B4x Booklets

B4x Getting started

 Edition 1.0

Last update : 2017.06.30

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

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

All the source code and files needed (layouts, images etc.) of the example projects in this booklet are included in the SourceCode folder.
For each program there are three folders.

```
SourceCode
    MyFirstProgram
        B4A
            MyFirstProgram.b4a
        B4i
            MyFirstProgram.b4i
        B4J
            MyFirstProgram.b4j
```

Both programs MyFirstProgram and SecondProgram are almost the same for B4A, B4i and B4J.

Updated for following versions:
B4A version 7.01
B4i version 4.01
B4J version 5.51
B4R version 2.00

Other Booklets:
B4x Baisc Language
B4x IDE Integrated Development Environment
B4x CustomViews

1 B4x

B4x is a suite of BASIC programming languages for different platforms.

B4X suite supports more platforms than any other tool
ANDROID | IOS | WINDOWS | MAC | LINUX | ARDUINO | RASPBERRY PI | ESP8266 | AND MORE...

- **B4A**  **Android**

  B4A includes all the features needed to quickly develop any type of Android app.

- **B4i**  **iOS**

  B4i is a development tool for native iOS applications.
  B4i follows the same concepts as B4A, allowing you to reuse most of the code and build apps for both Android and iOS.

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

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

- **B4R** 

Arduino / ESP8266

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

2 Getting started B4A

B4A is a simple yet powerful development environment that targets Android devices.
B4A language is similar to Visual Basic language with additional support for objects.
B4A compiled applications are native Android applications; there are no extra runtimes or dependencies.
Unlike other IDE's, B4A is 100% focused on Android development.
B4A includes a powerful GUI designer with built-in support for multiple screens and orientations.
No XML writing is required.
You can develop and debug with:
- a real device connected via B4Abridge
- a real device connected via USBcable
- or an Android emulator.

B4A has a rich set of libraries that make it easy to develop advanced applications.
This includes: SQL databases, GPS, Serial ports (Bluetooth), Camera, XML parsing, Web services (HTTP), Services (background tasks), JSON, Animations, Network (TCP and UDP), Text To Speech (TTS), Voice Recognition, WebView, AdMob (ads), Charts, OpenGL, Graphics and more.

Android 1.6 and above are supported (including tablets).

2.1 B4A Trial version

Look at this page for instructions how to use the trial version: https://www.b4x.com/b4a.html

2.2 Installing B4A and Android SDK

B4A depends on two additional (free) components:
- Java JDK
- Android SDK

2.2.1 Installing Java JDK

Installation instructions:
The first step should be to install the **Java JDK**, as Android SDK requires it as well.
Note that there is no problem with having several versions of Java installed on the same computer.

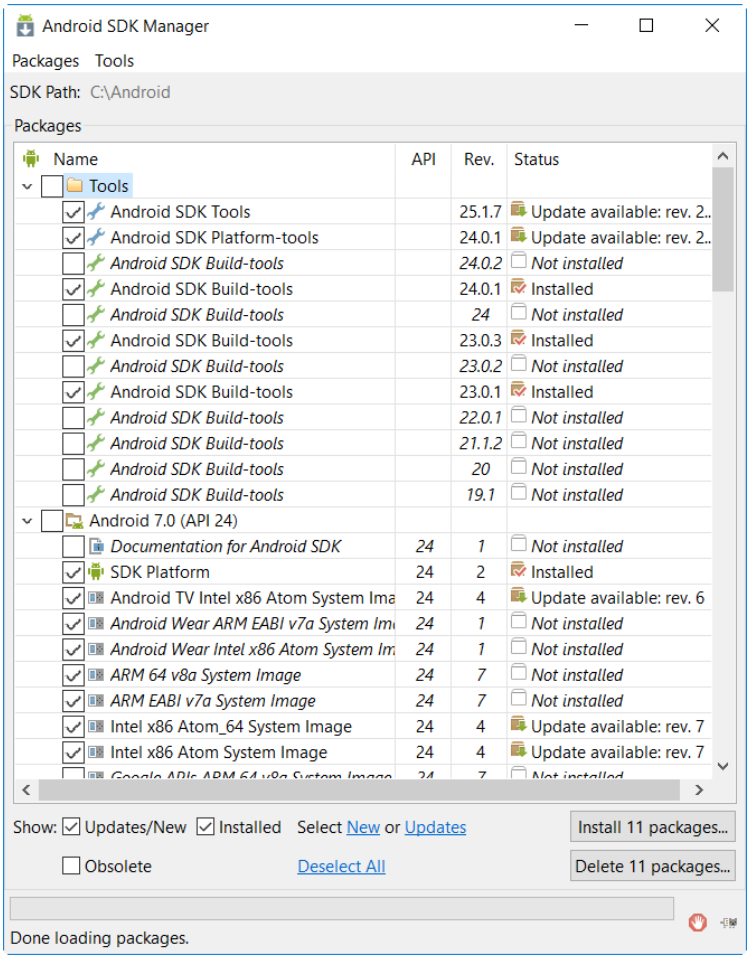
- Open the Java 8 JDK download link.
- Check the Accept License Agreement radio button.
- Select **"Windows x86"** or **"Windows x64"** (for 64 bit machines) in the platforms list.
- Download the file and install it.

2.2.2 Installing Android SDK

The next step is to install the Android SDK and a platform:

- Install the SDK . **The SDK doesn't always behave properly when it is installed in a path with embedded spaces (like Program Files). It is recommended to install it to a custom folder similar to C:\Android**.
- You should now install the platform tools and at least one platform image. Use the latest one or at least API 8.

You should also install Google USB Driver to be able to connect a physical device with USB. A list of other drivers is available here.
Note that B4A allows you to connect to any device over the local network with B4A-Bridge tool.

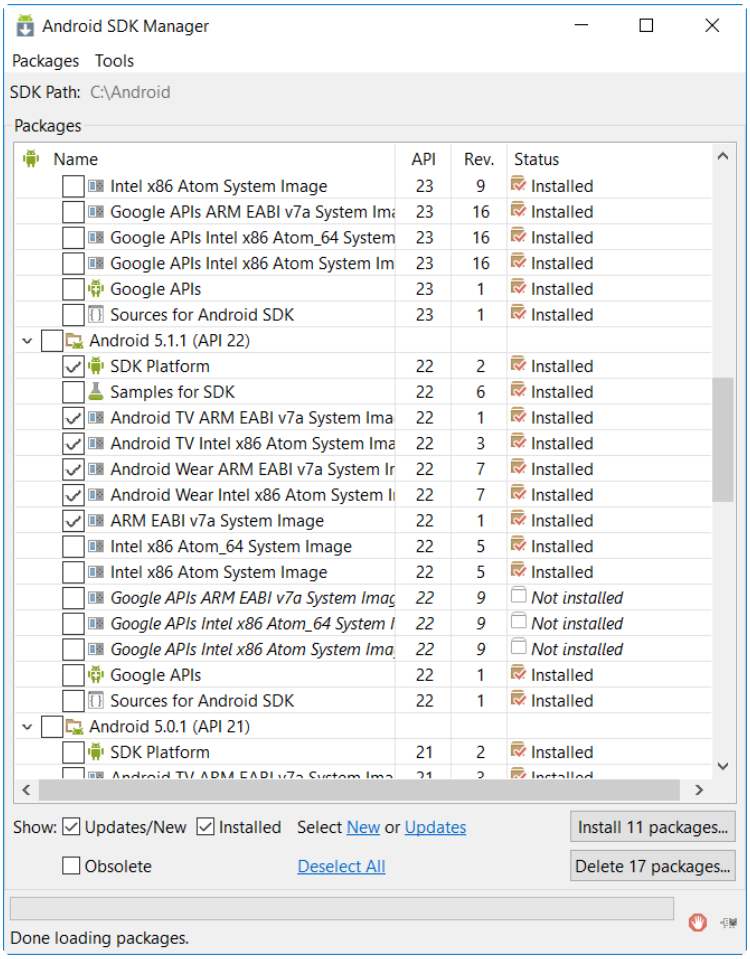


A screen similar to this will be shown.

Select the API version you want to download.
In the example, I choose API 24.

You can select several APIs and install them in parallel.

In this example, API 22 is also selected.



Note that you can install more packages later.

- Press on Install Selected and install both packages.

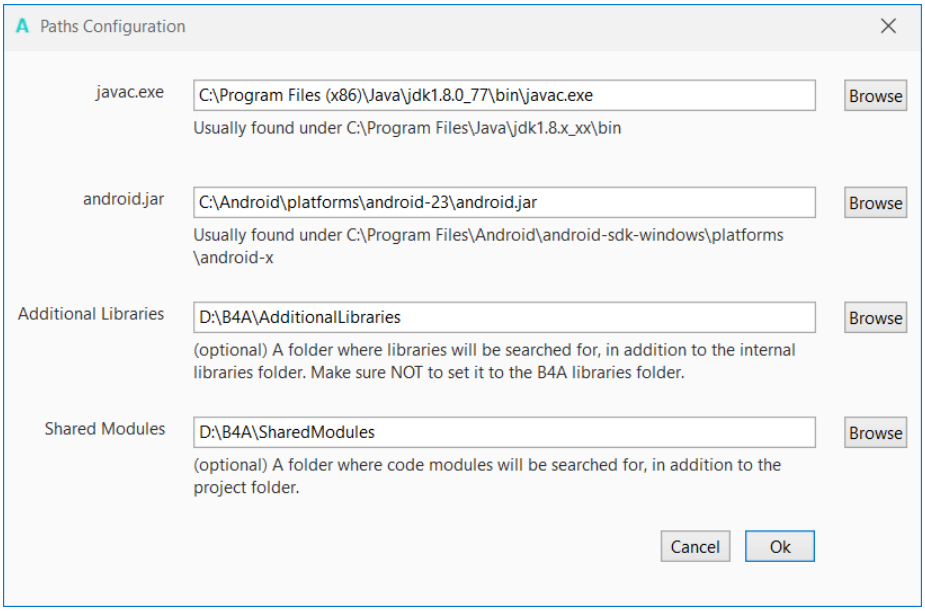
If you want to connect a device with USB you might also download the Google USB driver.

2.2.3 Installing B4A

- Download and install B4A.
- Open B4A.

2.3 Configure Paths in the IDE

- Open B4A.
- Choose Tools menu - Configure Paths.



- Use the browse buttons to locate "javac.exe" and "android.jar"
javac is located under <java folder>\bin.
android.jar is located under <android-sdk-windows>\platforms\android-2**1.**
The folder depends on where you installed the Android SDK,
It should be: C:\Android\platforms\android-21\android.jar
or C:\Android\platforms\android-8\android.jar.
The number depends on the Android version you loaded.

On older versions it could be under:
C:\Android\android-sdk-windows\platforms\android-8\android.jar.
On Windows 64 bit, Java will probably be installed under C:\Program Files (x86).

It is recommended to create a specific folder for Additional libraries.
B4A utilizes two types of libraries:

- Standard libraries, which come with B4A and are located in the Libraries folder of B4A.
  These libraries are automatically updated when you install a new version of B4A.
- Additional libraries, which are not part of B4A, and are mostly written by members. These libraries should be saved in a specific folder different from the standard libraries folder.

Shared modules: Module files can be shared between different projects and must therefore be saved in a specific folder.

Common errors
- Windows XP - "Basic4Android.exe Application could not be initialised correctly error 0xc0000135" on start-up. B4A requires .Net Framework 4.0 or above.
Windows XP users who didn't install it before should first install the framework.

2.4 Connecting a real device

There are different means to connect a real device:

- USB
  Needs that the device supports ADB debugging.
  Need to activate USB Debugging on the device.
- B4A Bridge
  - via WiFi
  - via Bluetooth till B4A version 4.3 it is no more available since version 5.00.

2.4.1 Conneting via B4A-Bridge

It is always recommended to use a real device instead of an Android emulator which is very slow compared to a real device (especially with applications installation).

However not all devices support ADB debugging. This is the reason for the B4A-Bridge tool.
B4A-Bridge is made of two components. One component runs on the device and allows the second component which is part of the IDE to connect and communicate with the device.
The connection is done over a network (B4A-Bridge cannot work if there is no network available).

Once connected, B4A-Bridge supports all of the IDE features which include: installing applications, viewing LogCat and the visual designer.

Android doesn't allow applications to quietly install other applications, therefore when you run your application using B4A-Bridge you will see a dialog asking for your approval.

2.4.1.1 Getting started with B4A-Bridge

First you need to install B4A-Bridge on your device.

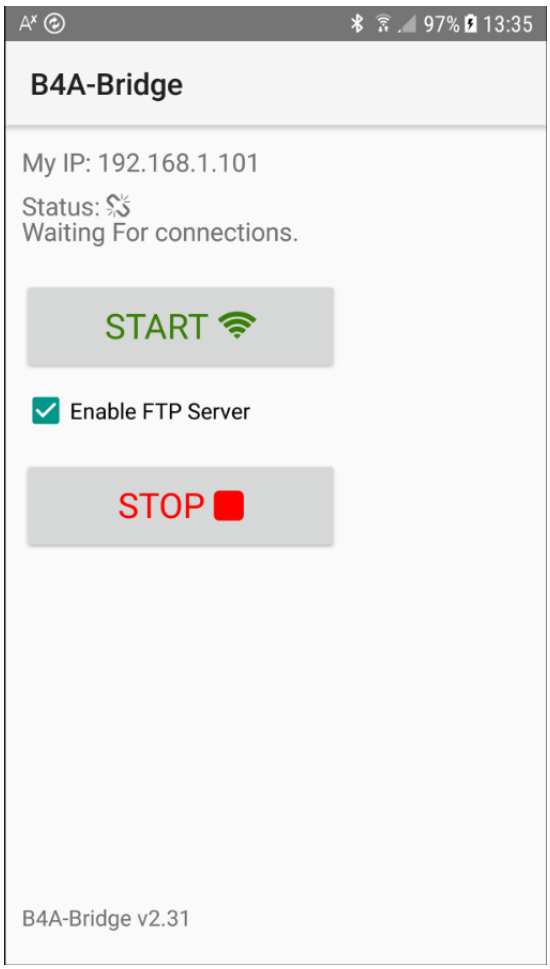
B4A-Bridge can be downloaded here: http://www.basic4ppc.com/android/files/b4a_bridge.apk.

B4A-Bridge is also available on Play Store. Search for: B4A Bridge.
Note that you need to allow install of applications from "Unknown sources". This is done by choosing Settings from the Home screen - Manage Applications.

B4A-Bridge requires writable storage card. It is not possible to install applications without it.

2.4.1.2 Run B4A-Bridge on your device

Run B4A-Bridge on your device, it will display a screen similar to the picture below.



Status will be: Status:
Press on Start to listen for connections.

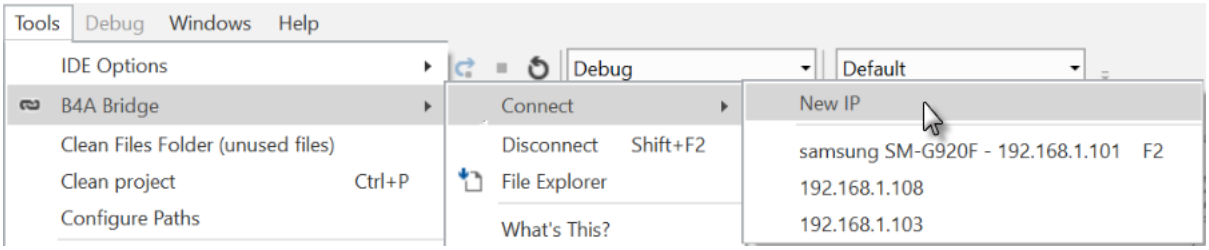
Waiting for connections.

Press START for wireless connection.
Status will change to:
Waiting for connections.

Note that B4A-Bridge was written with B4A.

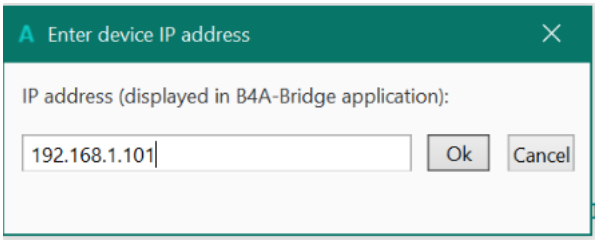
2.4.1.3 Wireless connection

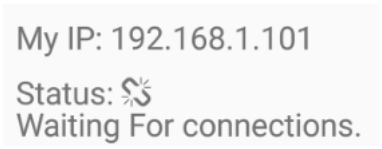
In the IDE menu Tools select New IP.
If the address already exists click directly on this address.
If this device was already connected before you can simply press F2 to connect it.



Enter the IP of the device, you find it on top of the B4A-Bridge screen on the device.

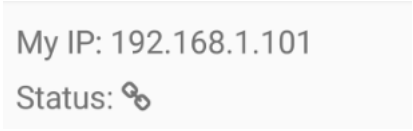
In some cases the address displayed may be the mobile network address. In that case you can find the local wireless address in the wireless advanced settings page.

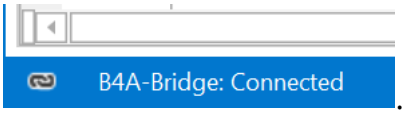




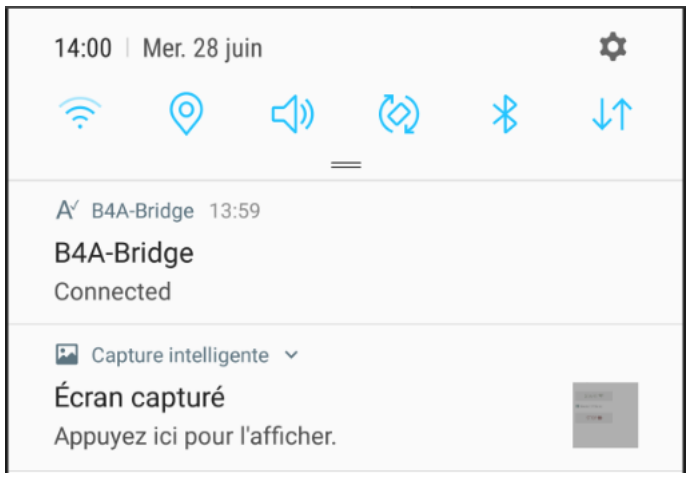
Click on Ok, the device is connected to the IDE.

You see that the status changed on both,
the device and the IDE in the lower left corner.





