you can even extend a clicked item with extra content. You can rearrange the items programmatically or manually (drag & drop).
- [SlideMenu](#) corwin42  
This is a class that implements a sliding menu as seen in many apps like Google+, Evernote, Facebook etc.
- [ScrollPanel](#) Informatix  
This class displays a small panel beside the vertical scrollbar in a scrollview. You can display what you want in its label and drag it to quickly scroll.
- [Floating Windows](#) Informatix  
With this class, you can create floating windows, move them with the finger, dock them, stick them to an edge, maximize them, customize their action bar... You fill them as you fill a panel.
- [ActionBar](#) Informatix  
This class allows to create [Action Bars](#).  
There was already a good library to do that (AHActionBar) but I needed more features and more flexibility.
- [Animated Sliding Menu](#) NJDude  
This class will allow you to create animated sliding menus.
- [ICS Like Horizontal and Vertical Seekbars](#) mabool  
This class implements **ICS** Like SeekBars.
- [ClsWheel](#) klaus  
It allows to display different data input screens with wheels. What can be done ? You can define five different types of Wheel input screens.

If you want to develop your own classes you should have a look at the classes above to see and understand what has been done and how.

## 12.4 Custom views

With classes you can add your own custom views which can be based on standard views but with more functions.

### 12.4.1 Custom view class structure

Default layout of a CustomView class:

```

1
2 #Event: ExampleEvent (Value As Int)
3 #DesignerProperty: Key: BooleanExample, DisplayName: BooleanExample
4 #DesignerProperty: Key: IntExample, DisplayName: IntExample
5 #DesignerProperty: Key: stringWithListExample, DisplayName: stringWithListExample
6 #DesignerProperty: Key: StringExample, DisplayName: StringExample
7 #DesignerProperty: Key: ColorExample, DisplayName: ColorExample
8 Sub Class_Globals
9     Private eventName As String 'ignore
10    Private Callback As Object 'ignore
11    Private mBase As Panel
12 End Sub
13
14 Public Sub Initialize (vCallback As Object, vEventName As String)
15     eventName = vEventName
16     Callback = vCallback
17 End Sub
18
19 Public Sub DesignerCreateView (Base As Panel, lbl As Label)
20     mBase = Base
21
22 End Sub
23
24 Public Sub GetBase As Panel
25     Return mBase
26 End Sub

```

Several declarations and routines are predefined:

#### 12.4.1.1 Event declarations

You should add Event declarations if you compile the custom view into a library. If the event routine has parameters these must also be declared.

```
#Event: ExampleEvent (Value As Int)
```

### 12.4.1.2 Designer properties declarations

```
#DesignerProperty: Key: BooleanExample, DisplayName: Boolean Example, FieldType: Boolean, DefaultValue: True, Description: Example of a boolean property.
```

You can add custom properties for the Designer.

More details in the chapter [Custum view in the Designer](#).

### 12.4.1.3 Global variable declarations

In this routine you should declare all global variables used in the class.

The variables below are mandatory.

```
Sub Class_Globals
  Private EventName As String 'ignore
  Private CallBack As Object 'ignore
  Private mBase As Panel
End Sub
```

EventName    Event name used for the events in the code, same as for standard views.  
 Call Back    Module where the class is declared, used for event calls.  
 mBase        Main panel of the custom view.

You can, if you want, change the name of the base panel.

What is this for 'ignore' ?

It avoids a warning of the compiler that these variables are unused.

Variables only used in the class should be declared as `Private`.

If you want to have access to variables from outside you must declare them as `Public`.

### 12.4.1.4 Initialization routine

The initialize routine initiates a new instance of the custom view.

```
Public Sub Initialize (vCallback As Object, vEventName As String)
  EventName = vEventName
  Call Back = vCall back
End Sub
```

These two variables will be used to call event routines in the module where the custom view is initialized.

Example:

```
' if a callback routine exists in the calling module we call it
If SubExists(Callback, EventName & "_ValuesChanged") Then
  Call Sub3(Callback, EventName & "_ValuesChanged", cLimitLeft, cLimitRight)
End If
```

### 12.4.1.5 Designer support routine

This routine assures the support for the Designer, it is called directly after the Initialize routine of the custom view class.

You should not modify its signature.

```
Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
    mBase = Base
End Sub
```

**Base** Is the base panel defined in the Designer, it holds the Left, Top, Width and Height properties of the custom view. The Base panel can be used or not.

**Lbl** Is a Label which holds all the text properties defined in the Designer. This Label can be used or not.

**Props** Is a Map holding additional properties.

### 12.4.1.6 Routine to get the base Panel

You can use this routine if you want to access the base panel from outsides.

```
Public Sub GetBase As Panel
    Return mBase
End Sub
```

In the calling module:

```
Private pnlClass As Panel
pnlClass = clsTest.GetBase
```

## 12.4.2 Adding a custom view by code

To offer the possibility to add the custom view by code you must add a routine in the class which adds the custom view onto a parent view which can be either an Activity or a Panel.

Example:

```
Public Sub AddToParent (Parent As Activity, Left As Int, Top As Int, Width As Int,
Height As Int)
    mBase.Initialize("mBase")
    Parent.AddView(mBase, Left, Top, Width, Height)
End Sub
```

Parent is the parent view which can be an Activity or a Panel.

Left is the Left property.

Top is the Top property.

Width is the Width property.

Height is the Height property.

You can add other parameters or properties to the routine if necessary.

And in the calling module:

```
Private clsTest2 As clsCustomView

clsTest2.Initialize(Me, "clsTest2")
clsTest2.AddToParent(Activity, 10dip, 10dip, 200dip, 50dip)
```

### 12.4.3 Add properties

Property routines can be added which work like any property of the standard views.

These properties can be read and or set.

To read a property you must add a routine beginning with get, **lower case** and the property name.

Examples:

Get the *Left* Property.

```
'gets the Left value  
Public Sub getLeft As Int  
    Return I tbPanel Back. Left  
End Sub
```

Get the custom *Max* property.

```
'gets the Max value  
Public Sub getMax As Int  
    Return MaxVal  
End Sub
```

To set a property you must add a routine beginning with set, **lower case** and the property name.

Examples:

Set the *Left* Property.

```
'sets the Left value  
Public Sub setLeft(Left As Int)  
    I tbPanel Back. Left = Left  
End Sub
```

Set the custom *Max* property.

```
'sets the Max value  
Public Sub setMax(cMax As Int)  
    MaxVal = cMax  
    Scale = (x1 - x0) / MaxVal  
End Sub
```

If you define only a get routine the property is read only.

If you define only a set routine the property is write only.

If you define both a set and a get routine, the property is write and read.

## 12.4.4 Custom view in the Designer

You can add code to make custom properties visible in the Designer.

The images below are from the TestClass project in the SourceCode\Classes folder.

On the top of the code you must include declaration lines. The default layout of a custom view class includes these example declarations:

```
#DesignerProperty: Key: BooleanExample, DisplayName: Boolean Example, FieldType: Boolean, DefaultValue: True, Description: Example of a boolean property.
#DesignerProperty: Key: IntExample, DisplayName: Int Example, FieldType: Int, DefaultValue: 10, MinRange: 0, MaxRange: 100, Description: Note that MinRange and MaxRange are optional.
#DesignerProperty: Key: StringWithListExample, DisplayName: String With List, FieldType: String, DefaultValue: Sunday, List: Sunday|Monday|Tuesday|Wednesday|Thursday|Friday|Saturday
#DesignerProperty: Key: StringExample, DisplayName: String Example, FieldType: String, DefaultValue: Text
#DesignerProperty: Key: ColorExample, DisplayName: Color Example, FieldType: Color, DefaultValue: 0xFFCFDCDC, Description: You can use the built-in color picker to find the color values.
```

Each property declaration is made of several fields, the following fields are required:

<b>Key</b>	Is the key value for the Map. This will be used to get the value from the Props map.
<b>DisplayName</b>	Is the name displayed in the Designer property grid.
<b>FieldType</b>	Is the type of the field. Possible values: String, Int, Double, Boolean or Color.
<b>DefaultValue</b>	Is the default value which is set in the Designer.