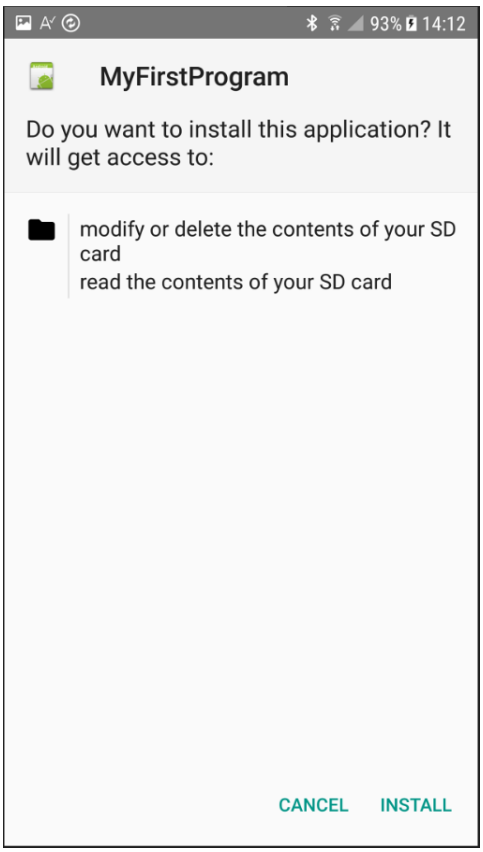
B4A-Bridge keeps running as a service until you press on the Stop button.



You can always reach it by opening the notifications screen.

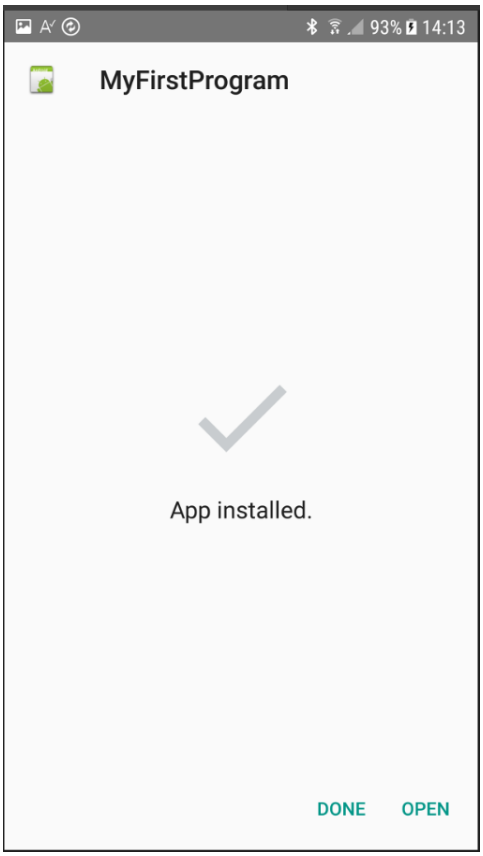
You see B4A-Bridge with the current status.

Note that the Internet permission are automatically added in debug mode.



When you run an application you are required to approve the installation. You will usually see a screens like the picture.

Press on INSTALL to install the program.



If you pressed on INSTALL you will see a screen like in picture.

On this screen you should choose OPEN to start the application.
If you try to install an existing application signed with a different key, the install will fail (without any meaningful message). You should first uninstall the existing application. Go to the home screen - Settings - Applications - Manage applications - choose the application - Uninstall.

Once you finished developing you should press on the Stop button in B4A-Bridge in order to save battery.

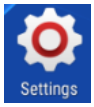
2.4.2 Connecting vis USB

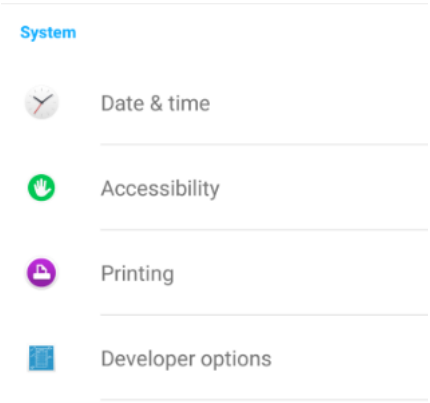
You should download Google USB Driver in the Android SDK Manager.
If this driver doesn't work you must search for a specific driver for your device.

To be able to connect a device with USB you must activate USB Debugging.
This is also need if you use an Emulator.

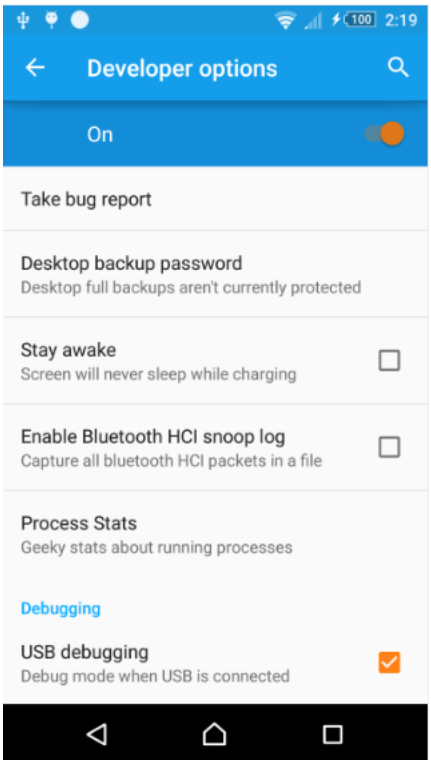
In this state on some older devices you will not be able to access the SD card from the PC.
If you want to access the SD card you must uncheck USB Debugging.

Launch Settings





In System, select Developer options.



Select On.

Check USB Debugging

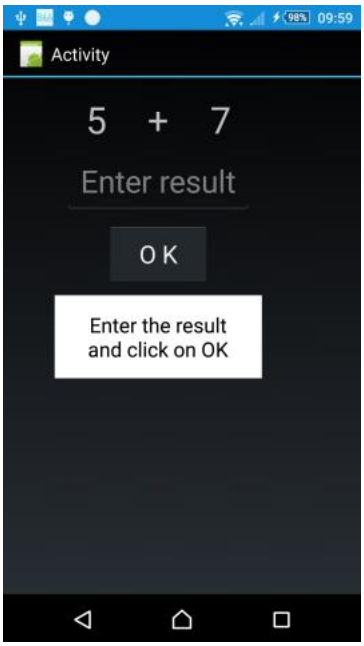
The device will automatically be recognized by the IDE.

2.5 My first B4A program (MyFirstProgram.b4a)

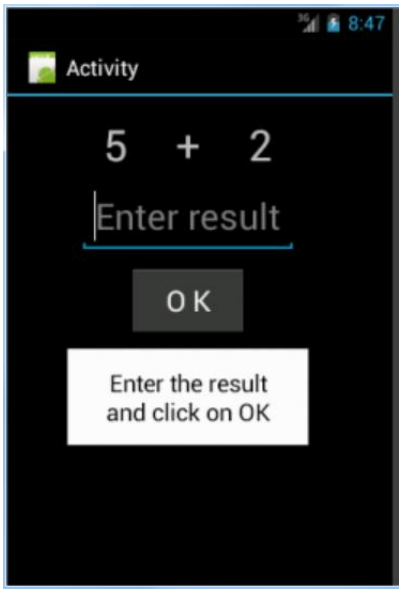
Let us write our first program. The suggested program is a math trainer for kids.

The project is available in the SourceCode folder shipped with this booklet:
SourceCode\MyFirstProgram\ B4A\MyFirstProgram.b4a

The look of the screen is different depending on the Android version of the devices, also with Emulators.



Sony xperia z1



Emulator Android version 4.2

Emulator Android version 2.2

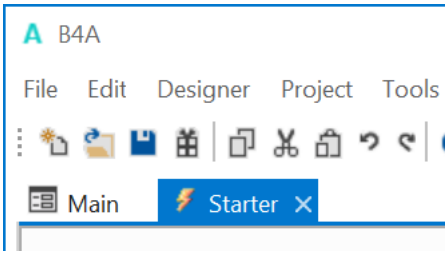
On the screen, we will have:
- 2 Labels displaying randomly generated numbers (between 1 and 9)
- 1 Label with the math sign (+)
- 1 EditText view where the user must enter the result
- 1 Button, used to either confirm when the user has finished entering the result or generate a new calculation.
- 1 Label with a comment about the result.

In Android:
- Label is an object to show text.
- EditText is an object allowing the user to enter text.
- Button is an object allowing user actions.

We will design the layout of the user interface with the VisualDesigner and go step by step through the whole process.

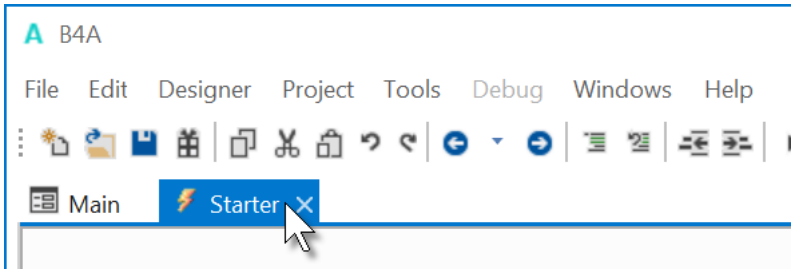
Run the IDE 

When you open the IDE you will see on the top left two Tabs Main and Starter.

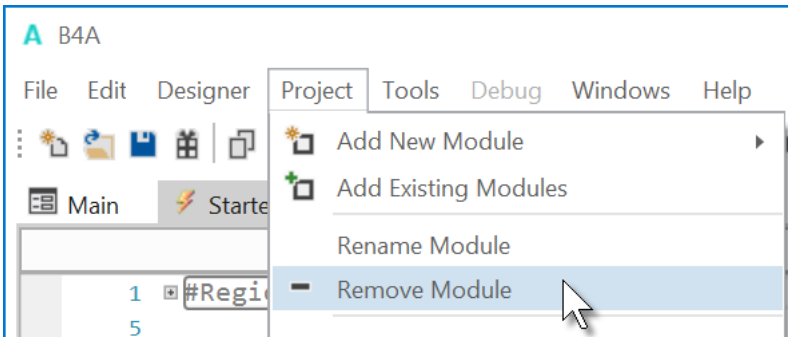


- Main
  Is the Main module for B4A which is the normal starting module. Its name cannot be changed.
- Starter
  Is a Service, which is startet when the program is lauched.

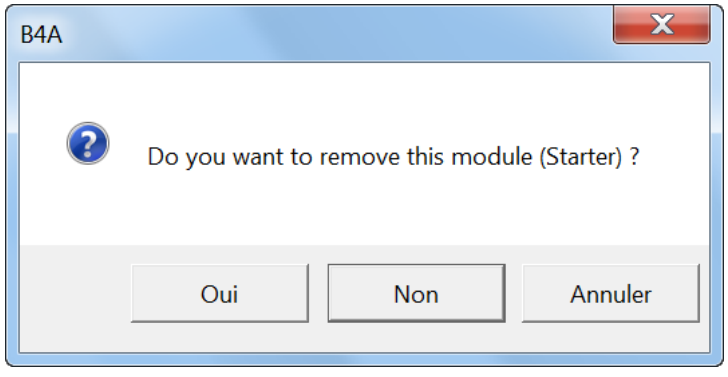
For our first program we don't need this Starter service, so we delete it.



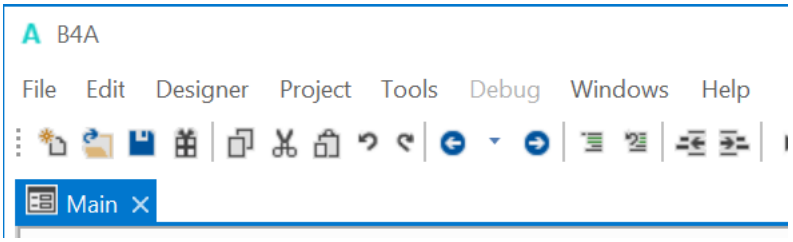
Click on Starter tab to select the service.



In the menu Project click on Remove Module.



You are asked if you really want to remove it.
Click on OUI (YES)



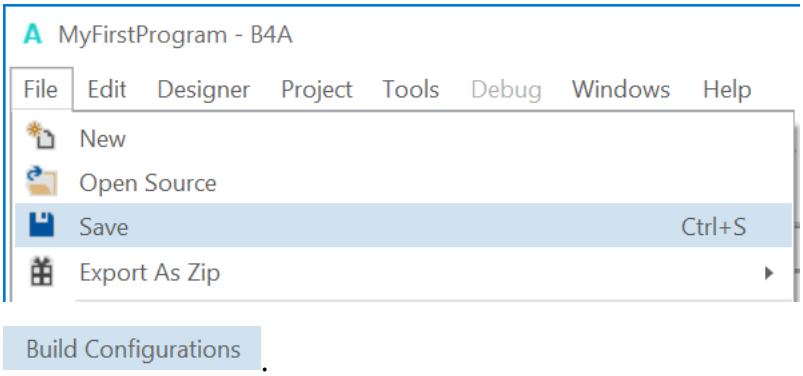
The Starter service module is removed.

You could also leave the Starter service, the removal is not mandatory.

Save the project.

You must save the project before you can run the Designer.

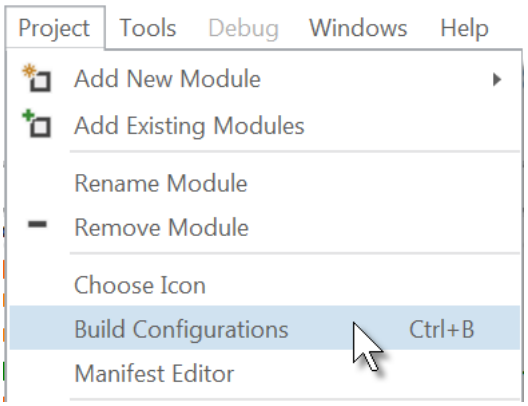
Create a new folder MyFirstProgram and save the project with the name MyFirstProgram.



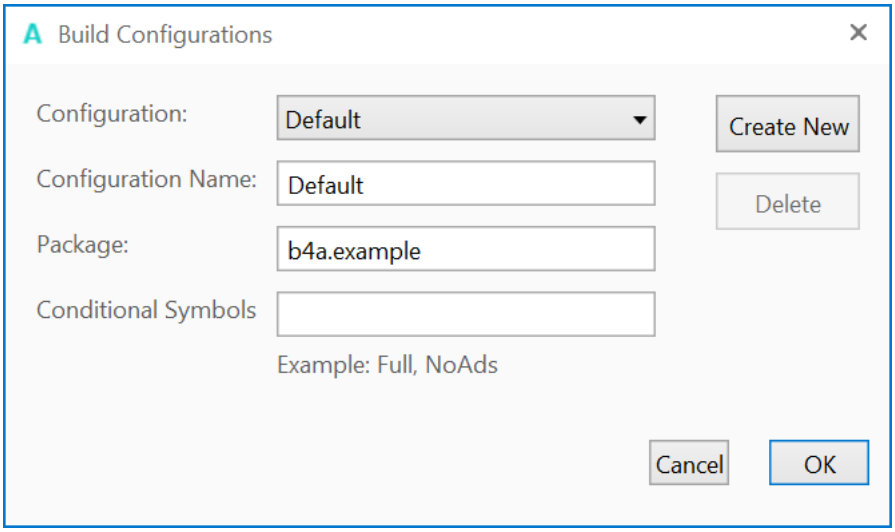
Set the Package Name.

Each program needs a package name.

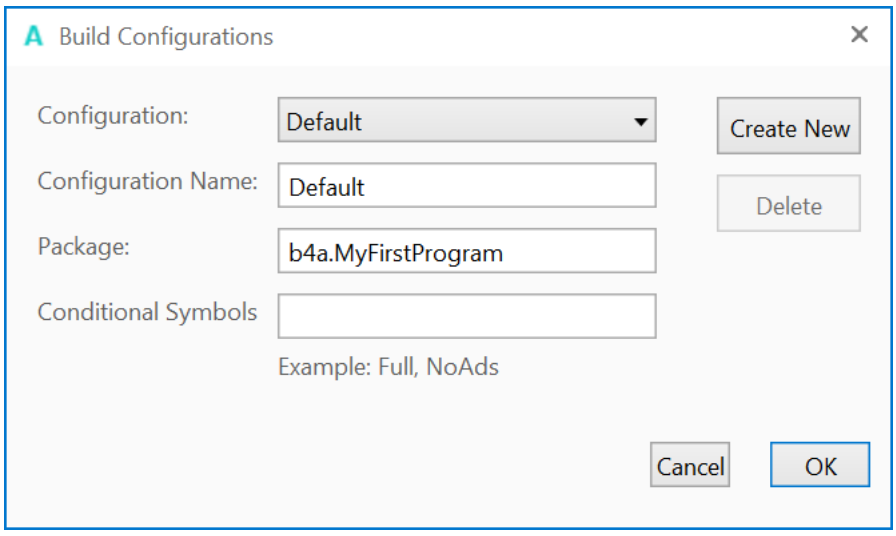
In the menu Project click on Build Configurations.



This window appears:



The default name is b4a.example.



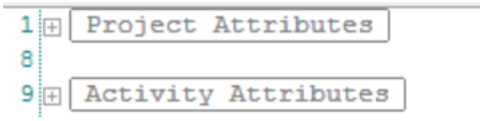
We will change it to b4a.MyFirstProgram.

And click on OK.

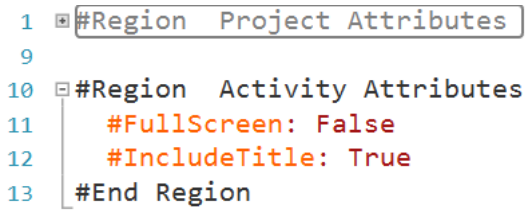
Set the Application Label.

The Application label is the name of the program that will be shown on the device.

On top of the code screen you see these two lines showing two 'regions'.



Regions are code parts which can be collapsed or extended.
Clicking on ⊞ will expand the Region.
Clicking on ⊟ will collapse the Region.
Regions are explained in Chapter Collapse a Region.




```
#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
```

The default name is `B4A Example`, but we will change it to `MyFirstProgram` for naming consistency.

Change this line:

```
   #ApplicationLabel: B4A Example
```

to

```
   #ApplicationLabel: MyFirstProgram
```

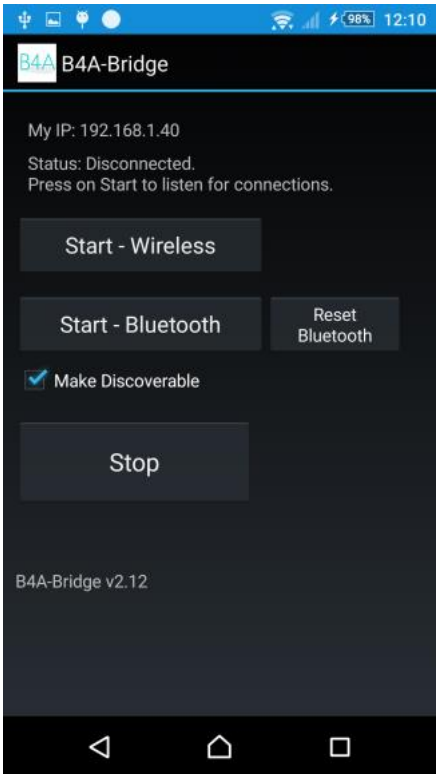
The other lines are explained in Chapter Code header Project Attributes / Activity Attributes.

Connect a device

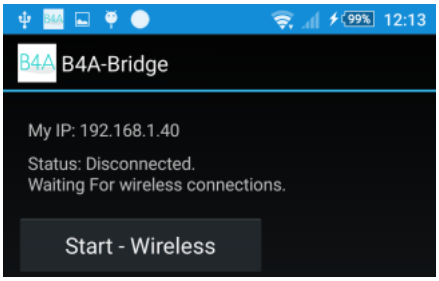
To test the program you should connect a device to the IDE.
The best connection is via B4A-Bridge.

It is also possible to connect an Emulator.

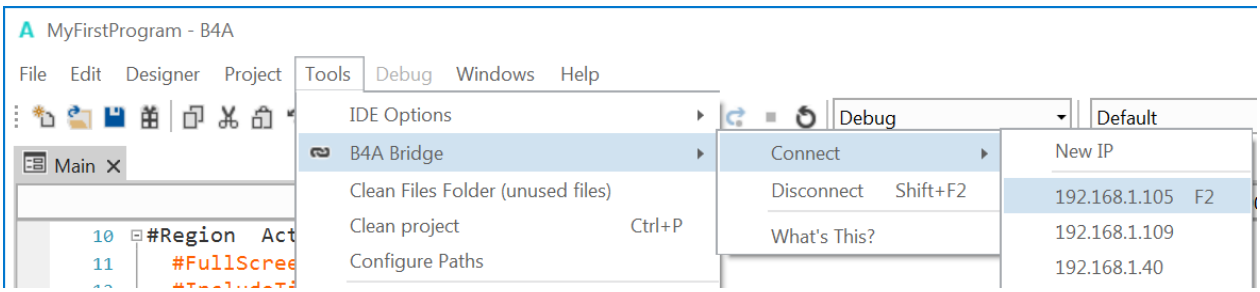
On the device run B4A-Bridge. If you haven't it on your device it's the right moment to install it.



Click on Start - Wireless.

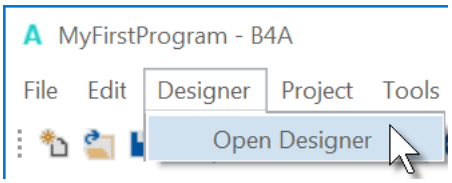


You will see Status: Disconnected. Waiting For wireless connections. on top.



In the IDE click on the address of the device you want to connect.
The address is shown on the B4A-Bridge screen on the device.

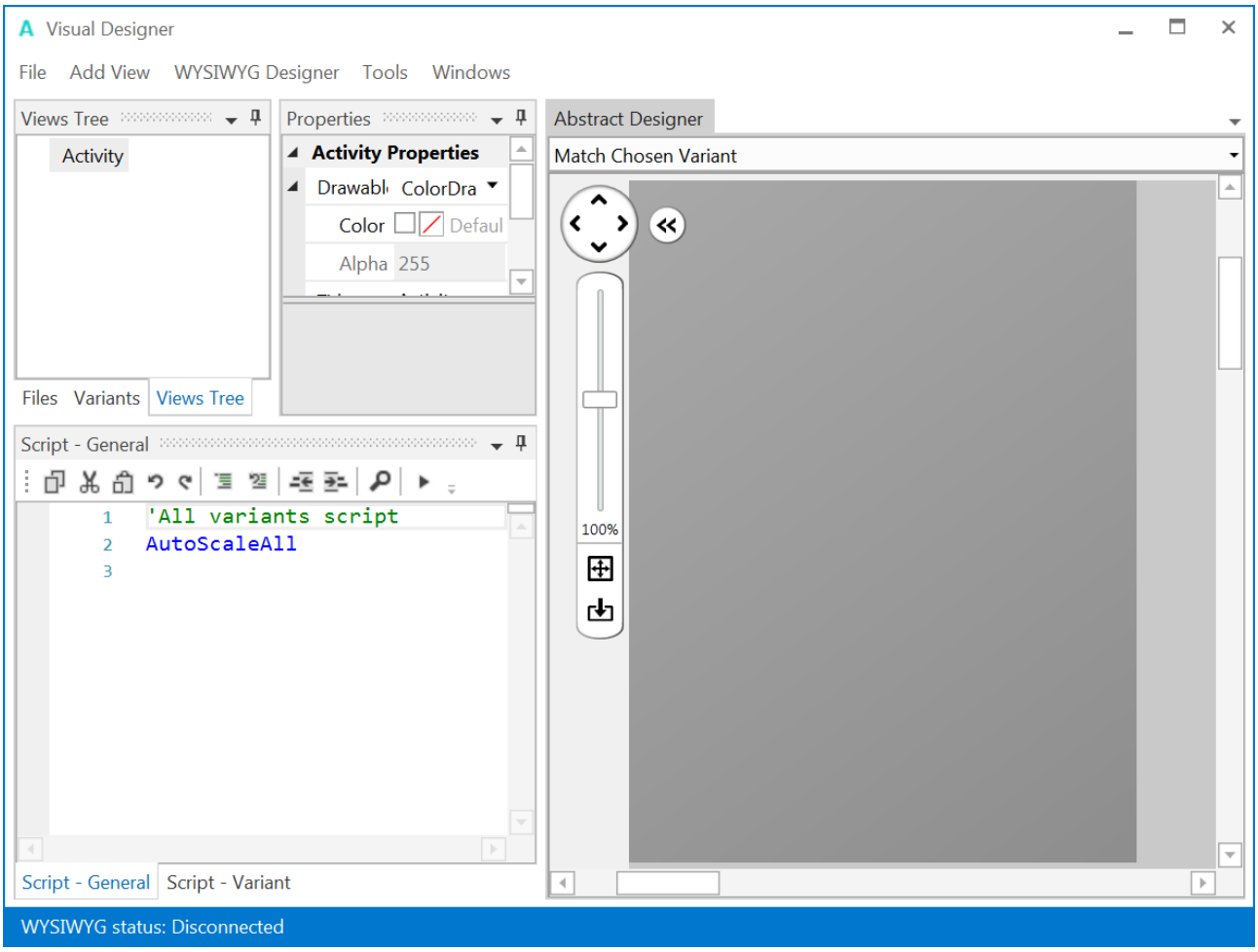
In the IDE run the Designer.



Click on 

.

The Visual Designer looks like this.



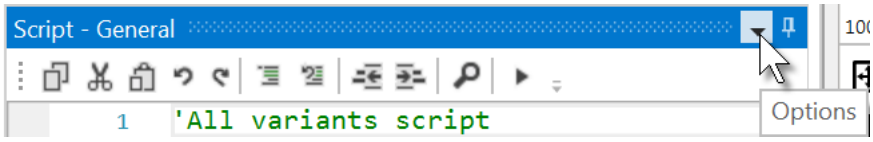
There are different windows:

- Views Tree shows all views as a tree.
- Properties shows all properties of the selected view.
- Abstract Designer shows the views on a screen
- Script - General allows to 'fine tune' the layouts.

The Designer is explained in detail in the chapter The Designer.

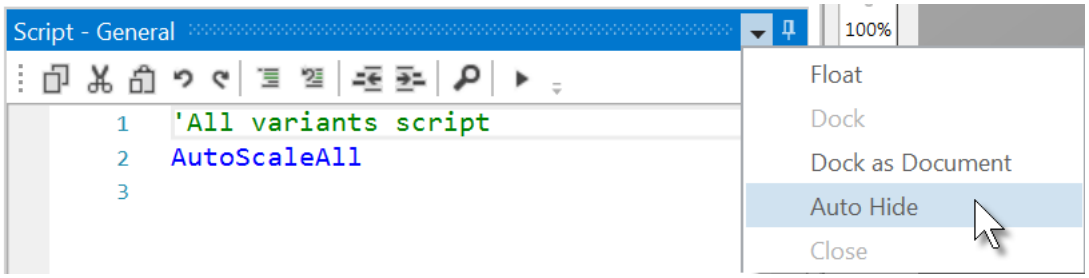
In this first project we will only look at the three first windows.

So we hide the Script- General window to increase the size of the two other windows on top.
Click on  .

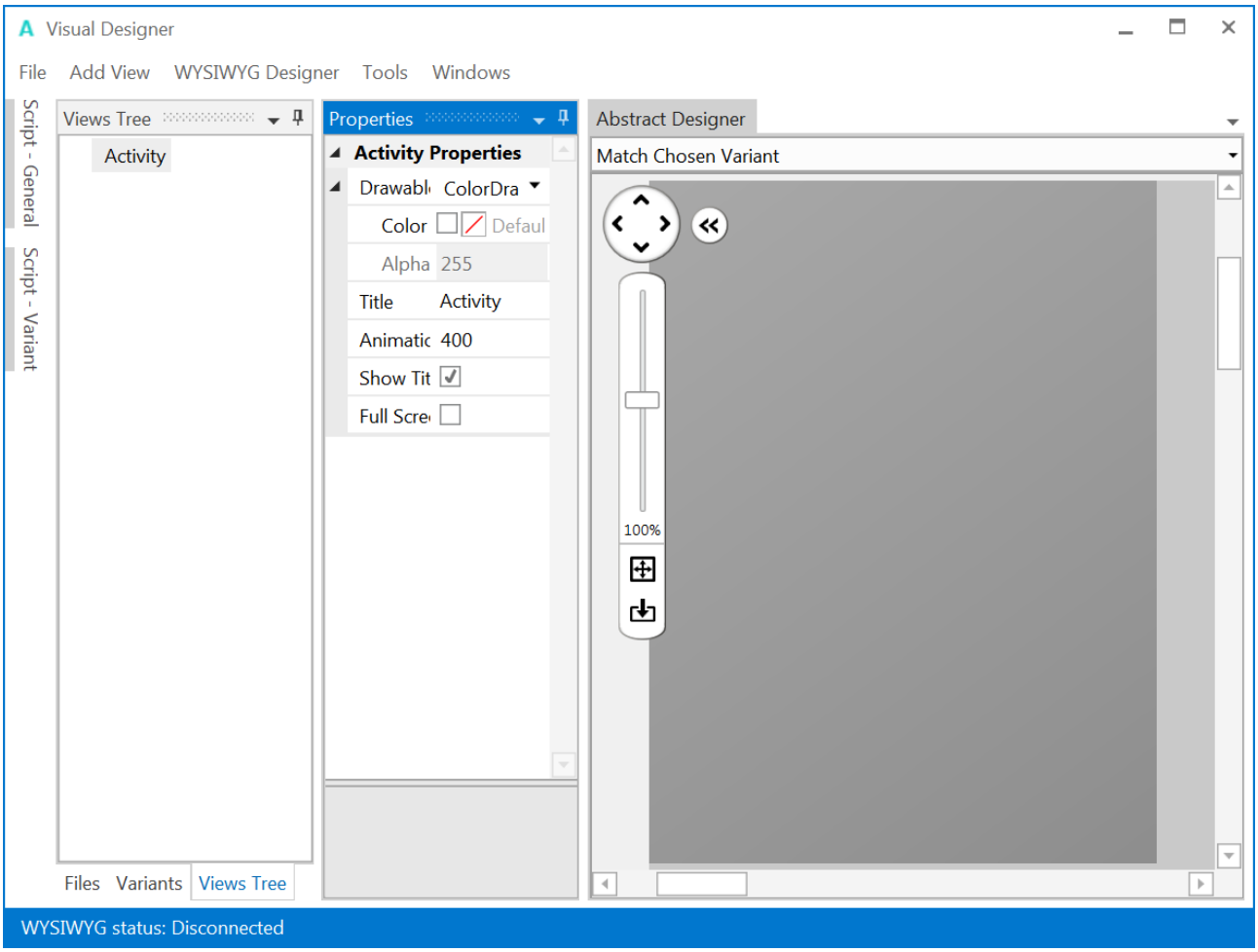


And on 

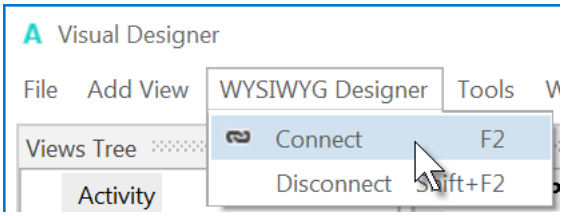
.



The Designer will look like this.

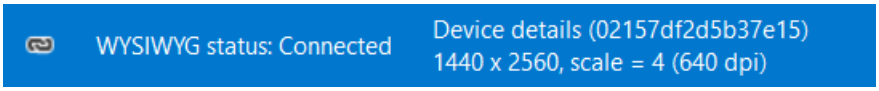


To shows the views on the device you must connect the device to the Designer.

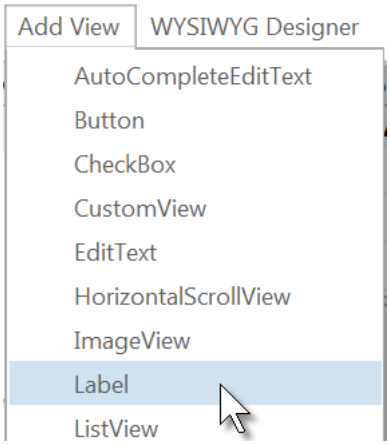


Wait until the Designer and the device are connected. This can take some time, so be patient.

You will see the state of the Designer here on the bottom of the Designer with the parameters of the connected device:

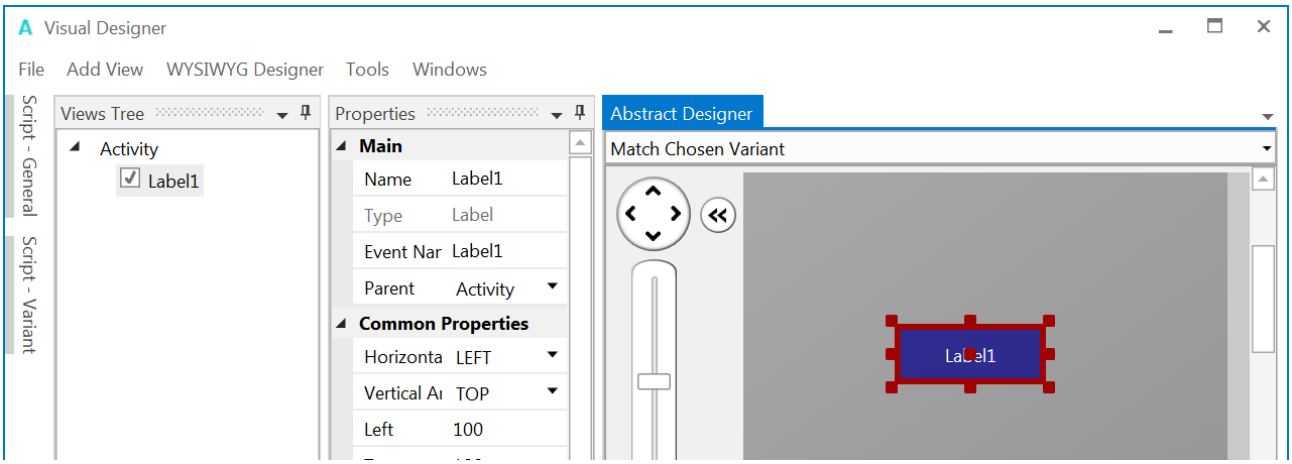


Now we will add the 2 Labels for the numbers.
In the Designer, add a Label.

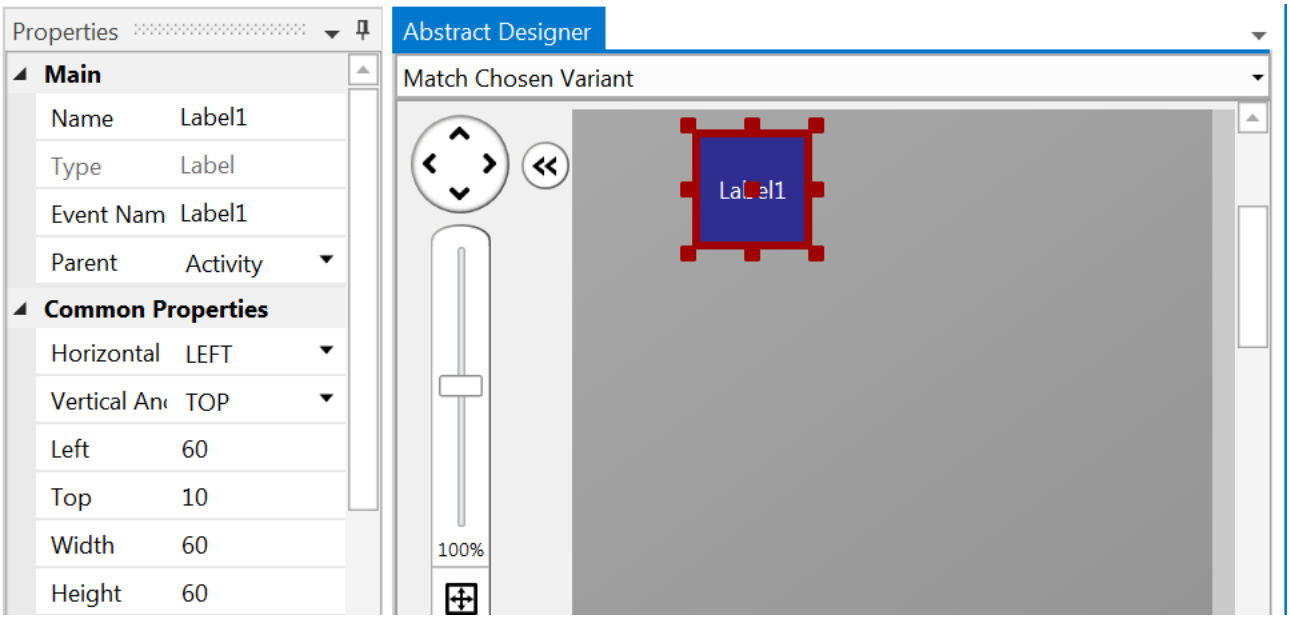


We see the Label with the default name `Label1` in following windows:

Views Tree	**Properties** with its default properties.	**Abstract Designer** at its default position and Default dimensions.



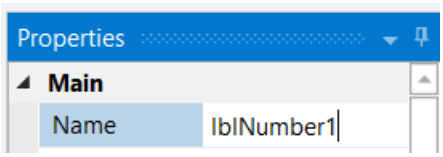
Resize and move the Label with the red squares like this.