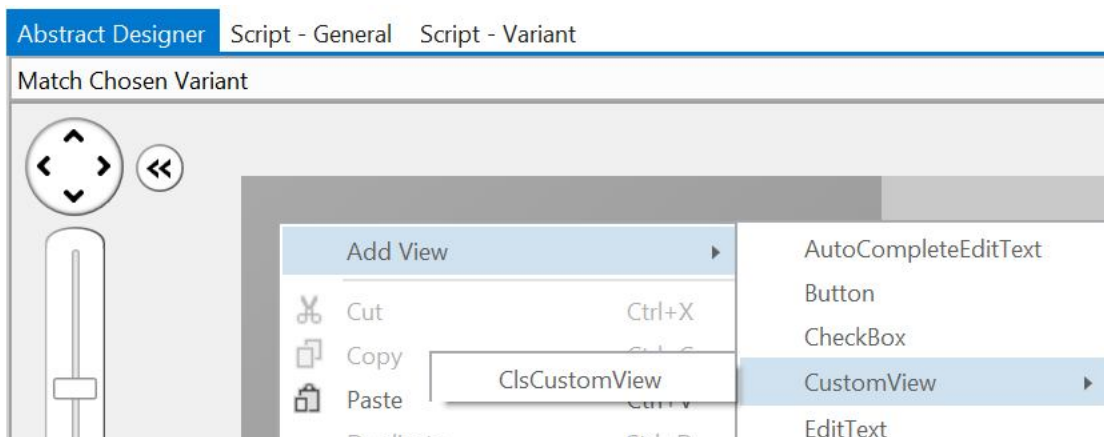
Optional fields:

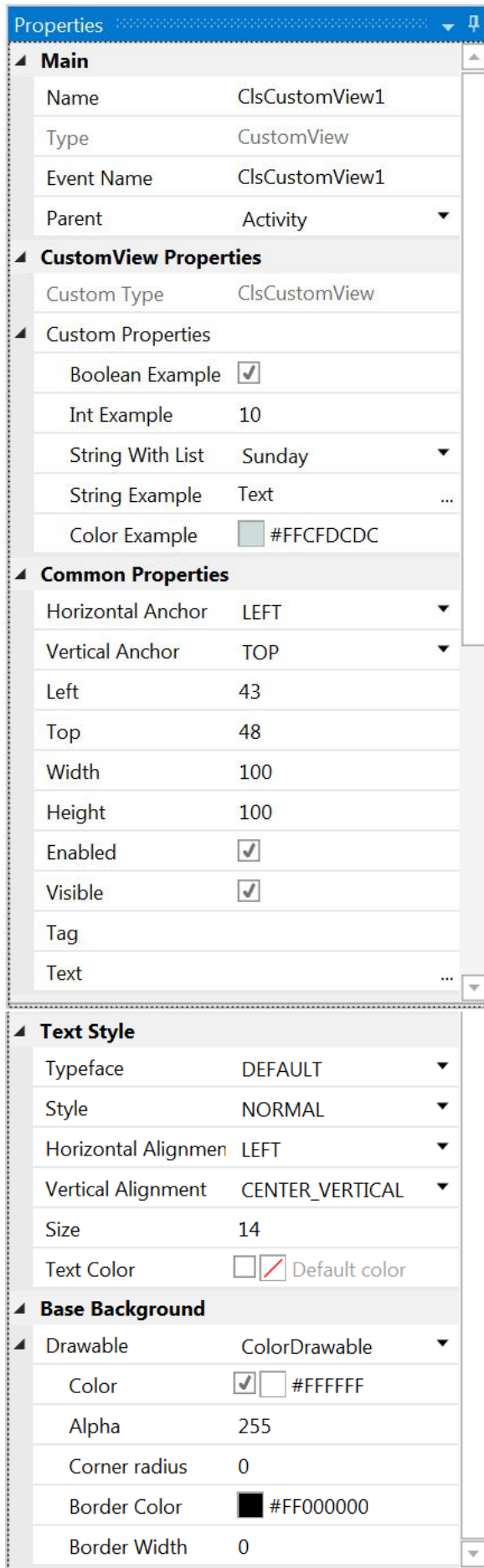
<b>Description</b>	Is the explanation text displayed in the Designer.
<b>MinRange / MaxRange</b>	Minimum and maximum numeric values allowed.
<b>List</b>	A pipe ( ) separated list of items from which the developer can choose (should be used with string fields).

In the Designer you can add a CustomView like this:

Right click in the screen area, select Add View and select CustomView.

Select the custom from the list of available custom views *ClsCustomView* in the example.



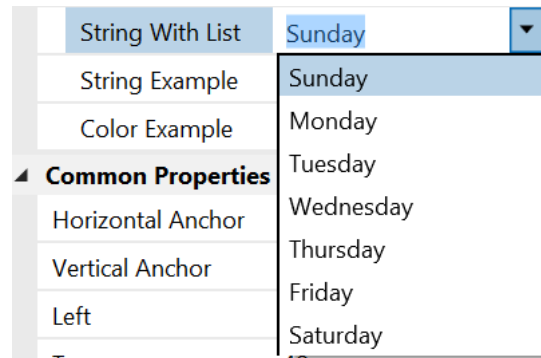


In the Properties window you find all the properties for the selected custom view.

**Custom properties:**

Here we see the five custom properties declared on top of the Class code.

Example with the String With List property.



**Common Properties:**

The common properties like any view.

**Text Style:**

The properties are set to the Lbl Label of the class.

**Base Background:**

Background of the base panel mBase.



To recuperate the custom properties you must use the Props Map in the **DesignerCreateView** routine.

Variable declaration :

```
Private BooleanTest As Boolean
Private IntTest As Int
Private Day As String
Private StringTest As String
Private ColorTest As Int
```

And the DesignerCreateView routine:

```
Public Sub DesignerCreateView (Base As Panel , Lbl As Label , Props As Map)
    mBase = Base

    BooleanTest = Props.Get("BooleanExample")
    IntTest = Props.Get("IntExample")
    Day = Props.Get("StringWithListExample")
    StringTest = Props.Get("StringExample")
    ColorTest = Props.Get("ColorExample")
End Sub
```

You can get the text properties from the Lbl Label like:

```
TextSize = Lbl.TextSize
```

## 12.5 First example LimitBar

The first example is a LimitBar.

The source code is in the Classes\ClsLimitBar folder.  
It supports adding it in the Designer or in the code.

The LimitBar looks like this:



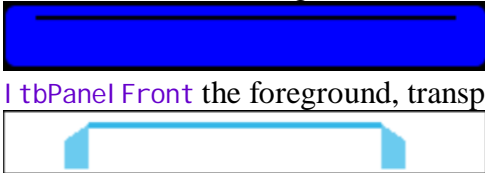
Two cursors allow to define two limits.

In the demo program we add two labels one on each side to display the two limit values.



We'll use two panels:

- `I tbPanel Back` the background with the background color and the dark 'background' line.
- `I tbPanel Front` the foreground, transparent with the 'foreground' line and the two cursors.



and two canvases:

- `cvsPanel Back` to draw the background line onto `I tbPanel Back`.
- `cvsPanel Front` to draw the foreground line and the cursors onto `I tbPanel Front`.

First we declare several objects and variables in `Sub Class_Global s` :

### Sub Class\_Global s

```

Private Callback As Object           ' calling module
Private I tbPanel Back As Panel      ' the background panel
Private I tbPanel Front As Panel     ' the foreground panel
Private cvsPanel Back As Canvas      ' the background canvas
Private cvsPanel Front As Canvas     ' the foreground canvas
Private rectPanel Front As Rect      ' a rectangle for the foreground canvas

Private BackLineColor As Int        ' color for the background line
BackLineColor = Colors.Black
Private FrontLineColor As Int       ' color for the foreground line
FrontLineColor = Colors.RGB(51, 181, 229)
Private LimitSliderColor As Int     ' color for the cursors
LimitSliderColor = Colors.ARGB(180, 51, 181, 229)
Private Margin = 15dip As Float     ' left and right margins for the line
Private x0, y0, x1 As Int           ' values used internally
Private MaxVal = 100 As Int         ' value of the Max property
Private Scale As Double             ' scale between position value and pixels
Private cLimitLeft As Int           ' value of the left limit
Private cLimitRight As Int          ' value of the right limit
Private PositionPixels(2) As Int    ' left and right positions in pixels
                                     ' 0 = left  1 = right
Private PosIndex As Int             ' current index of the position
Private EventName As String         ' event name
Private Paths(2) As Path            ' two paths for the cursor shape
End Sub

```

Then we need the routine to initialize the LimitBar, the code is self explanatory.

```
' Initializes the object.
' CallbackModule = name of the calling module
' cEventName = event name
' Example if added in the code:
' <Code>|tbTest.Initialize(Me, "|tbTest")' </Code>
Public Sub Initialize(CallbackModule As Object, cEventName As String)
    Callback = CallbackModule
    EventName = cEventName
End Sub
```

Then we have the DesignerCreateView routine.

```
Public Sub DesignerCreateView(Base As Panel, Lbl As Label, Props As Map)
    ' we use the Base panel as the background panel
    |tbPanelBack = Base

    MaxVal = Props.Get("Max")
    cLimitLeft = Props.Get("LimitLeft")
    cLimitRight = Props.Get("LimitRight")
    cBackColor = Props.Get("BackColor")
    cFrontLineColor = Props.Get("FrontLineColor")

    Init
End Sub
```

We use the Base Panel with the name |tbPanelBack, and get the custom properties from the Props Map object.

As the LimitBar custom view can also be added in the code we initialize the rest in the Init routine.

The AddToParent routine.

```
' Adds the LimitBar to the Parent object
' Parent = parent view, the Activity or a Panel
' Left, Right, Width, Height = position and dimensions properties of the LimitBar
' Color = background color of the LimitBar
' Radius = corner radius of the LimitBar
Public Sub AddToParent(Parent As Activity, Left As Int, Top As Int, Width As Int,
    Height As Int, Color As Int, Radius As Int)

    Height = Max(Height, 30dip) ' limits the height to min
    30dip
    Radius = Min(Radius, Height / 2) ' limits the max radius to half the height

    ' initialize the background panel |tbPanelBack and add it onto the parent view
    |tbPanelBack.Initialize("")
    Parent.AddView(|tbPanelBack, Left, Top, Width, Height)

    ' initialize and set the ColorDrawable for the background panel
    Dim cdw As ColorDrawable
    cdw.Initialize(Color, Radius)
    |tbPanelBack.Background = cdw

    Init
End Sub
```

We initialize |tbPanelBack, add it onto the parent view and set its background and call Init.

The next routine is the `Init` routine where we initialize the rest.

This routine is called either from the `DesignerCreateView` when the custom view is added in the Designer or from the `AddToParent` routine when the custom view is added in the code.

#### Private Sub Init

```
' initialize the foreground panel and add it onto the background panel
I tbPanel Front. Initialize("I tbPanel Front")
I tbPanel Back. AddView(I tbPanel Front, 0, 0, I tbPanel Back. Width, I tbPanel Back. Height)
' initialize the foreground panel rectangle used to erase I tbPanel Front
rectPanel Front. Initialize(0, 0, I tbPanel Front. Width, I tbPanel Front. Height)
I tbPanel Front. BringToFront

' set local variables
x0 = Margin
x1 = I tbPanel Back. Width - Margin
y0 = 6dip
setMax(100)

' initialize the background canvas and draw the background line
cvsPanel Back. Initialize(I tbPanel Back)
cvsPanel Back. DrawLine(x0, y0, x1, y0, cBackColor, 2)
I tbPanel Back. Invalidate

' initialize the foreground canvas
cvsPanel Front. Initialize(I tbPanel Front)

' set the left cursor parameters and draw it
PosIndex = 0
PositionPixels(0) = x0
DrawPos(x0)

' set the right cursor parameters and draw it
PosIndex = 1
PositionPixels(1) = x1
DrawPos(x1)
```

#### End Sub

The code is self explanatory.

The drawing routine for the cursors and the foreground line:

We:

- Erase the whole panel by drawing a transparent rectangle.
- Set the current position to the active cursor `PositionPixels(PosX) = x`  
`PosX` is the index of the current cursor.
- Define both cursors according to the current position.  
The cursor shapes are defined with two Paths.
- Draw the cursors.
- Draw the foreground line.

**Private Sub DrawPos(x As Int)**

```
' draw a transparent rectangle to erase the foreground panel
cvsPanel Front.DrawRect(rectPanel Front, Colors.Transparent, True, 1)

' set the current cursor position
PositionPixels(PosX) = x

' define the left cursor path according to its current position
Paths(0).Initialize(PositionPixels(0), y0)
Paths(0).LineTo(PositionPixels(0), y0 + 22di p)
Paths(0).LineTo(PositionPixels(0) - 12di p, y0 + 22di p)
Paths(0).LineTo(PositionPixels(0) - 12di p, y0 + 8di p)
Paths(0).LineTo(PositionPixels(0), y0)

' define the right cursor path according to its current position
Paths(1).Initialize(PositionPixels(1), y0)
Paths(1).LineTo(PositionPixels(1), y0 + 22di p)
Paths(1).LineTo(PositionPixels(1) + 12di p, y0 + 22di p)
Paths(1).LineTo(PositionPixels(1) + 12di p, y0 + 8di p)
Paths(1).LineTo(PositionPixels(1), y0)

' draw the two cursors and the foreground line
cvsPanel Front.DrawPath(Paths(0), cFrontLineColor, True, 1)
cvsPanel Front.DrawPath(Paths(1), cFrontLineColor, True, 1)
cvsPanel Front.DrawLine(PositionPixels(0), y0, PositionPixels(1), y0, cFrontLineColor,
3di p)

' if Mode = 1 draw the current cursor with the FrontLineColor (highlighted)
cvsPanel Front.DrawPath(Paths(PosX), cFrontLineColor, True, 1)
ItbPanel Front.Invalidate ' update the foreground panel
End Sub
```

To detect cursor moves we use the touch event of the foreground panel:

```

Private Sub ItbPanelFront_Touch (Action As Int, X As Float, Y As Float)
    ' check if the cursor is outside the limits
    X = Max(x0, X)
    X = Min(x1, X)

    ' select the Action type
    Select Action
    Case 0 ' DOWN
        If X < Abs(PositionPixels(0) + PositionPixels(1)) / 2 Then
            ' if X is closer to the left cursor we choose it
            PosIndex = 0
        Else
            ' otherwise we choose the right cursor
            PosIndex = 1
        End If
        DrawPos(X) ' we draw the current cursor highlighted
    Case 2 ' MOVE
        DrawPos(X) ' we draw the current cursor highlighted
    Case 1 ' UP
        DrawPos(X) ' we draw the current cursor not highlighted
    End Select

    ' we calculate the current limit value from the X position in pixels
    If PosIndex = 0 Then
        cLimitLeft = Floor((X - x0) / Scale + .5)
    Else
        cLimitRight = Floor((X - x0) / Scale + .5)
    End If

    ' when Action is UP check if cLimitLeft > cLimitRight
    ' if yes we invert the limit values and redraw the cursors
    If Action = 1 And cLimitLeft > cLimitRight Then
        Private val As Int
        val = cLimitLeft
        cLimitLeft = cLimitRight
        cLimitRight = val
        PosIndex = 0
        X = cLimitLeft * Scale + x0
        DrawPos(X)
        PosIndex = 1
        X = cLimitRight * Scale + x0
        DrawPos(X)
    End If

    ' if a callback routine exists in the calling module we call it
    If SubExists(Callback, EventName & "_ValuesChanged") Then
        CallSub3(Callback, EventName & "_ValuesChanged", cLimitLeft, cLimitRight)
    End If
End Sub

```

Finally we add a few properties:

To add properties see more details in [Chapter Add properties to a class](#).