The new properties Left, Top, Width and Height are directly updated in the Properties window.
You can also modify the Left, Top, Width and Height properties directly in the Properties window.

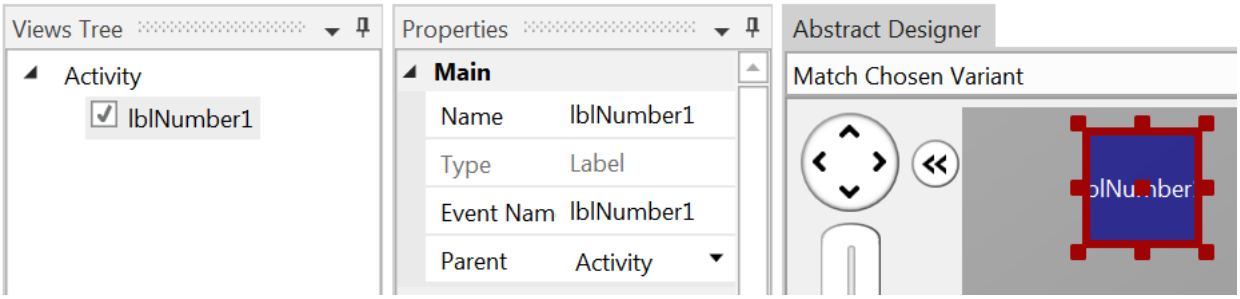
Let us change the properties of this first Label according to our requirements.

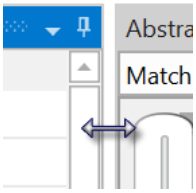
By default, the name is Label with a number, here Label1, let us change its name to lblNumber1.
The three letters 'lbl' at the beginning mean 'Label', and 'Number1' means the first number.
It is recommended to use meaningful names for views so we know directly what kind of view it is and its purpose.



Pressing the 'Return' key or clicking elsewhere will update the name in the other windows and change the Event Name property.

Main:	Main module.
Name:	Name of the view.
Type:	Type of the view. In this case, Label, which is not editable.
Event Name:	Generic name of the routines that handle the events of the Label.
Parent:	Parent view the Label belongs to.





To better see the other properties we enlarge the Properties window.

Let us check and change the other properties:

Common Properties	
Horizontal Anchor	LEFT
Vertical Anchor	TOP
Left	60
Top	10
Width	60
Height	60
Enabled	☑
Visible	☑
Tag	
Text	5
Text Style	
Typeface	DEFAULT
Style	NORMAL
Horizontal Alignm	CENTER_HORIZON
Vertical Alignment	CENTER_VERTICAL
Size	36
Text Color	Default color
Label Properties	
Drawable	ColorDrawable
Color	Default color
Alpha	0
Corner radius	0
Border Color	#FF000000
Border Width	0

Left, Top, Width and Height are OK.
Or if the values are not the same you should change them.

Enabled, Visible are OK

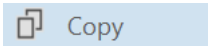
Tag, we leave empty.
Text, we set a default number, say 5

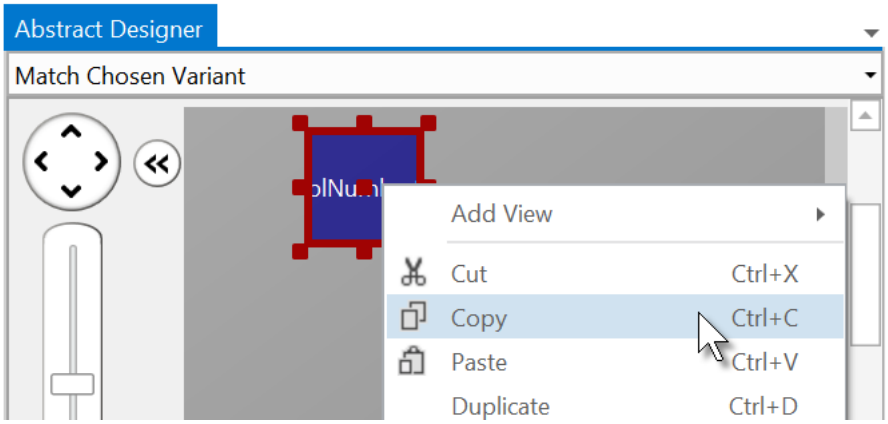
Typeface, Style are OK

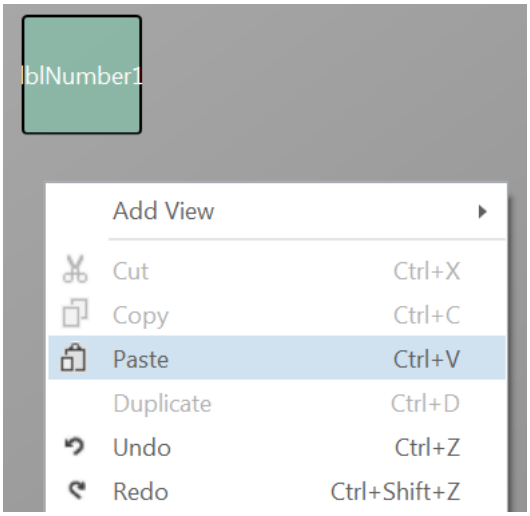
Horizontal Alignment, we set to CENTER_HORIZONTAL
Vertical Alignment, we leave CENTER_VERTICAL.
Size, we set to 36

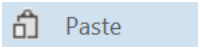
We leave all the other properties as they are.

We need a second Label similar to the first one. Instead of adding a new one, we copy the first one with the same properties. Only the Name and Left properties will change.

Right click in the Abstract Designer on lblNumber1 and click on 

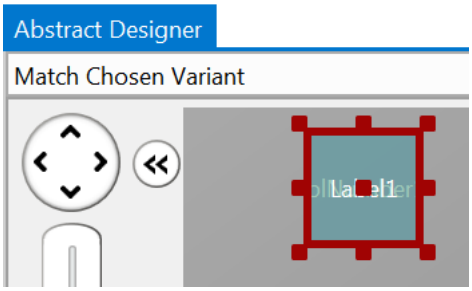


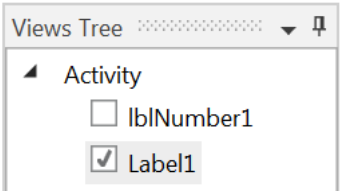


Click somewhere else in the Abstract Designer and right click again and click on 

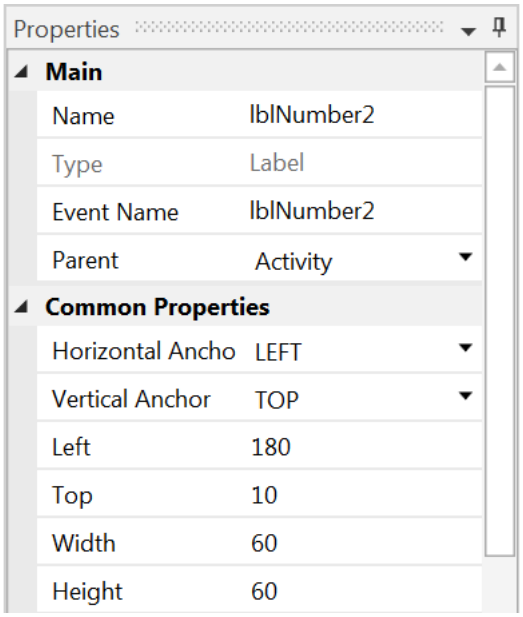
.

The new label covers the previous one.



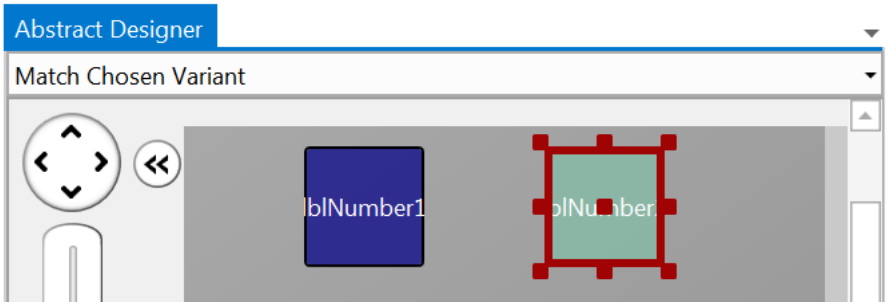


We see the new label added in the Views Tree.



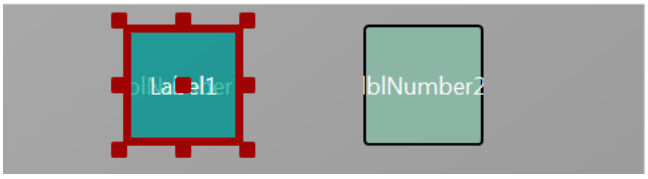
Change its name to lblNumber2.

Change the Left property to 180.

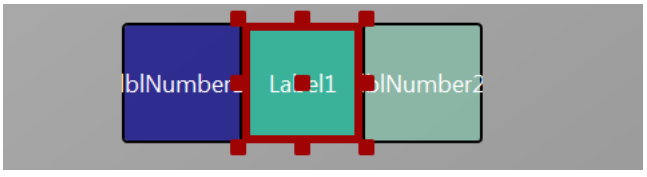


The new label with its new name and at its new position

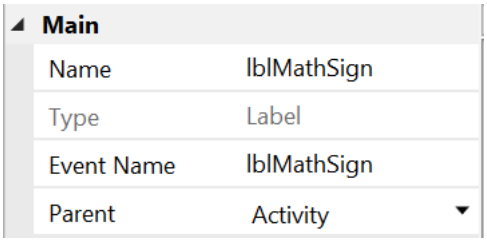
Now we add a 3rd Label for the math sign. We copy once again lblNumber1.
In the Abstract Designer right click on lblNumber1, click on Copy.
Click somewhere else, right click again and click on Paste.

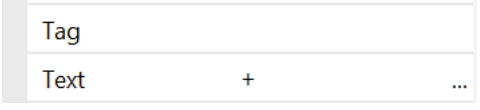


The new label covers lblNumber1.

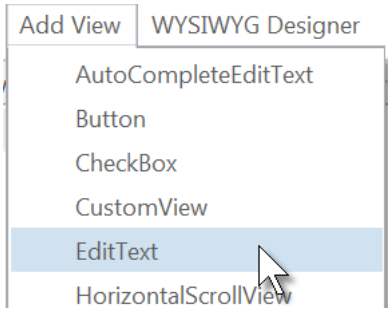


Position it between the first two Labels and change its name to lblMathSign and its Text property to '+'.

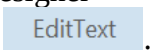




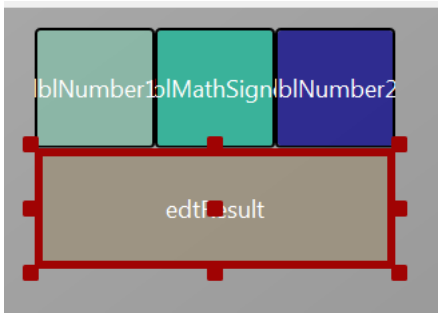
Now let us add an EditText view.



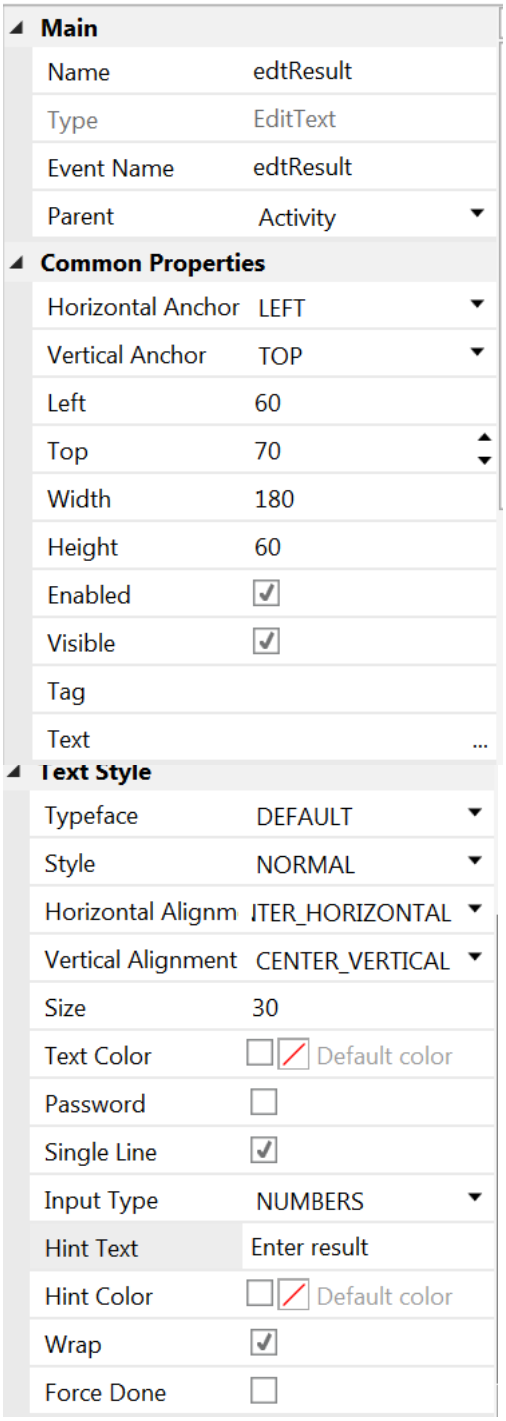
In the Designer 

menu click on 

.



Position it below the three Labels and change its name to `edtResult`. `'edt'` means EditText and `'Result'` for its purpose.



Let us change these properties.
Name to edtResult

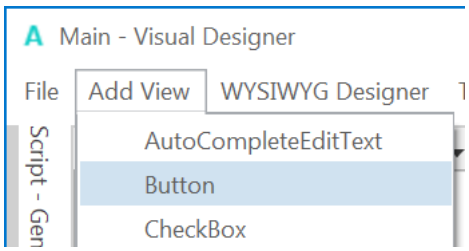
Horizontal Alignment to CENTER_HORIZONTAL

Text Size to 30

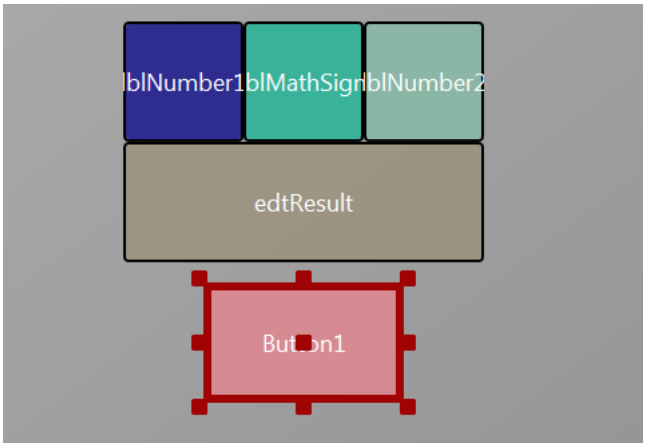
Input Type to NUMBERS
Hint Text to Enter result

Setting Input Type to NUMBERS lets the user enter only numbers.

Hint Text represents the text shown in the EditText view if no text is entered. After making these changes, you should see something like this.

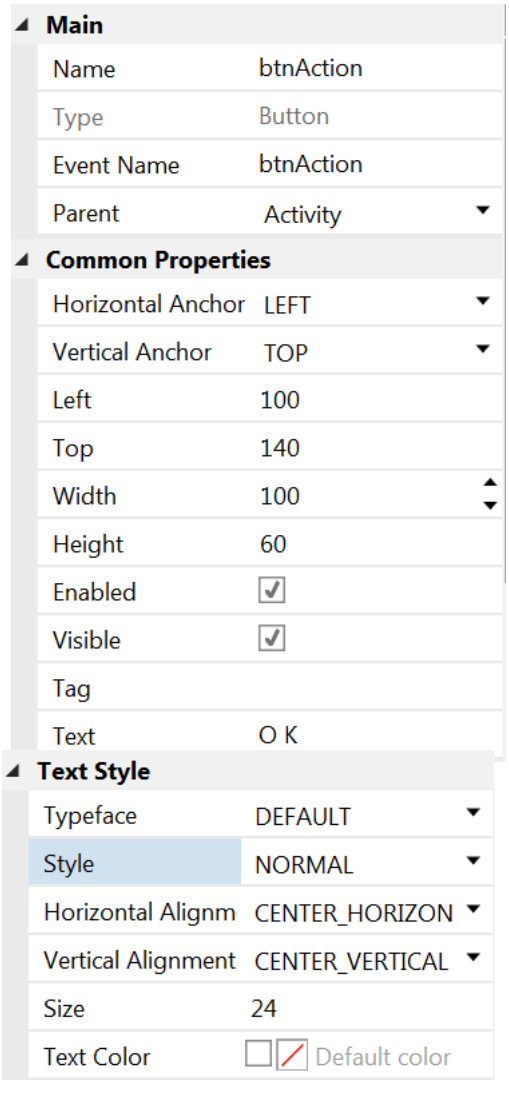


Now, let's add the Button which, when pressed, will either check the result the user supplied as an answer, or will generate a new math problem, depending on the user's input.



Position it below the EditText view. Resize it and change following properties:

Set the properties like below.

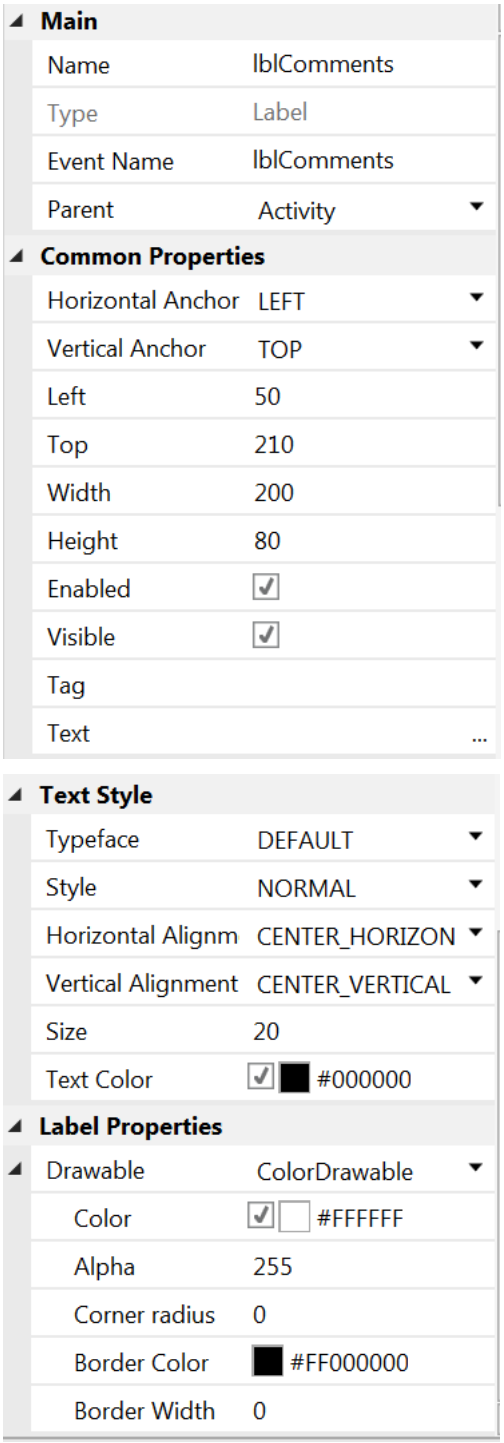


Name to btnAction

Text to O K (with a space between O and K)

Text Size to 24

Let us add the last Label for the comments. Position it below the Button and resize it.