The Max property:

```
'gets or sets the max value
Sub setMax(cMax As Int)
    MaxVal = cMax
    Scale = (x1 - x0) / MaxVal
End Sub
```

```
Sub getMax As Int
    Return MaxVal
End Sub
```

The LimitLeft property:

```
'gets or sets the left limit
Sub setLimitLeft(Pos As Int)
    ' if Pos is lower than 0 set cLimitLeft to 0
    cLimitLeft = Max(0, Pos)
    PosIndex = 0
    DrawPos(x0 + cLimitLeft * Scale, 0)
End Sub
```

```
Sub getLimitLeft As Int
    Return cLimitLeft
End Sub
```

The LimitRight property:

```
'gets or sets the right limit
Sub setLimitRight(Pos As Int)
    ' if Pos is higher than MaxVal set cLimitRight to MaxVal
    cLimitRight = Min(MaxVal, Pos)
    PosIndex = 1
    DrawPos(x0 + cLimitRight * Scale, 0)
End Sub
```

```
Sub getLimitRight As Int
    Return cLimitRight
End Sub
```

The Visible property:

```
'gets or sets the Visible property
Sub setVisible(IsVisible As Boolean)
    I tbPanel Back. Visible = IsVisible
End Sub
```

```
Sub getVisible As Boolean
    Return I tbPanel Back. Visible
End Sub
```

I didn't add more properties to keep the example code 'simple'.  
But other properties could easily be added.

## 12.6 Compile a class into a Library

In B4A you can compile your project, or part of it to a regular library.

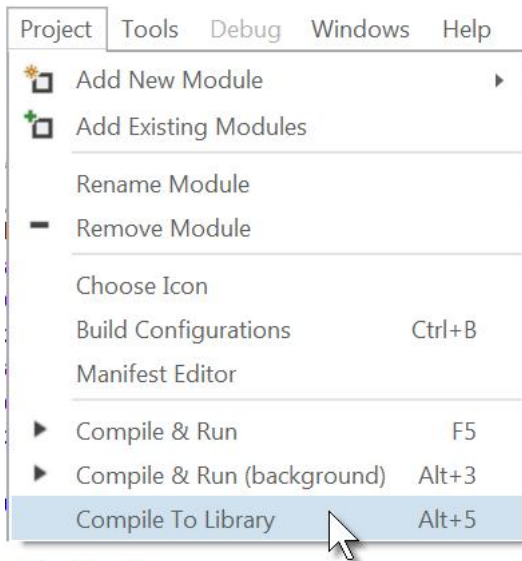
Why should I compile a library?

- Break large projects into several smaller (more maintainable) projects.
- Build reusable components and use them from any number of projects.
- Share components with other developers without sharing the source code.
- Create different versions of your application (free, pro...) by referencing the same "core" library.

The output of library compilation are two files: a jar file with the compiled code and a xml file that includes the metadata that is required by the IDE.

These two files are automatically saved in the additional libraries folders.

Compiling to a library is quite simple. Under Project menu there is the compile option - "Compile To Library (Alt + 5)". When you choose this option all the modules **except** of the main activity are compiled into a library.



You can exclude other modules as well with the ExcludeFromLibrary attribute (see this tutorial for more information about [attributes](#)).

The main activity and the other excluded modules can be used to test the library.

You can reference the library from other projects and access the same functionality as in the original project.



## Library specific attributes

The following attributes are specific for library compilation:

Project attributes (placed in the main activity):

*LibraryVersion* - A number that represents the library version. This number will appear next to the library name in the libraries list.

*LibraryAuthor* - The library author. This value is added to the library xml file.

*LibraryName* (B4A v2.70) - The compiled library name. Sets the library name instead of showing the save dialog.

All modules:

*ExcludeFromLibrary* - Whether to exclude this module during library compilation. Values: True or False. Note that the Main activity is always excluded.

Classes:

*Event* - Adds an event to the list of events. This attribute can be used multiple times. Note that the events list only affects the IDE events autocompletion feature.

Example:

In the Main module

```
#Region Project Attributes
  #ApplicationLabel: LimitBarDemo2
  #VersionCode: 1
  #VersionName:
  'SupportedOrientations possible values: unspecified, landscape or portrait.
  #SupportedOrientations: unspecified
  #CanInstallToExternalStorage: False

  #LibraryVersion: 1.0
  #LibraryName: LimitBar
  #LibraryAuthor: Klaus Christl
#End Region
```

In a code module:

```
#ExcludeFromLibrary
```

## Notes

- You should right click on the libraries list and choose Refresh after a library update.
- CallSub / CallSubDelayed - The first parameter for these keywords is a reference to the target module. When working with modules that reside in a library you should pass the module reference and not the module name as string (this is the better way to reference all modules in all cases).
- Code obfuscation - Libraries can be obfuscated during library compilation. Strings will not be obfuscated in this mode.
- Services that host home screen widgets cannot be compiled into a library.

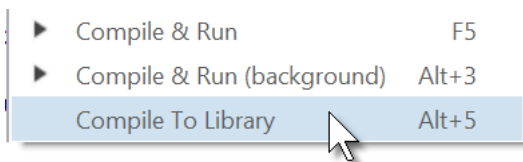
## 12.6.1 Example with the LimitBar class example

We take the LimtBar demo program to show the principle.  
Source code in the Classes\ClsLimtBar folder.

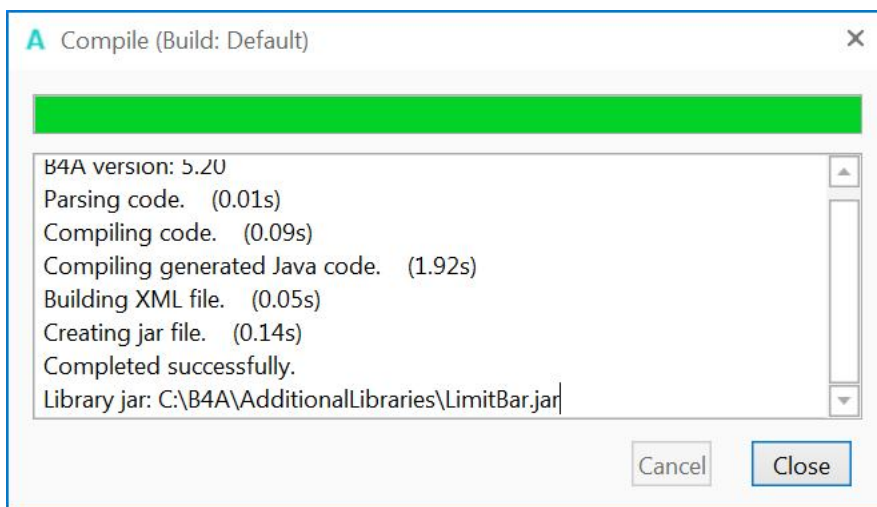
In the Project Attributes Region in the Main module we add following new attributes:

```
#Region Project Attributes
#ApplicationLabel: LimitBarDemo
#VersionCode: 1
#VersionName:
'SupportedOrientations possible values: unspecified, landscape or portrait.
#SupportedOrientations: unspecified
#CanInstallToExternalStorage: False

#LibraryVersion: 1.0
#LibraryName: LimitBar
#LibraryAuthor: Klaus Christl
#End Region
```



Then click on to compile the library.



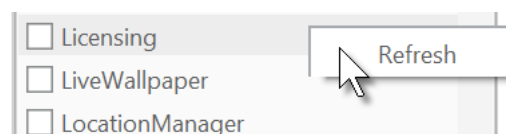
A screen similar to this one is displayed showing that the library has been successfully compiled.

Two files are automatically added in the additional library folder.

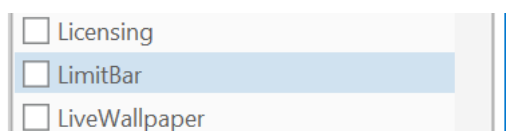
In our example:

LimitBar.xml

LimitBar.jar



Right click in the Libs tab and click on Refresh.



The new library is now available.

Then we add these two lines on top of the class code:

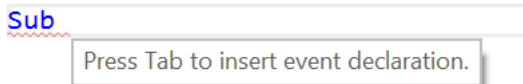
```
#Event: ValuesChanged(LimitLeft As Int, LimitRight As Int)
#RaisesSynchronousEvents: ValuesChanged
```

**#Event:** is used for the IDE events autocompletion feature, if the event routine has parameters they should be declared. You need one for each event.

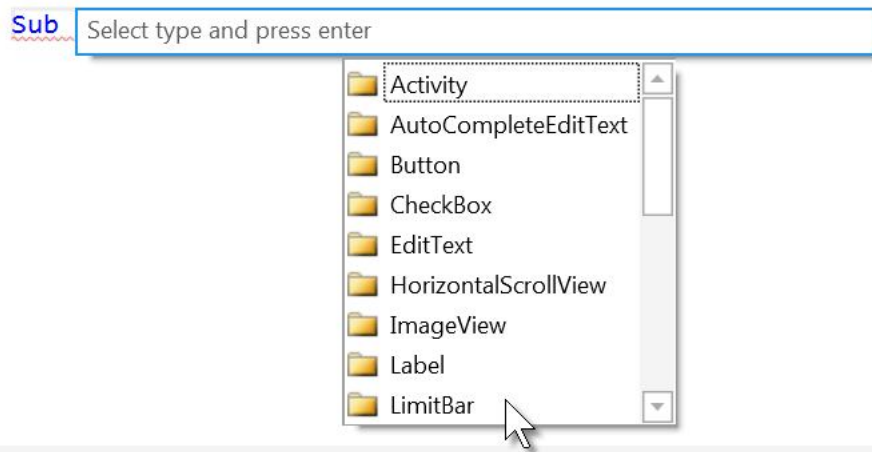
Once the class is compiled into a library you can use the events autocompletion feature in the IDE.

Example in the LblLimitBar project.

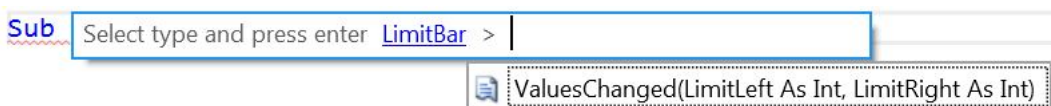
Write Sub, a blank character.



Press Tab, a list of views is displayed, you find the LimitBar view at the bottom.



A list of the available event is displayed, select one.



The routine is written,

```
Sub EventName_ValuesChanged(LimitLeft As Int, LimitRight As Int)
End Sub
```

Change the view name and the routine skeleton is ready.

```
Sub ltbTest(LimitLeft As Int, LimitRight As Int)
End Sub
```

**#RaisesSynchronousEvents:** Is used for the Rapid Debugger. You need one for each event.

## 12.6.2 Using the library in a program

To use the library we copy the LimitBarDemo to another folder and load it.

The resulting source code for this example is in the Classes\LblLimitBar1 folder.

We remove the LimitBar module.

We remove the three Project Attributes.

```
#LibraryVersion: 1.0
#LibraryName: LimitBar
#LibraryAuthor: Klaus Christl
```

We check the LimitBar library in the Libs tab  LimitBar (version: 1.00)

Run the program



that's it !

Open the Designer we find the lbtTest custom view as defined previously.

### Adding a LimitBar in the code.

In the example below we add one LimitBar in the Designer and another one in the code.

The source code is in the Classes\LblLimitBar2 folder.

We need to Dim the LimitBars

```
Private ltbTest, ltbTest1 As LimitBar
```

We initialize ltbTest1 in the code and add it to the parent view (Activity in our example).

```
'adds a second LimitBar in the code
ltbTest1.Initialize(Me, "ltbTest1")
ltbTest1.FrontLineColor = Colors.Blue
ltbTest1.AddToParent(Activity, 40dip, 100dip, 240dip, 30dip, Colors.Red, 5dip)
```

Initialize : Me = CallModule, current module and "ltbTest2" = EventName

AddToParent parameters : Parent view, Left, Top, Width, Height, BackgroundColor, CornerRadius.

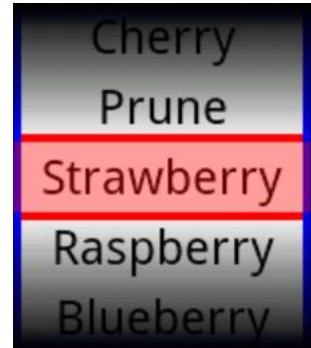
## 12.7 Second example Wheel selection

### 12.7.1 Simple example

We will develop a wheel selection class.

The source code is in the Classes/CIsWheelSimple1 folder.

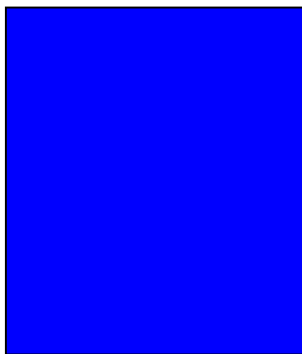
The height of the views is automatically adjusted according to the text size. The width of the views is automatically adjusted according to the longest text.



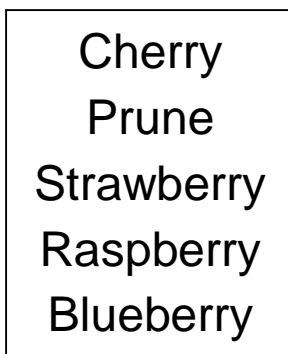
To display the wheel we use:

- 1 ScrollView to display the data and allow scrolling.
  - The entries in the ScrollView are Labels added in the code.
- 4 Panels
  - the background panel holding all views
  - a top and a bottom panel for the shadows
  - a center panel as the 'display window'

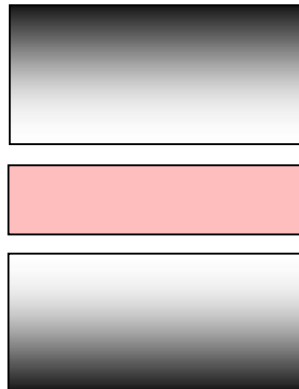
Background Panel



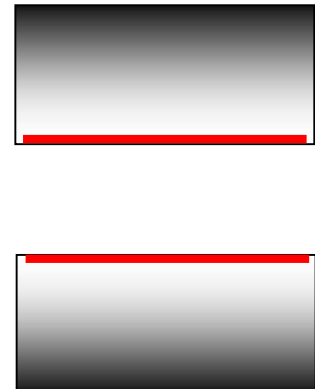
ScrollView



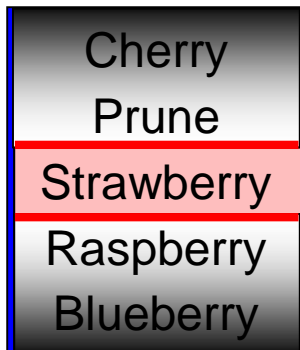
Top, center bottom Panels



Lines on top and bottom Panels



ScrollView on background Panel



All Views superimposed on background Panel

The code:

In `Class_Globals` we define different variables.

#### Sub `Class_Globals`

```

Private pnlBackground, pnlTop, pnlMiddle, pnlBottom As Panel
Private Callback As Object      ' calling module
Private scv As ScrollView      ' ScrollView for the data
Private ColWidth As Int        ' width of the column (ScrollView)
Private FontSize As Float      ' font size for the text in the ScrollView
Private lblHeight As Int       ' height of the ScrollView

Private WheelContent As List    ' content of the

' variables used to calculate the dimensions of pnlBackground
Private Left, Top, Width, Height As Float

Private LineWidth = 4dip As Int      ' width of the lines

' colors
Private colBackground = Colors.Blue As Int      ' background
Private colShadow = Colors.Black As Int        ' shadow of top and bottom panels
Private colWindowLine = Colors.Red As Int      ' lines of center window
Private colWindow = Colors.ARGB(96, 255, 0, 0) As Int ' center window
End Sub

```

Initialization routine:

We define local variables to use them in other routines.

```

' Initializes the Wheel.
' CallbackModule = calling module
' Parent          = parent activity or panel
' cWheelContent = List object with the content on the wheel
' cFontSize      = font size of the text in the wheel
Public Sub Initialize(CallbackModule As Object, Parent As Object, _
    cWheelContent As List, cFontSize As Float)

    Callback = CallbackModule
    FontSize = cFontSize
    WheelContent = cWheelContent

```

We calculate the height of one Label in the ScrollView. We define a Canvas for the Parent object to measure the height of the text in pixels for the given font size and add a top and bottom margin depending on the font size.

```

' calculate the text height in pixels according to the text size
Dim cvs As Canvas
cvs.Initialize(Parent)
lblHeight = cvs.MeasureStringHeight("Ag", Typeface.DEFAULT, FontSize) + _
DipToCurrent(FontSize / 2)

```

With `cvs.MeasureStringHeight` we measure the height of the text according to the font size. Then we add a small amount for margins depending also on the font size, `FontSize / 2` is a good value. But as the font size is independent of the screen density we need to transform this value into a `dip` (density independent pixels) value with the `DipToCurrent` function.