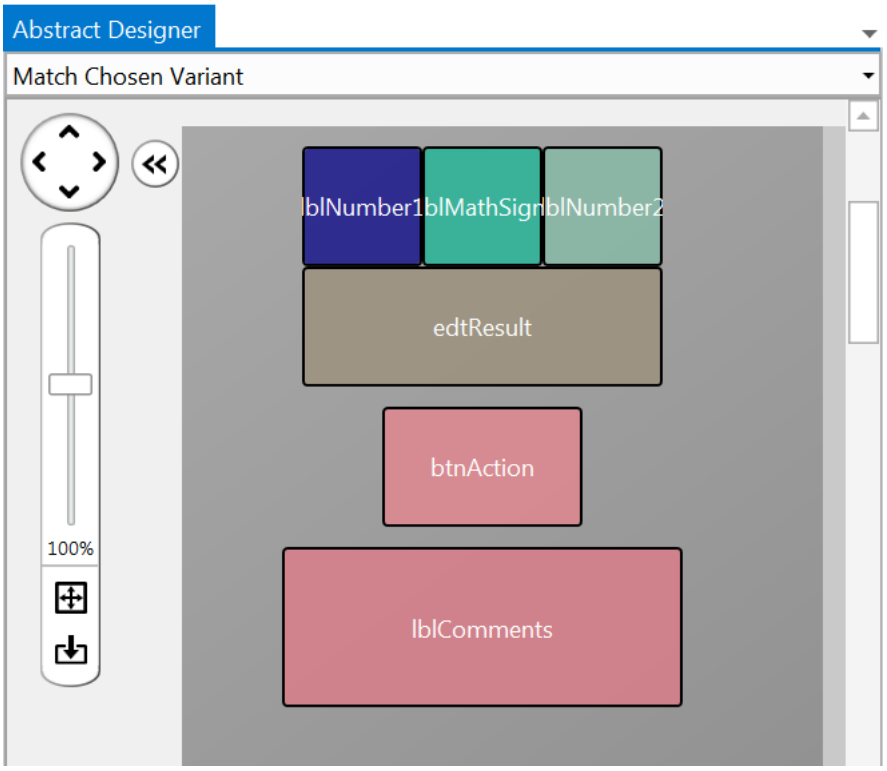
Change the following properties:
Name to lblComments

Horizontal Alignment CENTER_HORIZONTAL

Text Color to #000000
We set the Text Color property to Black (#000000).

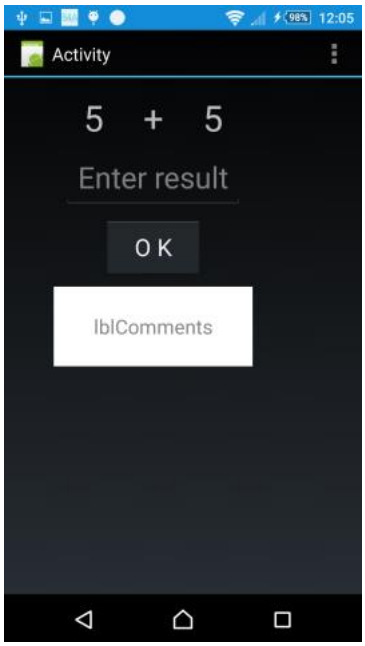
Color to #FFFFFF
Alpha to 255

By default, the Label background color is black and transparent.
We set it to white and opaque Alpha = 255.

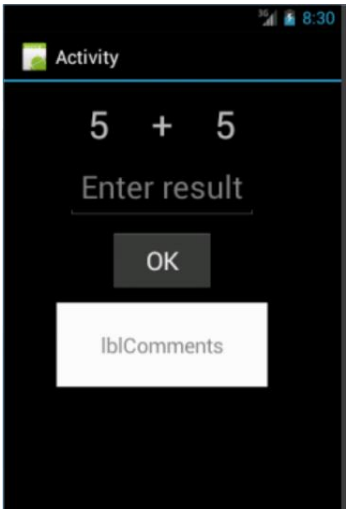


The result will look like this in the Designer.

And on a device or Emulator.



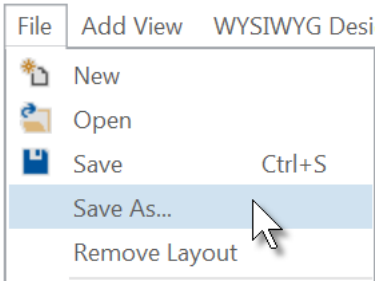
Sony xperia z1



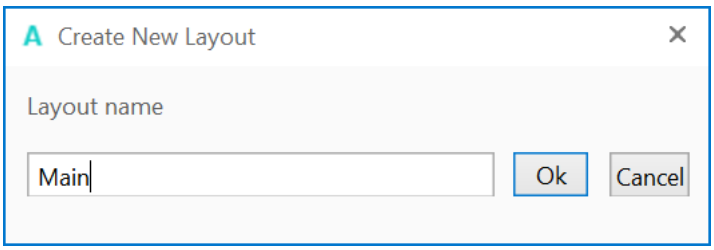
Android 4.2 Emulator

Android 2.2 Emulator

Let us save the layout in a file.



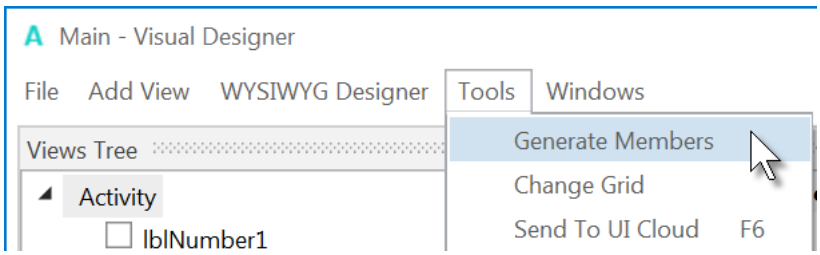
In the File menu click on Save As... and save it with the name 'Main'.



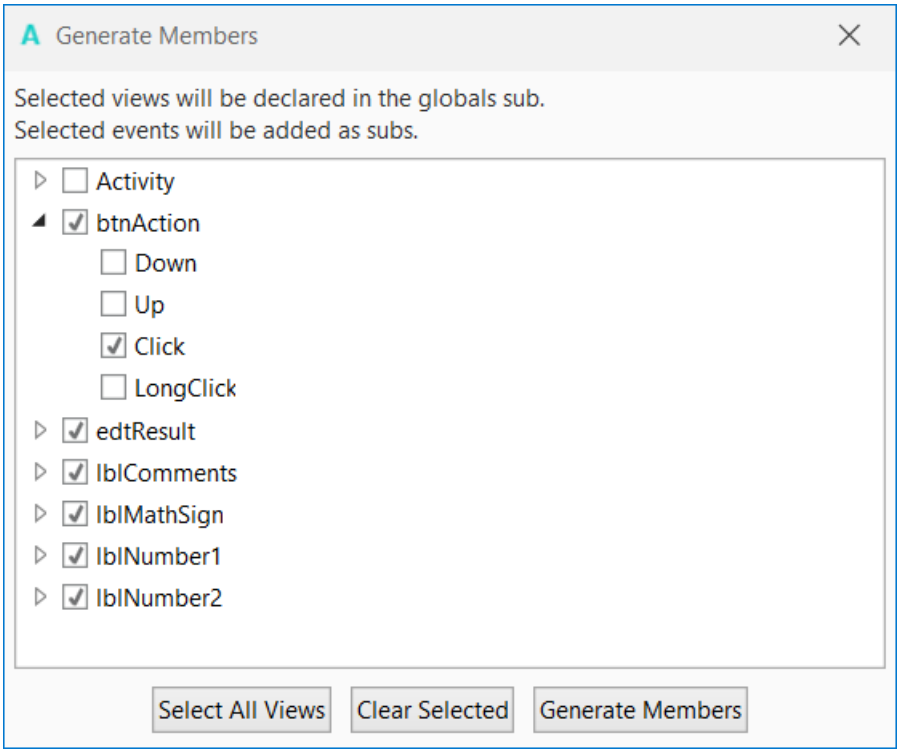
Click on Ok .

To write the routines for the project, we need to reference the Views in the code.
This can be done with the *Generate Members* tool in the Designer.

The *Generate Members* tool automatically generates references and subroutine frames.



In the Tools menu click on Generate Members to open the generator.



Here we find all the views added to the current layout.
We check all views and check the Click event for the btnAction Button.
Checking a view ▷ ☑ edtResult generates its reference in the Globals Sub routine in the code.
This is needed to make the view recognized by the system and allow the autocomplete function.

```
Private btnAction As Button
Private edtResult As EditText
Private lblComments As Label
Private lblMathSign As Label
Private lblNumber1 As Label
Private lblNumber2 As Label
```

Clicking on an event of a view ☑ Click generates the Sub frame for this event.

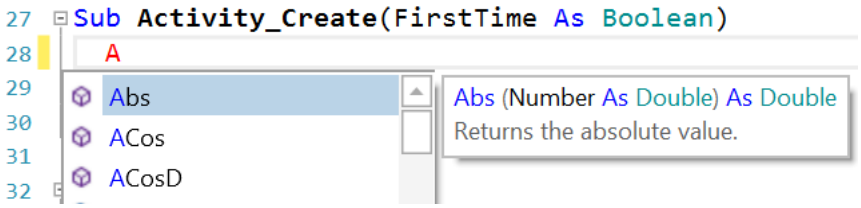
```
Sub btnAction_Click

End Sub
```

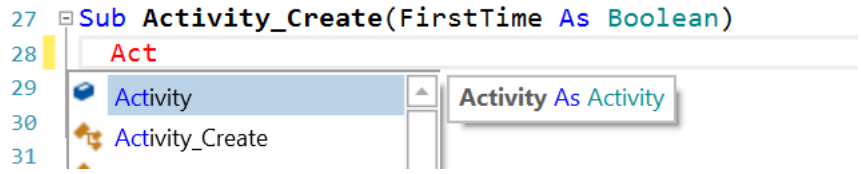
Click on Generate Members to generate the references and Sub frames.

Now we go back to the IDE to enter the code.
First, we need our Activity to load our layout file. Within the “Activity_Create” sub, do the following. You can remove the lines in green.
We will enter this line of code `Activity.LoadLayout(“Main”)`.

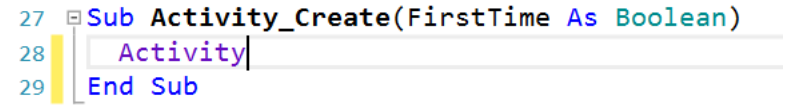
- Enter 'A’, as soon as you begin typing the autocomplete function shows you all keywords beginning with ‘a’.



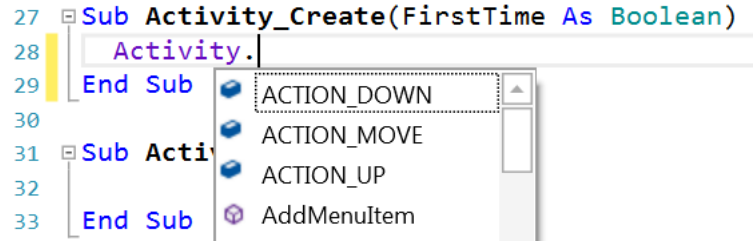
- Continue typing ‘Act’.



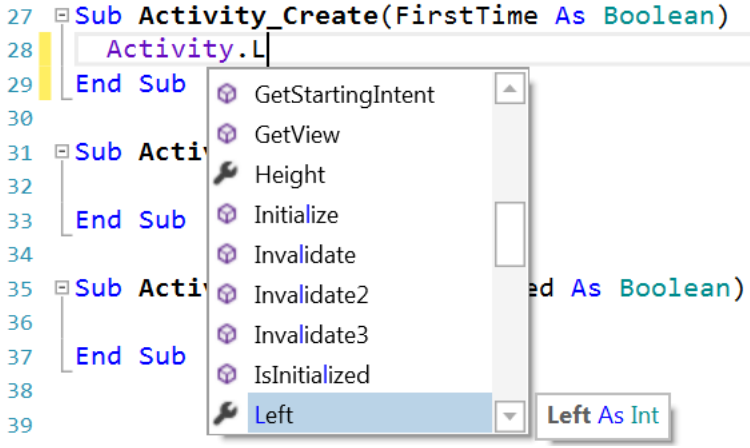
- Press ‘Return’ or click on Activity.



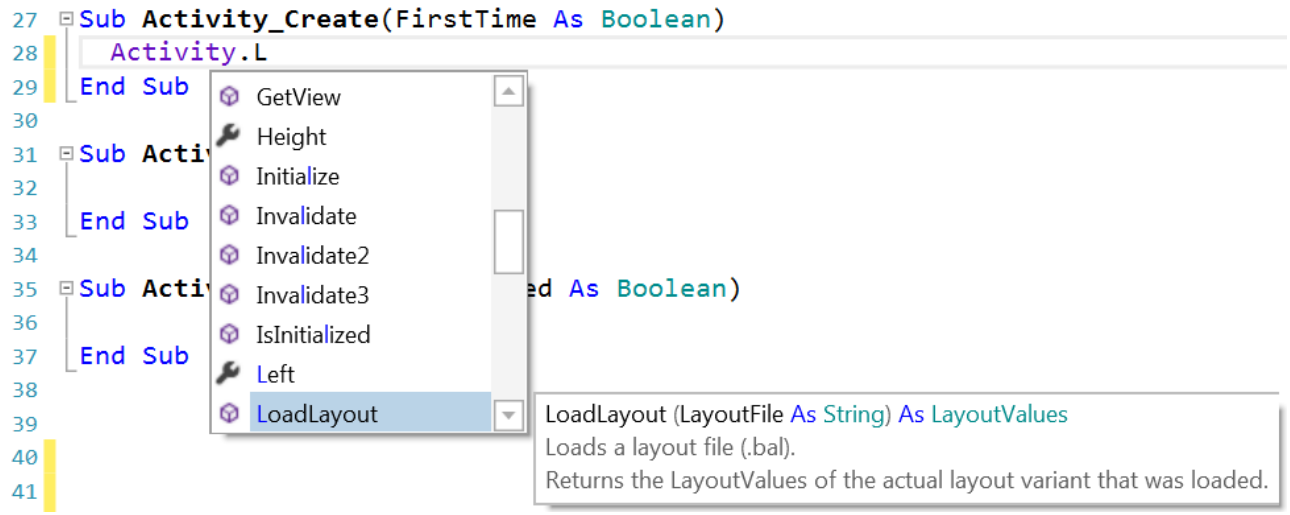
- We have the word Activity, now enter a dot.



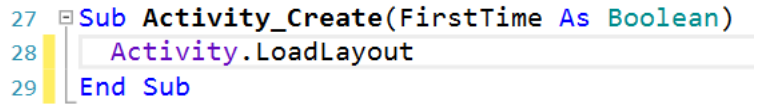
- The autocomplete function shows all the possible properties of the view.
- Enter 'L' , and the autocomplete function shows the properties beginning with 'L'



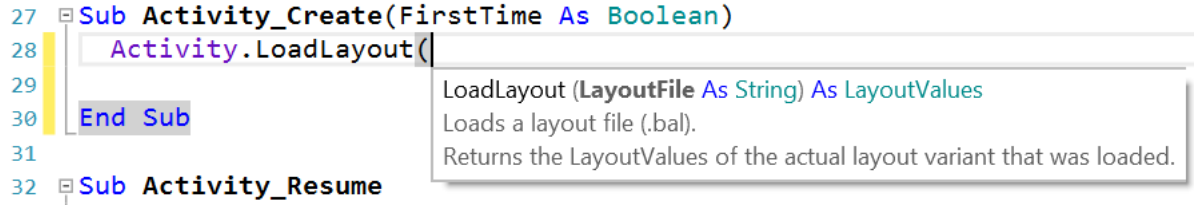
- Press the down arrow key, and LoadLayout will be highlighted with the online help for the given property or method.



- Press 'Return' to add LoadLayout.



- Press '(' to display the online help showing the needed properties for the method.



- Enter "Main")

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

We want to generate a new problem as soon as the program starts. Therefore, we add a call to the New subroutine.

```
Sub Activity_Create(FirstTime As Boolean)
   Activity.LoadLayout("Main")
   New
End Sub
```

Generating a new problem means generating two new random values between 1 and 9 (inclusive) for Number1 and Number2, then showing the values using the lblNumber1 and lblNumber2 ‘Text’ properties.

To do this we enter following code:
In Sub Globals we add two variables for the two numbers.

```
   Public Number1, Number2 As Int
End Sub
```

And the 'New' Subroutine:

```
Sub New
   Number1 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1           ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2           ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   edtResult.Text = ""                 ' Sets edtResult.Text to empty
End Sub
```

The following line of code generates a random number from '1' (inclusive) to '10' (exclusive):
`Rnd(1, 10)`
The following line displays the comment in the `lblComments` view:
`lblComments.Text = "Enter the result" & CRLF & "and click on OK"`
`CRLF` is the LineFeed character.

Now we add the code for the Button click event.

We have two cases:
- When the Button text is equal to "O K" (with a space between O and K), it means that a new problem is displayed, and the program is waiting for the user to enter a result and press the Button.
- When the Button text is equal to "NEW", it means that the user has entered a correct answer and when the user clicks on the Button a new problem will be generated.

```
Sub btnAction_Click
   If btnAction.Text = "O K" Then
      If edtResult.Text = "" Then
         Msgbox("No result entered","E R R O R")
      Else
         CheckResult
      End If
   Else
      New
      btnAction.Text = "O K"
   End If
End Sub
```

`If btnAction.Text = "O K" Then` checks if the Button text equals "O K"
If yes then we check if the EditText is empty.
If yes, we display a MessageBox telling the user that there is no result in the EditText view.
If no, we check if the result is correct or if it is false.
If no then we generate a new problem, set the Button text to "O K" and clear the EditText view.

The last routine checks the result.

```
Sub CheckResult
   If edtResult.Text = Number1 + Number2 Then
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      btnAction.Text = "N E W"
   Else
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
   End If
End Sub
```

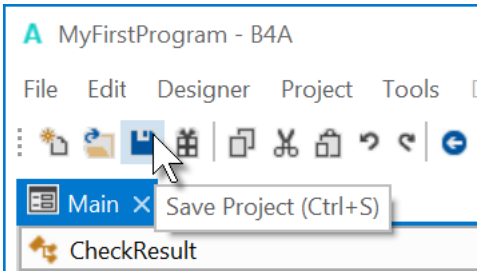
With `If edtResult.Text = Number1 + Number2 Then` we check if the entered result is correct.

If yes, we display in the lblComments label the text below:
'G O O D result'
'Click on NEW'
and we change the Button text to "N E W ".
If no, we display in the lblComments label the text below:
W R O N G result
Enter a new result
and click OK

On the left side of the editor you see a yellow line.
This means that the code was modified.

```
63 Sub CheckResult
64    If edtResult.
65      lblComments.
66      btnAction.Te
67    Else
68      lblComments.
69    End If
70 End Sub
71
```

If we click on  to save the project the yellow line becomes green showing a modified code but already saved. You can also press Ctrl + S to save the project.

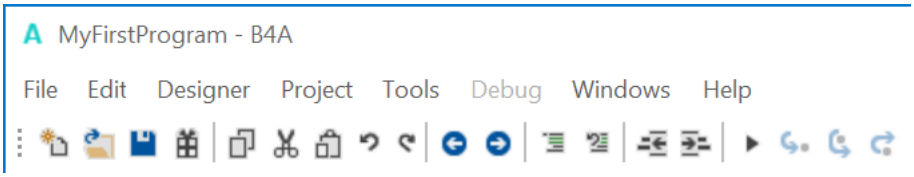



```
63 Sub CheckResult
64    If edtResult.
65      lblComments
66      btnAction.T
67    Else
68      lblComments
69    End If
70 End Sub
```

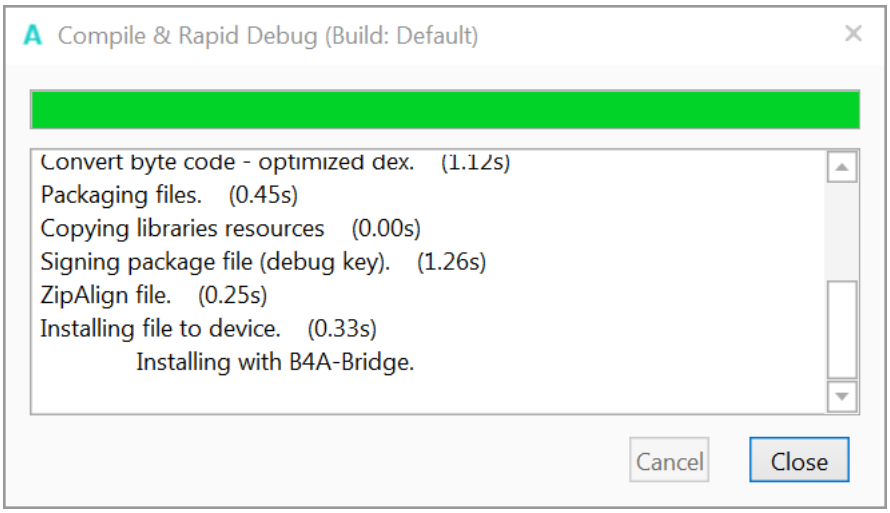
If we leave the IDE and load it again the green line disappears.

Let us now compile the program and transfer it to the Device.

In the IDE on top click on ▶ :

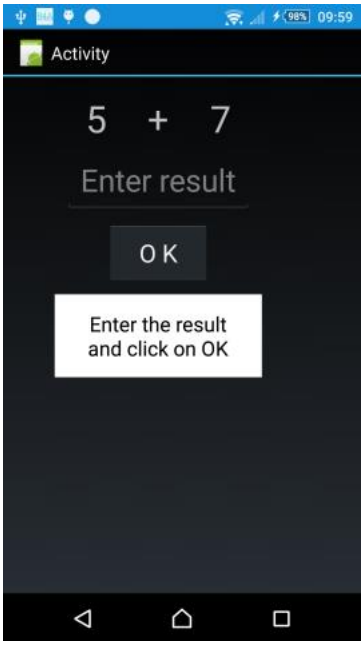


The program is going to be compiled.



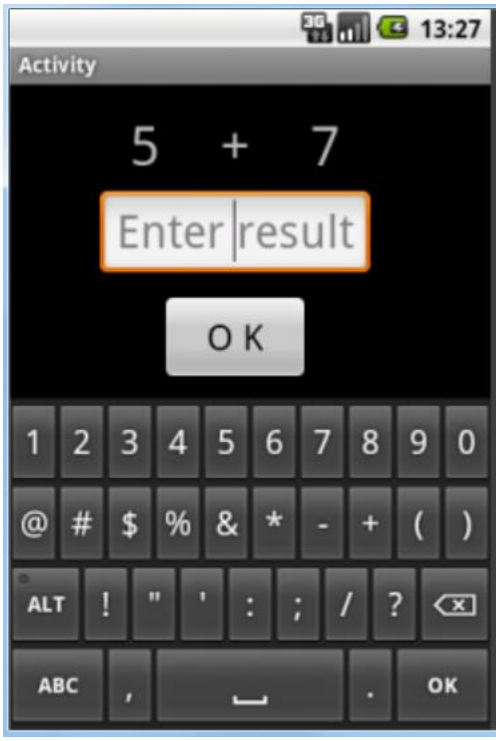
When the Close button becomes enabled as in message box, above, the compiling and transfer is finished.

Looking at the device, you should see something similar to the image below, with different numbers.



The screenshot may look different depending on the device and the Android version.

Of course, we could make aesthetic improvements in the layout, but this was not the main issue for the first program.



On a real device, you need to use the virtual keyboard.
Click on the EditText view to show the keyboard.

On some devices the current layout has the disadvantage that the comment label is covered by the virtual keyboard.

This will be improved in the next chapter,
'Second program', where we create our own keyboard.

2.6 Second B4A program (SecondProgram.b4a)

The project is available in the SourceCode folder:
SourceCode\SecondProgram\B4A\SecondProgram.b4a

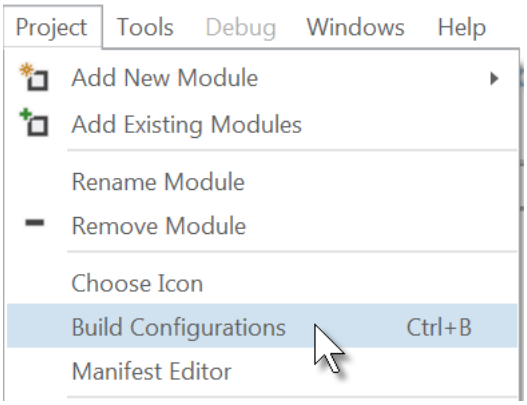
Improvements to " My first program".

We will add a numeric keyboard to the layout to avoid the use of the virtual keyboard.

Create a new folder called "SecondProgram". Copy all the files and folders from MyFirstProgram to the new SecondProgram folder and rename the program files MyFirstProgram.b4a to SecondProgram.b4a and MyFirstProgram.b4a.meta to SecondProgram.b4a.meta.

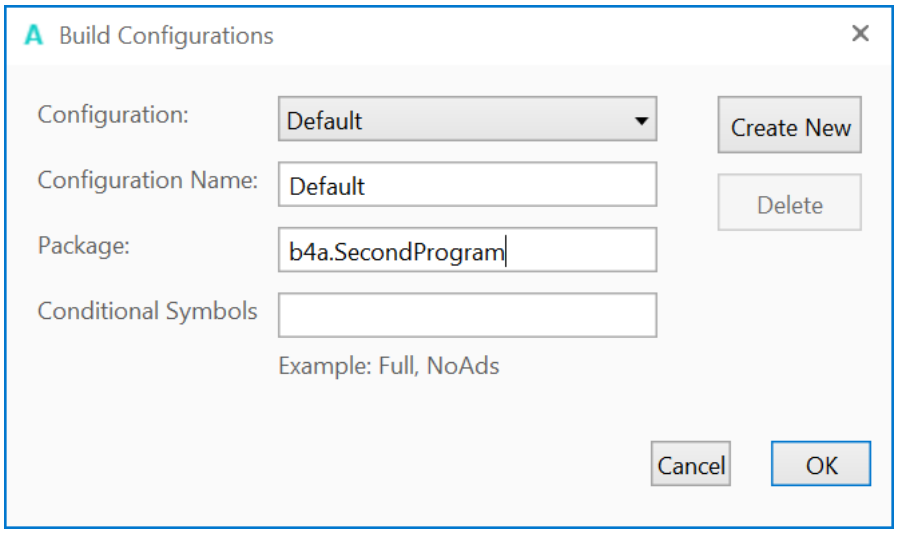
Load this new program in the IDE.

We need to change the Package Name.



In the IDE Project menu.

Click on Build Configurations



Change the Package name to

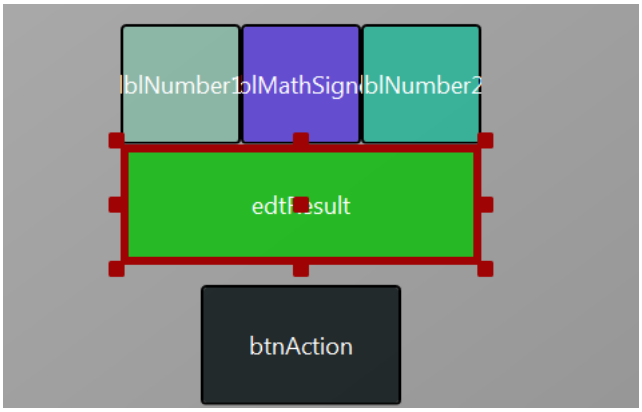
b4a.SecondProgram.

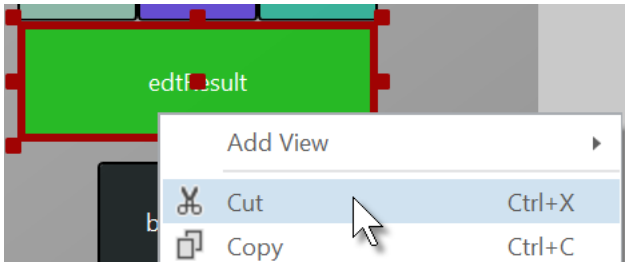
Click on OK.

Then we must change the ApplicationLabel on the very top of the code.

```
#Region  Project Attributes
   #ApplicationLabel: SecondProgram
```

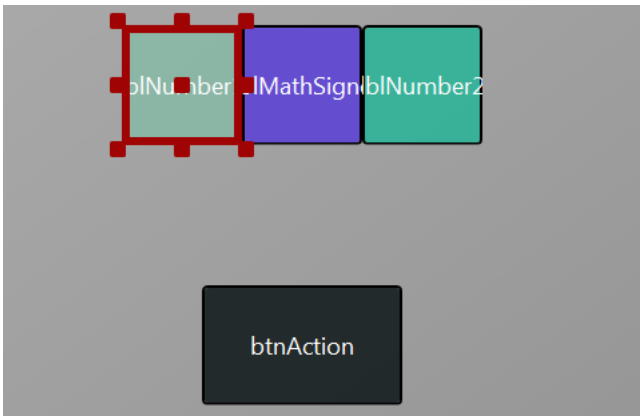
We want to replace the edtResult EditText view by a new Label.
Run the Visual Designer. If you want you can already connect the device or an Emulator.
In the Abstract Designer, click on the edtResult view.



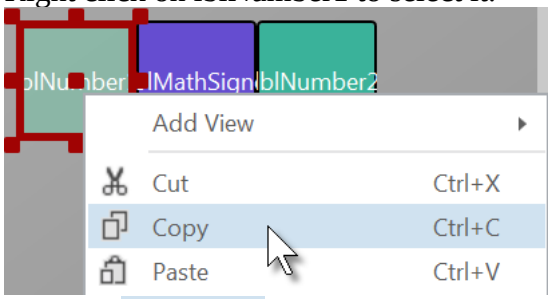


Right click on edtResult and click on 

.

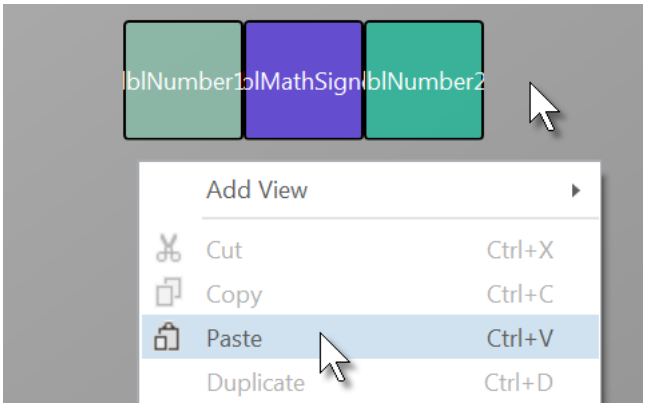


Right click on lblNumber1 to select it.

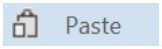


Click on 

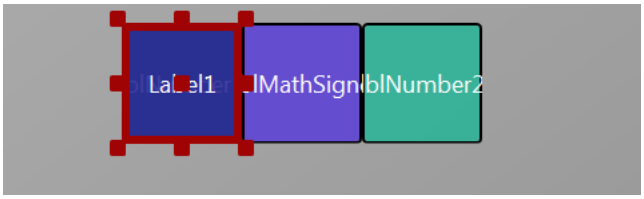
.



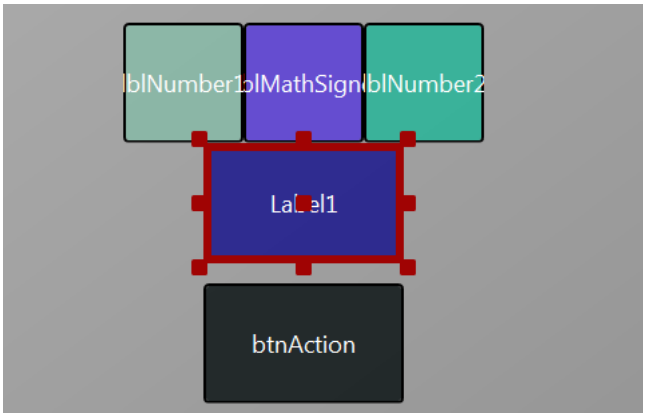
Right click somewhere else outsides a View.

And click on 

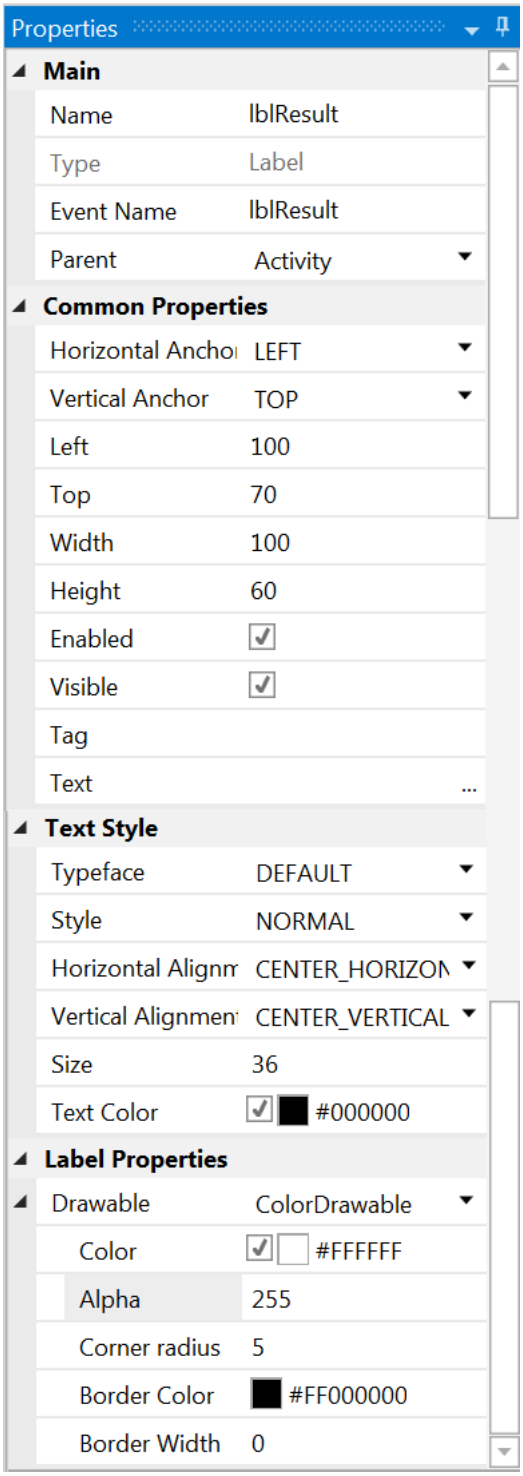
.



The new label covers lblNumber1.



Move it between the upper labels and the button and resize it.



Modify the following properties:

Name to lblResult

Change the Left, Top, Width and Height properties if they are not the same as in the image.