We admit the height of the wheel (the ScrollView) and the background Panel equal to 5 Label heights. We have 2 label heights above the center window and 2 label heights below it.

```
' we admit the total height to 5 times the label height
Height = 5 * lblHeight
```

We calculate the max text length, add a margin and add two line widths for the border.

```
' calculate the width of the longest text in the ScrollView
' according to the text size
ColWidth = 0
For j = 0 To WheelContent.Size - 1
    ColWidth = Max(ColWidth, cvs.MeasureStringWidth(WheelContent.Get(j), _
        Typeface.DEFAULT, FontSize))
Next
ColWidth = ColWidth + 20dip      ' add a margin
Width = ColWidth + 2 * LineWidth ' add two line widths for the total width
```

We set `ColWidth = 0` (the ScrollView width) and with `cvs.MeasureStringWidth` we measure the text length of each entry and memorize the bigger value in `ColWidth`.

Then we add `20dip` for the margins.

The whole width of the background panel is the scrollview width plus two line widths.

We initialize the background panel, add it onto the parent object and set its background color.

```
' initialize pnlBackground and add it onto the parent object
' and set its color
pnlBackground.Initialize("pnlBackground")
If Parent Is Activity Then
    Private act As Activity
    act = Parent
    Left = (act.Width - Width) / 2      ' center pnlBackground in the parent object
    Top = (act.Height - Height) / 2
    act.AddView(pnlBackground, Left, Top, Width, Height)
Else If IsPanel(Parent) Then
    Dim pnlp As Panel
    pnlp = Parent
    Left = (pnlp.Width - Width) / 2    ' center pnlBackground in the parent object
    Top = (pnlp.Height - Height) / 2
    pnlp.AddView(pnlBackground, Left, Top, Width, Height)
Else
    Log("Parent must be an activity or a panel.")
    Return False
End If
pnlBackground.Color = colBackground
```

We initialize `pnlBackground` calculate its `Left` and `Top` properties to centre it on the screen or on the parent panel. If the parent view is not an activity nor a panel an error message is displayed.

We initialize the ScrollView and add it onto the background panel.

```
' initialize the ScrollView and add it onto the parent object
scv.Initialize2(lblHeight * (WheelContent.Size + 4), "scv")
pnlBackground.AddView(scv, lineWidth, 0, ColWidth, Height)
```

The internal panel height is equal to the size of the WheelContent plus 4 label heights for the two empty labels on top and the two on the bottom.

We fill the ScrollView.

```
' fill the ScrollView
For j = 0 To WheelContent.Size + 5
    Private lbl As Label
    lbl.Initialize("")
    scv.Panel.AddView(lbl, 0, j * lblHeight, ColWidth, lblHeight)
    lbl.Gravity = Gravity.CENTER_HORIZONTAL + Gravity.CENTER_VERTICAL
    lbl.Color = Colors.White
    lbl.TextColor = Colors.Black
    lbl.TextSize = FontSize
    If j >= 2 AND j <= WheelContent.Size + 2 - 1 Then
        lbl.Text = WheelContent.Get(j - 2)
    Else
        lbl.Text = ""
    End If
Next
```

We need to dim the labels in the loop to have an independent instance for each.

No need to add an event name in the Initialize method, we don't use any Label event.

Add the Label onto the ScrollView.Panel.

Set Gravity of the Labels to centre the text vertically and horizontally.

Set the Label background and text color.

Fill the ScrollView, the first two Labels are empty to ensure that the first entry is shown in the middle window. And we add two empty Labels at the end.

We initialize the top panel and set its background colors.

```
' initialize the top panel and set its background colors
' and set its color
pnlTop.Initialize("")
pnlBackground.AddView(pnlTop, 0, 0, Width, 2 * lblHeight)
Private gdw As GradientDrawable
Private Dim cols(2) As Int
cols(0) = colShadow
cols(1) = Colors.Transparent
gdw.Initialize("TOP_BOTTOM", cols)
gdw.CornerRadius = 0
pnlTop.Background = gdw
Private cvs1 As Canvas
cvs1.Initialize(pnlTop) ' initialize a canvas and draw the line on the bottom
cvs1.DrawLine(lineWidth, pnlTop.Height - lineWidth / 2, Width - lineWidth, _
pnlTop.Height - lineWidth / 2, colWindowLine, lineWidth)
```

The background of the top panel is a GradientDrawable with two colors the `colShadow` color on top and transparent on the bottom.

We dim a GradientDrawable object and an array of Int for the two colors, initialize the GradientDrawable, set the color orientation "`TOP_BOTTOM`" and the colors, set the radius and set the GradientDrawable as the panel background.

Then we initialize a Canvas and draw a line on the bottom border.

We initialize the centre panel and set its background color.

```
' initialize the middle panel and set its background color
pnlMiddle.Initialize("")
pnlBackground.AddView(pnlMiddle, 0, 2 * lblHeight, Width, lblHeight)
pnlMiddle.Color = colWindow
```



We initialize the bottom panel and set its background colors.

```
' initialize the bottom panel and set its background
pnlBottom.Initialize("")
pnlBackground.AddView(pnlBottom, 0, (2 + 1) * lblHeight, Width, 2 * lblHeight)
Private gdw As GradientDrawable
Private cols(2) As Int
cols(0) = colShadow
cols(1) = Colors.Transparent
gdw.Initialize("BOTTOM_TOP", cols)
gdw.CornerRadius = 0
pnlBottom.Background = gdw
Private cvs2 As Canvas
cvs2.Initialize(pnlBottom) ' initialize a canvas and draw the line on the top
cvs2.DrawLine(lineWidth, lineWidth / 2, Width - lineWidth, lineWidth / 2, _
colWindowLine, lineWidth)
```

The principle is the same as for the top panel.

The wheel at this state is not usable it has two major drawbacks.

- When we scroll the wheel the selected entry is not shown centred in the middle window.
- We need a method to return the selected value.

## 12.7.2 Show the selected entry centred in the middle window.

The source code is in the Classes\ClsWheelSimple2 folder.

To achieve this we calculate the scrolling speed and when the user releases the scrolling we calculate an estimated time till the scroll end, set this time to a Timer and the Timer calls a routine to finish the scrolling to show the selected entry centred in the middle window.

We add a Touch event for the ScrollView with the Reflection library and a Timer.

In the Touch event we calculate the scrolling speed and when the user releases the scrolling we calculate an estimated time to finish the scrolling and set this time to the Timer

Code:

In the `Class_Global`s routine we add the variables below:

```
Private TimerWheel As Timer

Private ScrollPos As Int
Private ScrollPosMax As Int
Private y0 As Int
Private t0 As Long
Private speed As Double
```

Then, in the `Sub Initialize` routine just before the ScrollView filling we add the code below.

```
' add the Touch event to the ScrollView
Private objWheel As Reflector
objWheel.Target = scvWheel
objWheel.SetOnTouchListener("scvWheel_Touch")
objWheel.RunMethod2("setVerticalScrollBarEnabled", False, "java.lang.booolean")

' fill the ScrollView
```

We define a Reflector object, set its target to the ScrollView, add the ScrollView Touch event and remove the vertical scrollbar.

And at the end of the `Sub Initialize` routine we add:

```
TimerWheel.Initialize("TimerWheel", 200)
```

End Sub

Then we add the `Sub scvWheel_Touch` routine :

```
Private Sub scvWheel_Touch(ViewTag As Object, Action As Int, X As Float, _
    Y As Float, MotionEvent As Object) As Boolean
    Private dt As Long
    Private tt As Int

    Select Action
    Case 0 ' ACTION_DOWN
        t0 = DateTime.Now
        y0 = Y
    Case 1 ' ACTION_UP
        tt = Max(10, -Logarithm(1 / speed, cE) * 110)
        TimerWheel.Interval = tt
        TimerWheel.Enabled = True
    Case 2 ' ACTION_MOVE
        dt = (DateTime.Now - t0)
        speed = Abs((Y - y0) / dt * 250)
        t0 = DateTime.Now
        y0 = Y
    End Select

    Return False
End Sub
```

In `ACTION_DOWN` we set the variables `t0` to the current time and `y0` to the current y position.

In `ACTION_MOVE` we calculate the time difference `dt` between the current time and the previous move. Then we calculate the moving `speed`, the value of 250 was determined by trials.

And we set the variables `t0` to the current time and `y0` to the current y position.

In `ACTION_UP` we calculate the estimated time `tt` till scroll end. The equation is based on a logarithmic decay and the value of 110 was determined by trials.

Then we set this time to the `Timer.Interval` and enable the `Timer`.

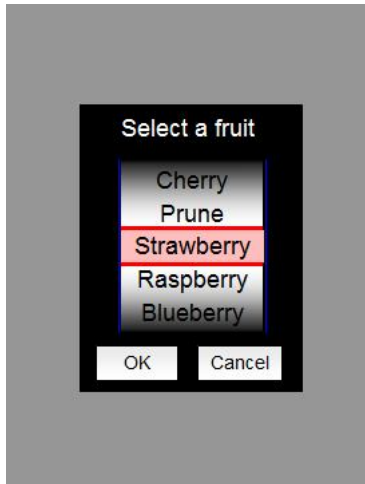
The `TimerWheel_Tick` routine:

```
Private Sub TimerWheel_Tick
    TimerWheel.Enabled = False
    scvWheel.ScrollPosition = Floor(scvWheel.ScrollPosition / lblHeight + .5) * _
    lblHeight
End Sub
```

We disable the `Timer` and calculate the final `ScrollPosition` as a multiple of the `lblHeight` with the current scroll position and set it to the `scvWheel.ScrollPosition`.

### 12.7.3 Return the selected value

The source code is in the Classes\CIsWheelSimple3 folder.



We add several new Views :

- Two Panels
  - pnlScreen covers the whole screen to consume the events of other views on the screen.
  - pnlMain holding the wheel, a title and two Buttons.
- A title Label
  - lblTitle
- Two Buttons
  - btnOK returns the selected value and hides pnlScreen.
  - btnCancel hides pnlScreen.

To avoid that events of Views behind the wheel are raised we add a Panel pnlScreen covering the whole screen with an empty pnlScreen\_Click event.

```
Private Sub pnlScreen_Click
    ' empty to consume the pnlScreen events
End Sub
```

The same for pnlMain.

```
Private Sub pnlMain_Click
    ' empty to consume the pnlMain events
End Sub
```

We declare some new variables :

```
Private btnSpace, btnHeight, btnWidth As Int ' button dimensions
Private pnlMainWidth, pnlMainHeight As Int ' pnlMain dimensions
Private Title As String
Private CallbackView As Object
```

We modify the Initialization routine to add the Title variable:

```
' Initializes the Wheel.
' CallbackModule = calling module
' Parent = parent activity or panel
' cTitle = title for the input
' cWheelContent = List object with the content on the wheel
' cFontSize = font size of the text in the wheel
Public Sub Initialize(CallbackModule As Object, Parent As Object, cTitle As _
    String, cWheelContent As List, cFontSize As Float)
```

In the routine we set the dimensions of the buttons and calculate the dimensions of pnlMain. The detail of this code is not explained here you can look at it in the provided source code.

We need to modify the Show routine and add an object for the returned value.

The return object can be a Label, an EditText, a Button or a String.

```
'show the wheel
'cCallBackView = view or string variable that gets the returned value
'Value = value to preset
Public Sub Show(cCallBackView As Object, Value As Object)
    Private index As Int

    CallBackView = cCallBackView

    pnlScreen.Visible = True

    index = WheelContent.IndexOf(Value)
    scvWheel.ScrollPosition = index * lblHeight
    DoEvents
    scvWheel.ScrollPosition = index * lblHeight
    DoEvents
End Sub
```

We get the selected value in this routine :

```
Private Sub GetSelection As String
    Private i As Int

    i = (Floor(scvWheel.ScrollPosition / lblHeight))
    Return WheelContent.Get(i)
End Sub
```

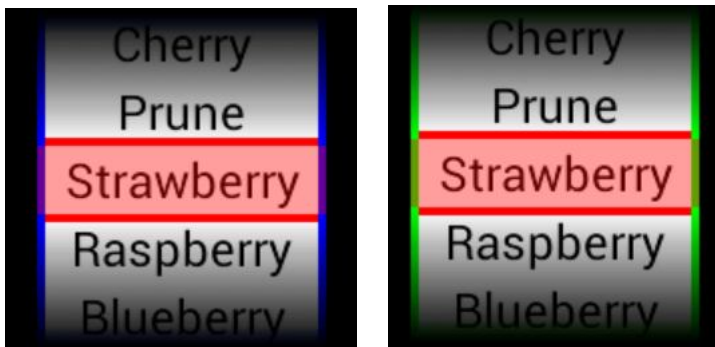
We return the selected value to the Text property of the given view in the Show routine or to the given String variable.

```
Private Sub btnOK_Click
    If CallBackView Is Label Then
        Private lbl As Label
        lbl = CallBackView
        lbl.Text = GetSelection
    Else If CallBackView Is String Then
        CallBackView = GetSelection
    End If
    pnlScreen.Visible = False
End Sub
```

## 12.7.4 Color properties

Another improvement would be to add color properties to the wheels.  
 Since B4A version 2.70 it is possible to add properties to a class.  
 More details in chapter 1 [Chapter 10.5 Add properties to a class.](#)

The source code is in the Classes\ClsWheelSimple4 folder.  
 First we add a BackgroundColor property that changes the color of the lateral lines.



The color can be set with the `setBackGroundColor`.  
 The color can be read back with `getBackGroundColor`.

The code in the class module :

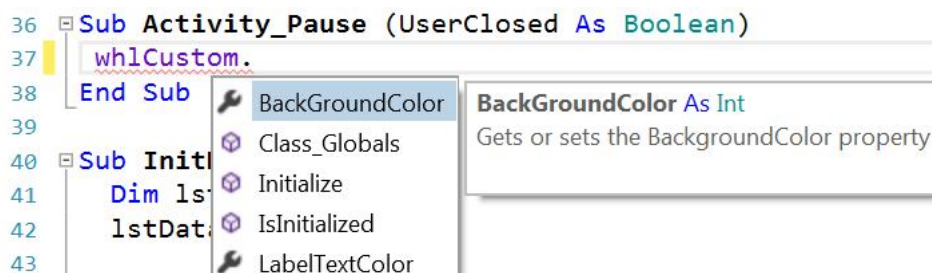
```
'Gets or sets the BackgroundColor property
Sub setBackGroundColor(col As Int)
    col Background = col
    pnl Background.Color = col Background
End Sub

Sub getBackGroundColor As Int
    Return col Background
End Sub
```

The code in the main module.

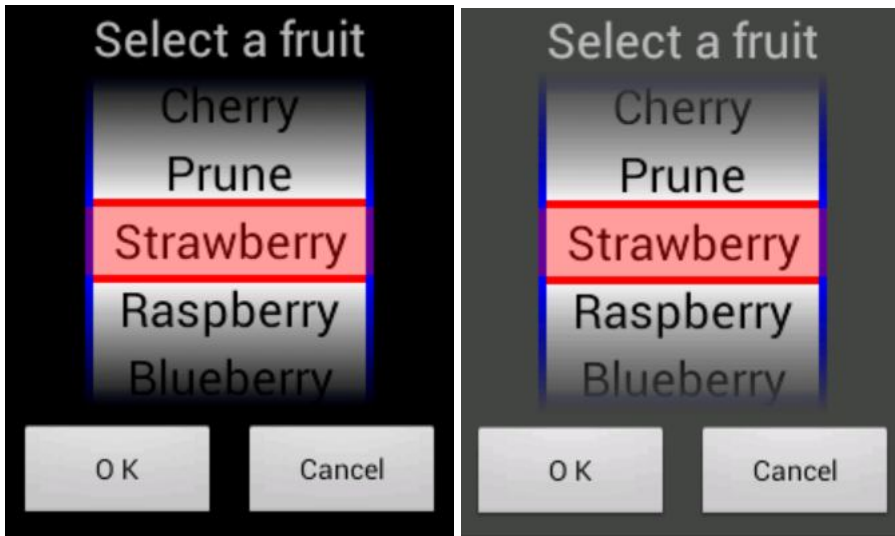
```
whlCustom.BackgroundColor = Colors.Green
```

When you write `whlCustom`, the different properties will be displayed.