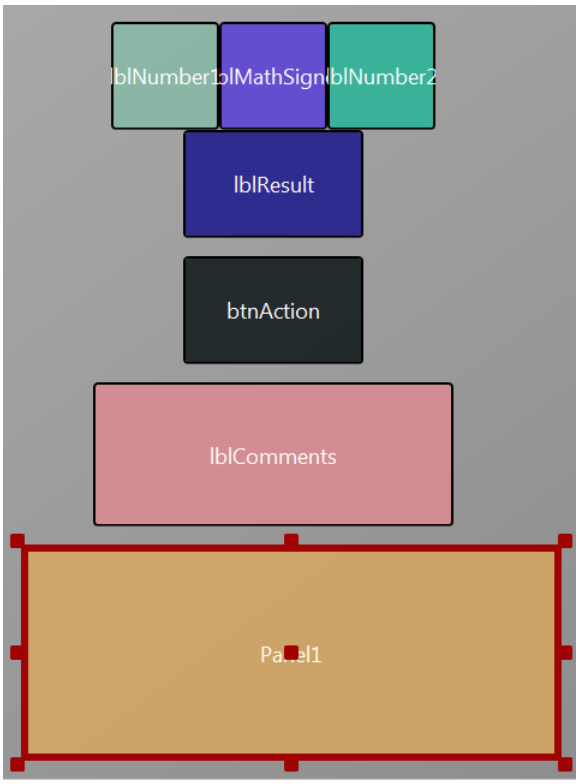
Text to " " blank character

Text Color to Black #000000

Color to White #FFFFFF

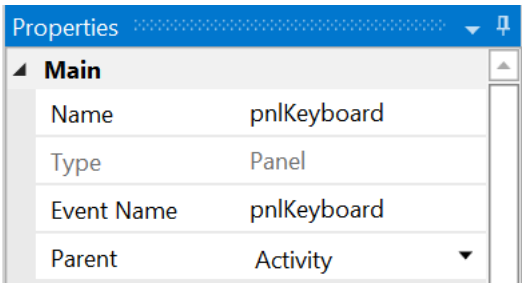
Alpha to 255

Corner Radius to 5

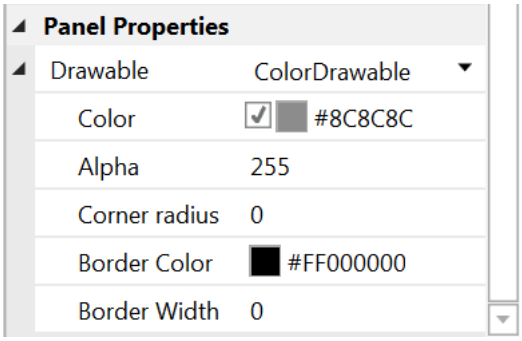


Now we add a Panel for the keyboard buttons.

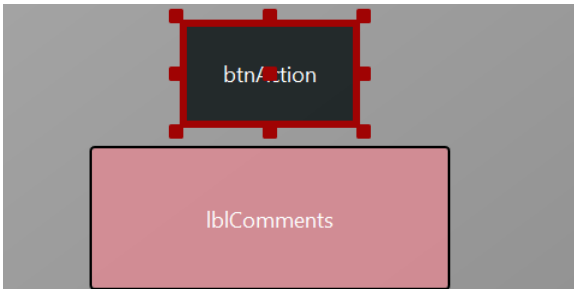
Position and resize it as in the image.



Change its Name to pnlKeyboard
"pnl" for Panel, the view type.

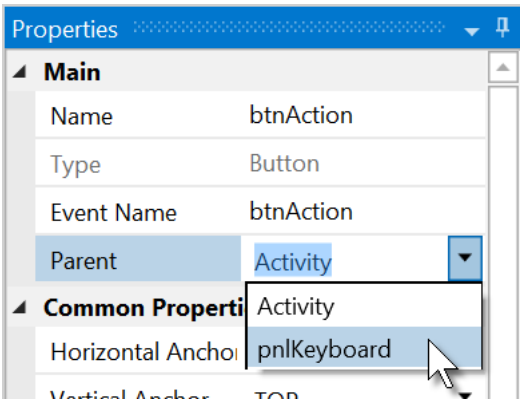


Change
Color to #8C8C8C
Corner radius to 0

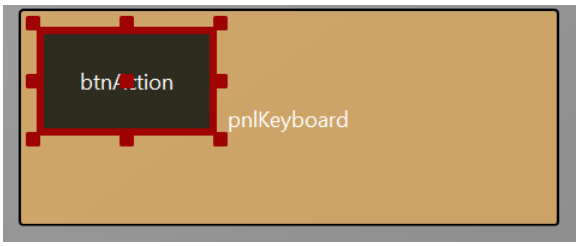


We will move the btnAction button from the Activity to the pnlKeyboard Panel.

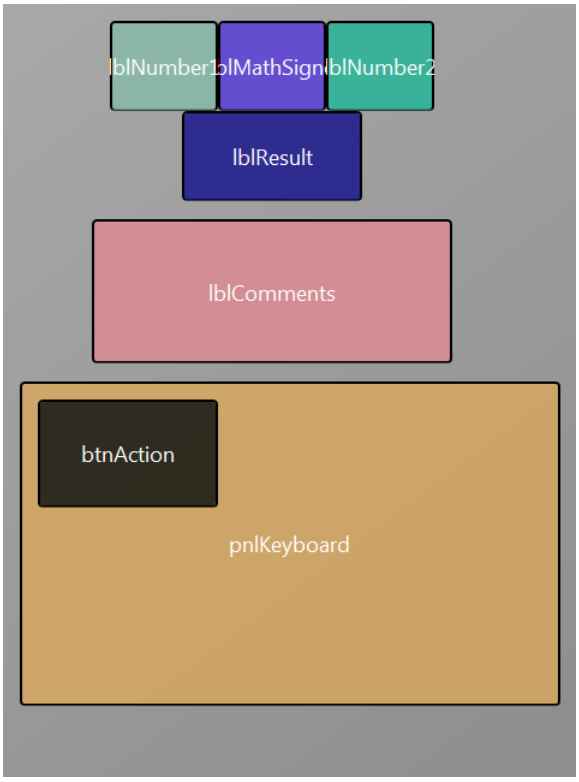
Click on btnAction.



and in the Parent list click on pnlKeyboard.

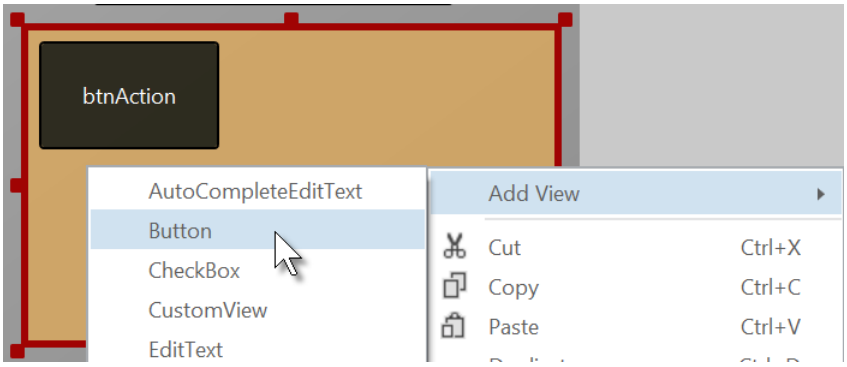


The button now belongs to the Panel.



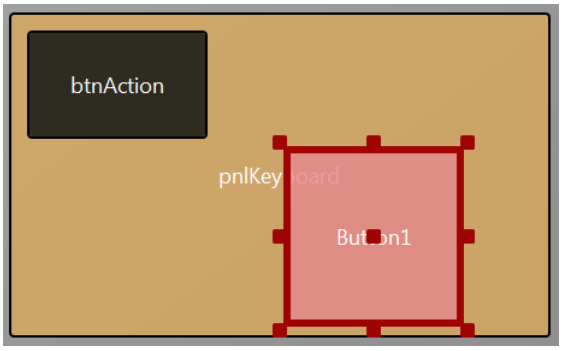
Now we rearrange the views to get some more space for the keyboard.

Set the Height property of the 4 Labels to 50 instead of 60.
Set the Top property of label lblResult to 60.
Set the Top property of label lblComments to 120.
Set the Top property of panel pnlKeyboard to 210.
Set the Height property of panel pnlKeyboard to 180.

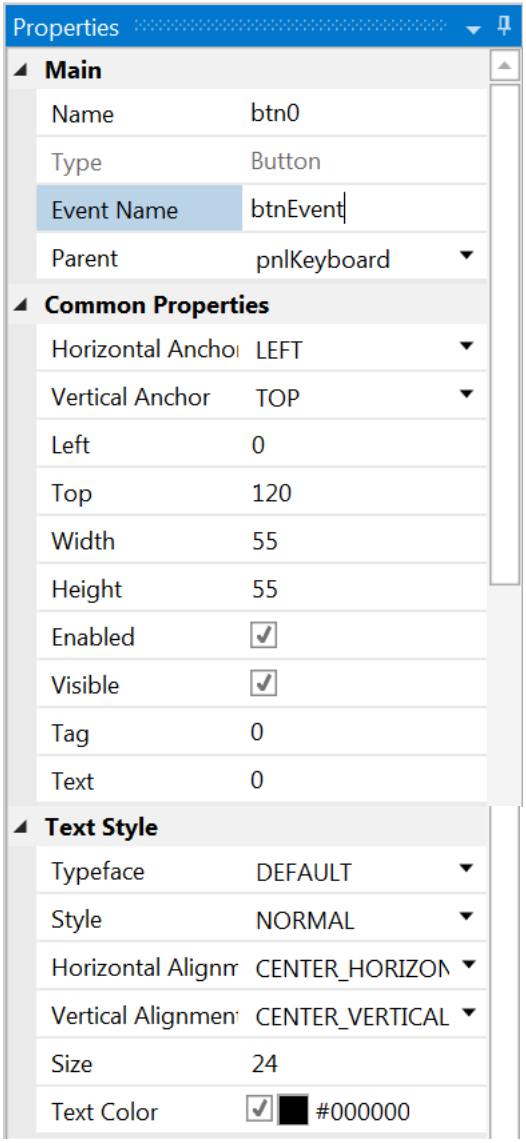


Right click on the pnlKeyboard and click on Add View
And click on Button.

to add a new button.



The new button is added.



Change the following properties:

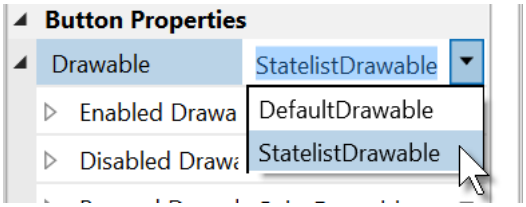
Name to btn0

Event name to btnEvent

Left to 0
Top to 120
Width to 55
Height to 55

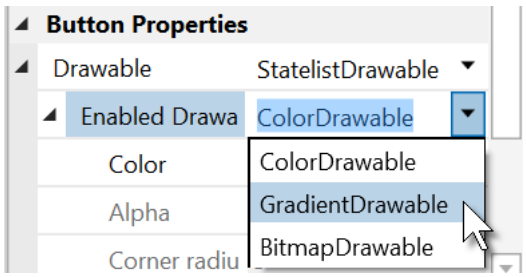
Tag to 0
Text to 0

Size to 24
TextColor to Black #000000

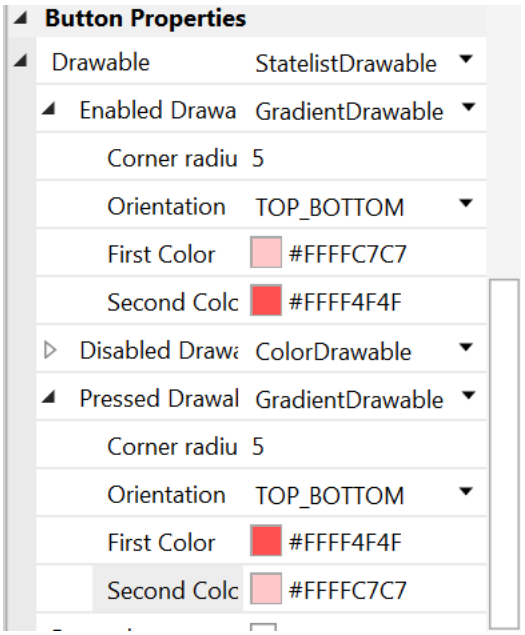


Now we want to change the button colors.

Click on StatelistDrawable.



In Enabled Drawable
click on GradientDrawable.



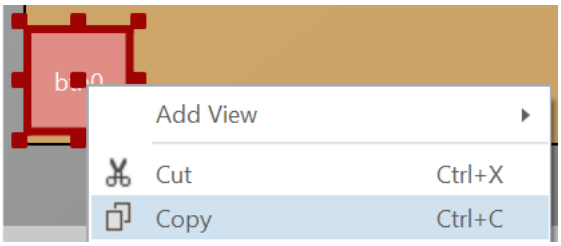
Change the following properties:

Orientation	to	TOP_BOTTOM
First Color		
Second Color		

Pressed Drawable	to	GradientDrawable

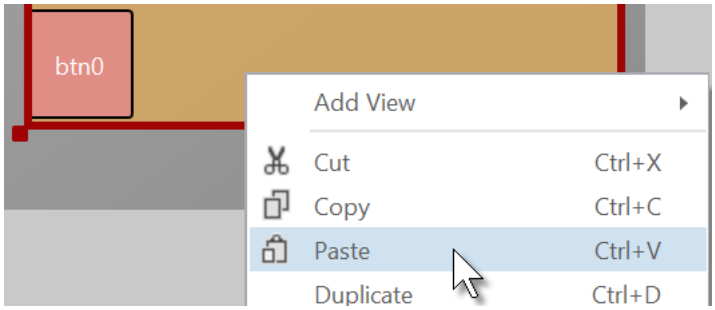
Orientation	to	TOP_BOTTOM
First Color		
Second Color		

If you have connected a device the button looks now like this.



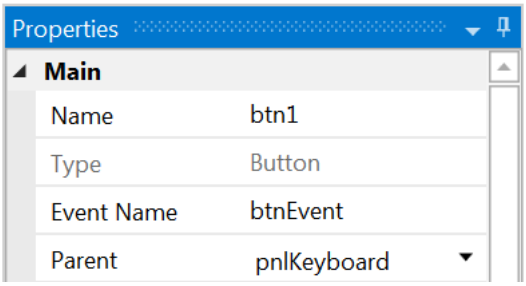
Now we duplicate btn0 and position the new one beside button btn0 with a small space.

Right click on btn0 and click on Copy.



Click on the pnlKeyboard view and click on Paste.

Move the new Button next to the previous one.



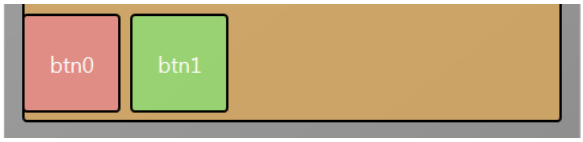
Change the following properties:

Name to btn1

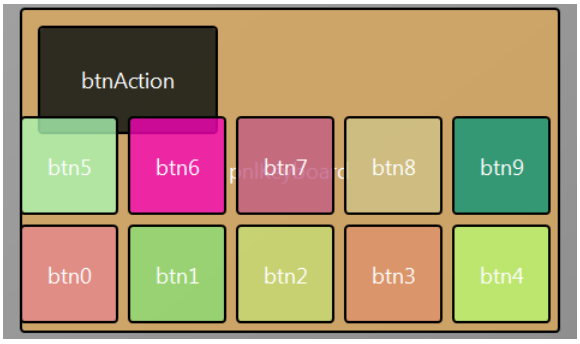
Tag to 1
Text to 1

And the result.

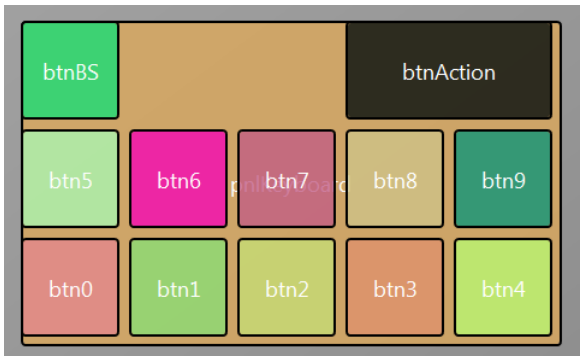
In the Abstract Designer and on the device.



Let us add 8 more Buttons and position them like in the image.



Change following properties:
Name btn2 , btn3 , btn4 etc.
Tag 2 , 3 , 4 etc.
Text 2 , 3 , 4 etc.



To create the BackSpace button, duplicate one of the number buttons, and position it like in the image.

Resize and position btnAction.

Change the pnlKeyboard Color to Black #000000.

Change their Name, Tag, Text and Color properties as below.

btnBS <

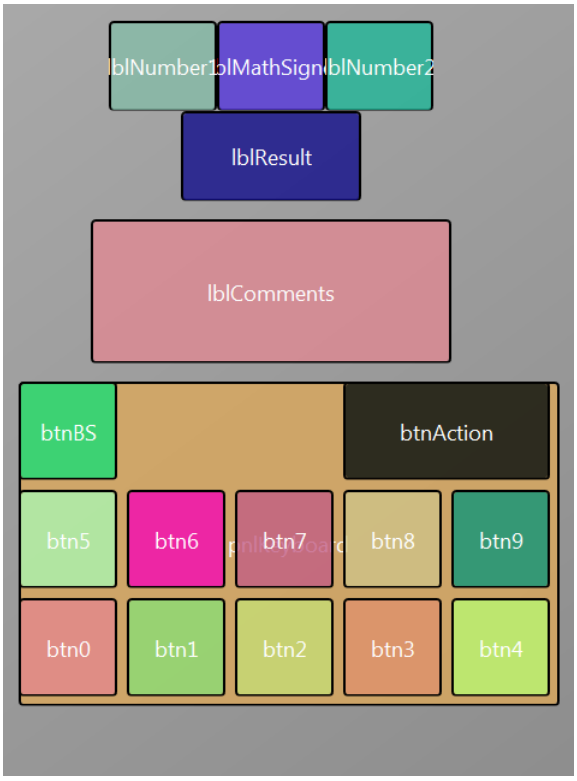
Properties	
Main	
Name	btnAction
Type	Button
Event Name	btnAction
Parent	pnlKeyboard
Common Properties	
Horizontal Ancho	LEFT
Vertical Anchor	TOP
Left	180
Top	0
Width	115
Height	55
Enabled	☑
Visible	☑
Tag	
Text	O K
Button Properties	
Drawable	StatelistDrawable
Enabled Drawa	GradientDrawable
Corner radiu	5
Orientation	TOP_BOTTOM
First Color	#FF8EFF8E
Second Colc	#FF0A800A
Disabled Drawa	ColorDrawable
Pressed Drawal	GradientDrawable
Corner radiu	5
Orientation	TOP_BOTTOM
First Color	#FF0A800A
Second Colc	#FF8EFF8E

btnAction O K

Properties	
Main	
Name	btnBS
Type	Button
Event Name	btnEvent
Parent	pnlKeyboard
Common Properties	
Horizontal Ancho	LEFT
Vertical Anchor	TOP
Left	0
Top	0
Width	55
Height	55
Enabled	☑
Visible	☑
Tag	BS
Text	<
Button Properties	
Drawable	StatelistDrawable
Enabled Drawa	GradientDrawable
Corner radiu	5
Orientation	TOP_BOTTOM
First Color	#FFC7C7FF
Second Colc	#FF4F4FFF
Disabled Drawa	ColorDrawable
Pressed Drawal	GradientDrawable
Corner radiu	5
Orientation	TOP_BOTTOM
First Color	#FF4F4FFF
Second Colc	#FFC7C7FF

Set the Color property of panel pnlKeyboard to Black.