As a reminder, a comment line added before the Sub declaration will be displayed in the inline help.  
 'Gets or sets the BackgroundColor property  
 Sub setBackGroundColor(col As Int)

The second color property is the ShadowColor property.



default color : black

new color : dark gray

The code in the class module :

'Gets or sets the ShadowColor property

```
Sub setShadowColor(col As Int)
```

```
    colShadow = col
```

```
    pnlMain.Color = colShadow
```

```
    lblTitle.Color = colShadow
```

```
    TopBackground
```

```
    BottomBackGround
```

```
End Sub
```

```
Sub getShadowColor As Int
```

```
    Return colShadow
```

```
End Sub
```

The code in the main module.

```
whl Custom.ShadowColor = Colors.DarkGray
```

The code to set the colors of the top and the bottom panel has been moved to two separate routines TopBackground and BottomBackGround. This code was previously in the Activity\_Create routine.

```
Private Sub TopBackground
```

```
    Private gdw As GradientDrawable
```

```
    Private cols(2) As Int
```

```
    cols(0) = colShadow
```

```
    cols(1) = Colors.Transparent
```

```
    gdw.Initialize("TOP_BOTTOM", cols)
```

```
    gdw.CornerRadius = 0
```

```
    pnlTop.Background = gdw
```

```
    Private cvs1 As Canvas
```

```
    cvs1.Initialize(pnlTop)' initialize a canvas and draw the line on the bottom
```

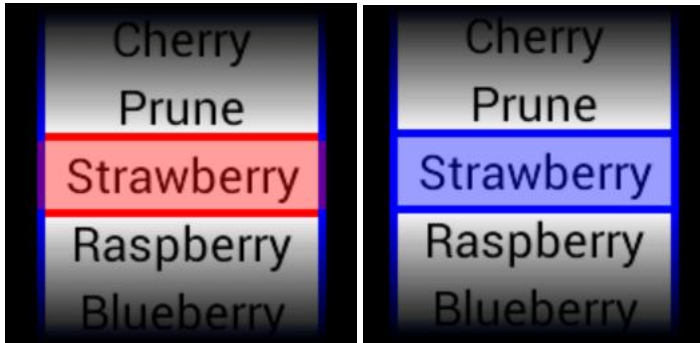
```
    cvs1.DrawLine(lineWidth, pnlTop.Height - lineWidth / 2, Width - lineWidth,
```

```
    pnlTop.Height - lineWidth / 2, colWindowLine, lineWidth)
```

```
End Sub
```

The BottomBackGround routine is similar to TopBackground.

The WindowColor property.



The code in the class module :

```
'Gets or sets the ShadowColor property
Sub setWindowColor(col As Int)
    col WindowLine = col

    ' get the color components
    Private res(4) As Int
    res(0) = Bit.UnsignedShiftRight(Bit.AND(col WindowLine, 0xff000000), 24) ' alpha
    res(1) = Bit.UnsignedShiftRight(Bit.AND(col WindowLine, 0xff0000), 16) ' red
    res(2) = Bit.UnsignedShiftRight(Bit.AND(col WindowLine, 0xff00), 8) ' green
    res(3) = Bit.AND(col WindowLine, 0xff) ' blue
    ' sets the alpha value to 96
    col Window = Colors.ARGB(96, res(1), res(2), res(3))

    TopBackground
    BottomBackGround
    pnl Middle.Color = col Window
End Sub
```

In this routine we get the alpha, red, blue and green components from the `col WindowLine` color and then set the alpha value to 96 for `col Window`.

```
Sub getWindowColor As Int
    Return col WindowLine
End Sub
```

The code in the main module.

```
whl Custom.WindowColor = Colors.Blue
```

The LabelTextColor property.



The code in the class module :

```
'Gets or sets the LabelTextColor property
Sub setLabelTextColor(col As Int)
    col LabelTextColor = col

    Dim i As Int

    For i = 0 To scWheel.Panel.NumberOfViews - 1
        Private lbl As Label
        lbl = scWheel.Panel.GetView(i) ' get the Label from the scrollView
        lbl.TextColor = col LabelTextColor ' set the new color
    Next
End Sub
```

In this routine we get each Label from the Scrollview and set its TextColor property to the new color.

```
Sub getLabelTextColor As Int
    Return col LabelTextColor
End Sub
```

The code in the main module.

```
whlCustom.LabelTextColor = Colors.Red
```

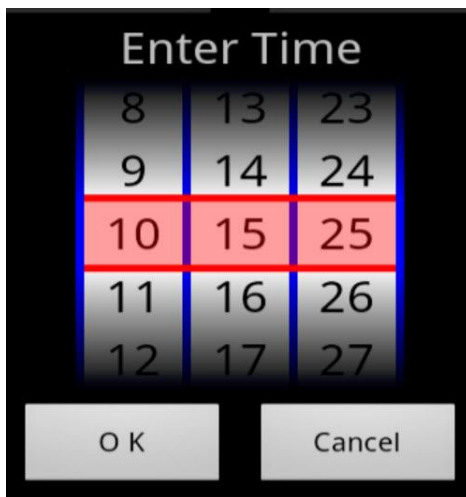


## 12.7.5 A more advanced example

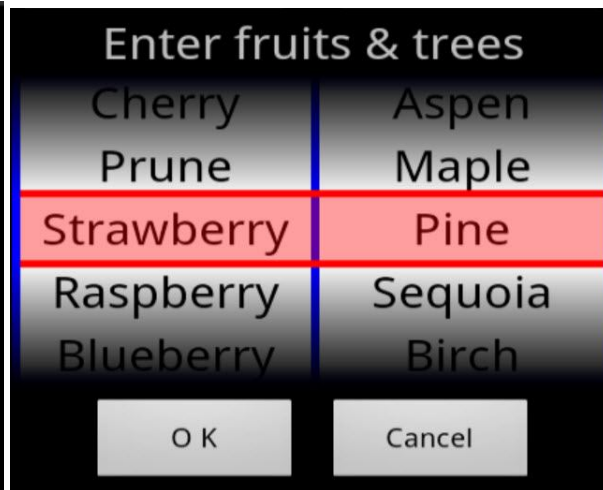
A more advanced Wheel Class can be found in on Forum here [ClsWheel](#).

With several predefined wheels :

- DATE a date input : year / month / day  
A default value can be defined or Now, the current date.  
The returned value has the current DateFormat.
- TIME\_HM a time input : hour / minute  
A default value can be defined or Now, the current time.  
The returned value has the current TimeFormat.
- TIME\_HMS a time input : hour / minute / second  
A default value can be defined or Now, the current time.  
The returned value has the current TimeFormat.
- DATE\_TIME a date + time input : year / month / day / hour / minute  
A default value can be defined or Now, the current date and time.  
The returned value has the current DateFormat and TimeFormat.
- CUSTOM a custom input with user defined input values.  
The number of side by side wheels is user defined (max. 5 wheels).  
A default value can be defined.  
A specific separation character can be defined, a blank character is default.
- INTEGER positive and negative integers.
- INTEGER\_POS only positive integers.
- NUMBER positive and negative numbers.
- NUMBER\_POS only positive numbers.



TIME\_HMS



CUSTOM a custom input  
with user defined input values.

The number of side by side wheels is user defined (max. 5 wheels).

A default value can be defined.

A specific separation character can be defined, a blanc character is default. CUSTOM