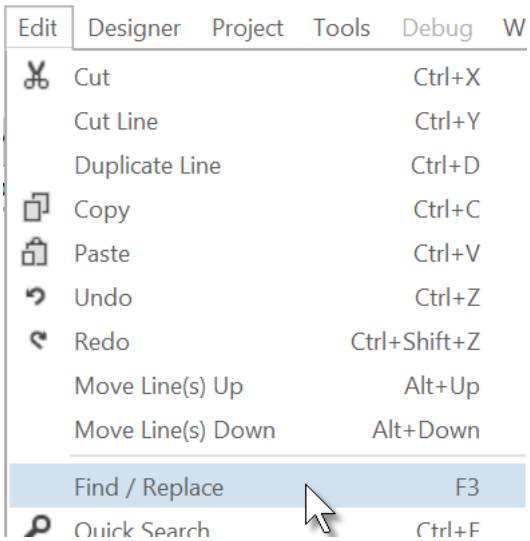
The finished new layout.
In the Abstract Designer and on the device.



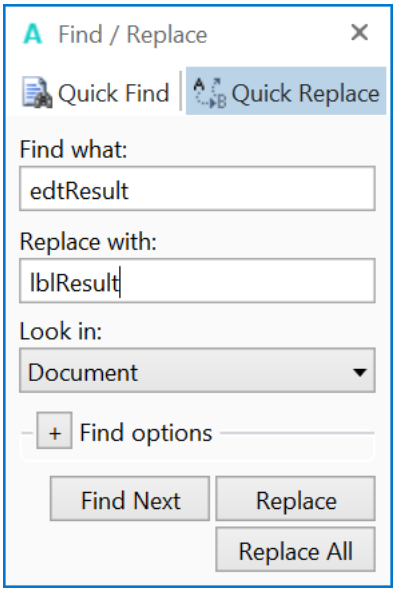
Now we will update the code.
First, we must replace the edtResult by lblResult because we replaced the EditText view by a Label.

```
19 Sub Globals
20   Private btnAction As Button
21   Private edtResult As EditText
22   Private lblComments As Label
23   Private lblMathSigne As Label
```

Double click on edtResult to select it.



In the Edit menu click on Find / Replace or press F3.



Enter 'lblResult' in the Replace with field.

Click on Replace All

We also need to change its view type form EditText to Label.

```
Private lblResult As Label
```

Now we write the routine that handles the Click events of the Buttons.
The Event Name for all buttons, except btnAction, is "btnEvent".
The routine name for the associated click event will be btnEvent_Click.
Enter the following code:

```
Sub btnEvent_Click

End Sub
```

We need to know what button raised the event. For this, we use the Sender object which is a special object that holds the object reference of the view that generated the event in the event routine.

```
Sub btnEvent_Click
   Private btnSender As Button
   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
   Case Else
   End Select
End Sub
```

To have access to the properties of the view that raised the event we declare a local variable `Private btnSender As Button`.

And set `btnSender = Sender`.

Then, to differentiate between the backspace button and the numeric buttons we use a Select / Case / End Select structure and use the Tag property of the buttons.
Remember, when we added the different buttons we set their Tag property to BS, 0, 1, 2 etc.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
```

sets the variable to test.
checks if it is the button with the "BS" tag value.
handles all the other buttons.

Now we add the code for the numeric buttons.
We want to add the value of the button to the text in the lblResult Label.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

This is done in this line

```
   lblResult.Text = lblResult.Text & btnSender.Text
```

The "&" character means concatenation, so we just append to the already existing text the value of the Text property of the button that raised the event.

Now we add the code for the BackSpace button.

```
   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0, lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

When clicking on the BS button we must remove the last character from the existing text in lblResult.
However, this is only valid if the length of the text is bigger than 0. This is checked with:

```
If lblResult.Text.Length >0 Then
```

To remove the last character we use the SubString2 function.

```
lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
```

SubString2(BeginIndex, EndIndex) extracts a new string beginning at BeginIndex (inclusive) until EndIndex (exclusive).

Now the whole routine is finished.

```
Sub btnEvent_Click
   Private btnSender As Button

   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

We can try to improve the user interface of the program by adding some colors to the lblComments Label.
Let us set:
- Yellow for a new problem
- Light Green for a GOOD answer
- Light Red for a WRONG answer.

Let us first modify the New routine, where we add the line `lblResult.Text = ""`.

```
Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   lblComments.Color = Colors.RGB(255,235,128)      ' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
End Sub
```

And in the CheckResult routine we add lines 76 and 80.

```
Sub CheckResult
   If lblResult.Text = Number1 + Number2 Then
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      lblComments.Color = Colors.RGB(128,255,128)       ' light green color
      btnAction.Text = "N E W"
   Else
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
      lblComments.Color = Colors.RGB(255,128,128)       ' light red color
   End If
End Sub
```

Another improvement would be to hide the '0' button to avoid entering a leading '0'.
For this, we hide the button in the New subroutine in line `btn0.Visible = False`.

```
Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   lblComments.Color = Colors.RGB(255,235,128)      ' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
   btn0.Visible = False
End Sub
```

In addition, in the btnEvent_Click subroutine, we hide the button if the length of the text in lblResult is equal to zero and show it if the length is greater than zero, lines 98 to 102.

```
Sub btnEvent_Click
   Private btnSender As Button

   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Tag
   End Select

   If lblResult.Text.Length = 0 Then
      btn0.Visible = False
   Else
      btn0.Visible = True
   End If
End Sub
```

As we are accessing btn0 in the code we need to declare it in the Globals routine.

Modify line 25 like below:

```
   Private btnAction, btn0 As Button
```

Run the program to check the result.

3 Getting started B4i

B4i is a simple yet powerful development environment that targets Apple devices (iPhone, iPad etc.).
The B4i language is similar to Visual Basic and B4A language.
B4i compiled applications are native iOS applications; there are no extra runtimes or dependencies.
Unlike other IDE's, B4i is 100% focused on iOS.
B4i includes a powerful GUI designer, built-in support for multiple screens and orientations.
You can develop and debug with a real device.

iOS 7 and above are supported.

What you need:

- The B4i program, this is a Windows program running on a PC.
- The Java SDK on the PC, free
- An Apple developer license, cost 99$ per year.
- A device for testing.
- The Basi4i-Bridge program on the device, free
- A Mac builder to compile the program, this be either
    - A Mac computer with the Mac Builder program, on local wifi.
    - The hosted Mac Builder service over Internet, cost 26$ per year
- A Mac computer or a MacInCloud service to distribute the program

Links to tutorials in the forum:
Local Mac Builder Installation
Creating a certificate and provisioning profile
Installing B4i-Bridge and debugging first app

3.1 Installing B4i

3.1.1 Installing Java JDK

B4i depends on the free Java JDK component

If you are already using B4A you can skip this chapter.

Installation instructions:

The first step should be to install the **Java JDK**, as B4i requires it.
Note that there is no problem with having several versions of Java installed on the same computer.

- Open the Java 8 JDK download link.
- Check the Accept License Agreement radio button.
- Select **"Windows x86"** or **"Windows x64"** (for 64 bit machines) in the platforms list.
- Download the file and install it.

3.1.2 Installing B4i

Download and install the B4i file on your computer.

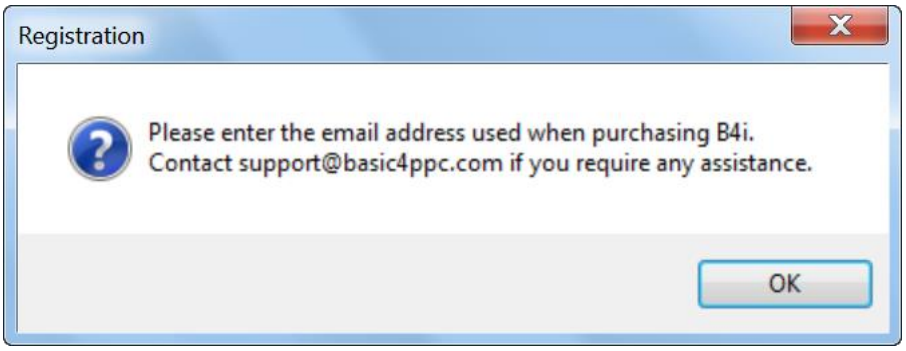
iOS compilation requires an Apple Mac computer. Developers have two options with B4i:

- Use a local Mac machine connected over the local network.
  For this you should also download the Mac builder and install it.

- Use our hosted builder rental service. Hosted Mac Builder installation.
  The builder service allows you to develop iOS applications without a Mac computer.
  All of the development steps can be done with the builder service except of the final step which is uploading the application to Apple App Store. This step requires a Mac or a service such as MacInCloud.
  Note that the builder is currently limited to projects of up to 15mb.

Copy the license file *b4i-license.txt* to the B4i folder and to a safe place on the computer for backup.
Note that this is not a text file, do not open it with a text editor.

When you first run B4i you will be asked to enter your e-mail address, the one you used when you purchased it B4i.
You find it also in the mail you received with the B4i file.



Enter the e-mail address.

You get a confirmation window that B4i is registered.

Contact support@basic4ppc.com if you require any assistance.

3.1.3 Mac Builder installation

iOS compilation requires an Apple Mac computer. Developers have two options with B4i:
- Use a local Mac machine connected over the local network.
- Use our hosted builder rental service.

Link to the tutorial in the forum: Local Mac Builder Installation.

These instructions explain how to install the builder on a local Mac machine.

1. Install Java JDK 8: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Install Xcode 6.
3. Download and unzip the B4i-Builder.
4. Open a terminal and navigate to B4i-Builder folder.
5. Run it with: java -jar B4iBuildServer.jar
6. Set the builder IP address in the IDE under Tools - Build Server - Server Settings

Notes & Tips

- By default ports numbers 51041 (http) and 51042 (https) are used.
- The firewall should be either disabled or allow incoming connections on these two ports.
- You can test that the server is running by going to the following link: http://<server ip>:51041/test
- You can kill the server with: http://<server ip>:51041/kill
- It is recommended to set your Mac server ip address to a static address. This can be done in your router settings or in the Mac under Network settings.
- A single Mac builder can serve multiple developers as long as they are all connected to the same local network. Note that you are not allowed to host builders for developers outside of your organization.

Multiple IPs.

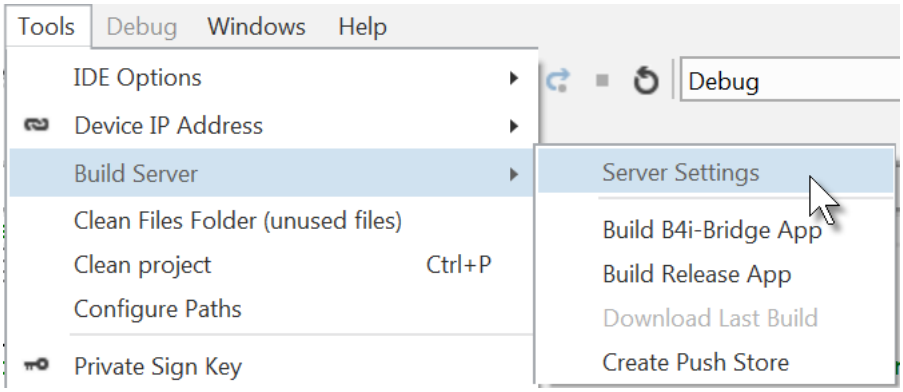
When the server is started it takes the first IP address reported by the OS and uses it as its own IP address. You can see this address in the server messages.
In most cases this is the correct address. However if it is not the correct IP address then the server will not be usable.
In that case you need to explicitly set the correct address:
- Open the key folder and delete all files.
- Edit key.txt and change it to:

3.1.4 Hosted Mac builder service (optional)

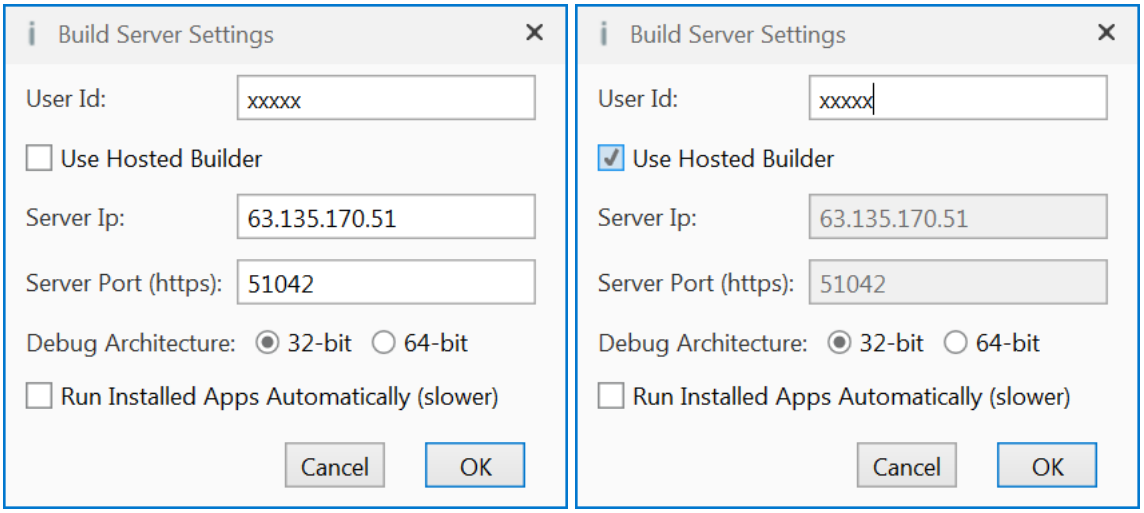
If you have bought the *hosted Mac builder service* you got a mail with your user ID.

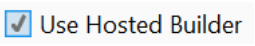
Link to the tutorial in the forum:

You must enter it in the IDE.



Enter the ID.

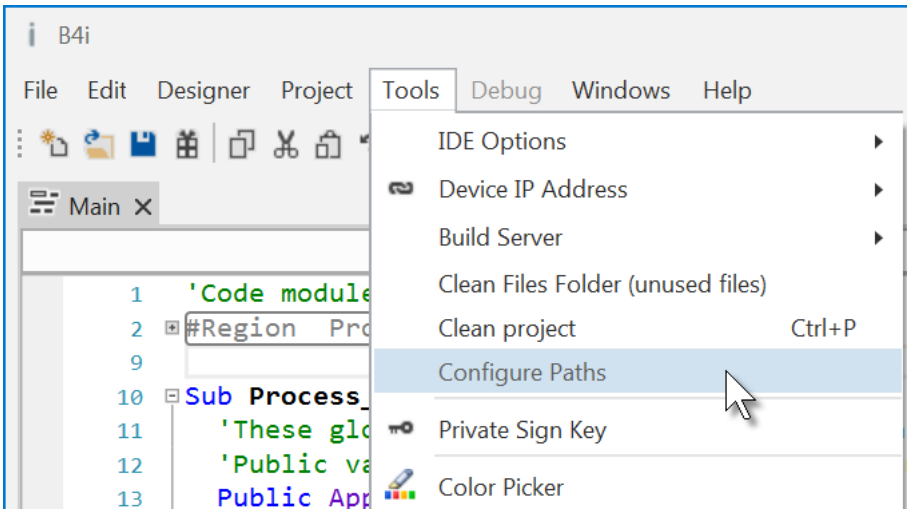


Don't forget to check 

if you use the Hosted Builder Service!

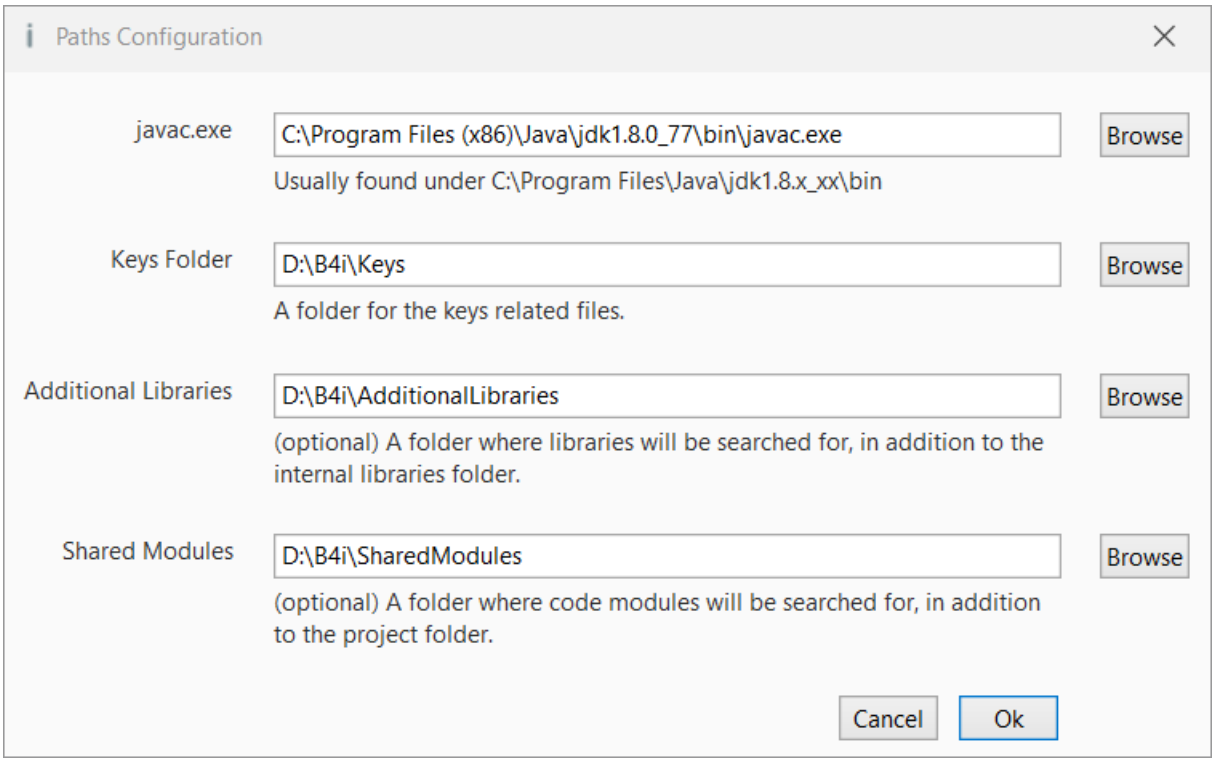
3.2 Configure Paths in the IDE

Then you need to configure the different paths in the IDE.



Run the IDE.

In the Tools menu click on Configure Paths.



javac.exe:
Enter the folder of the javac.exe file.

Keys folder:
Create a special folder for the Keys, for example C:\B4i\Keys.

Additional libraries:
Create s specific folder for additional libraries, for example C:\B4i\AdditionalLibraries.

Shared Modules:
Create s specific folder for shared modules, for example C:\B4i\SharedModules.

3.3 Creating a certificate and provisioning profile

Don't panic!
While this process can be a bit annoying it is not too complicated and you can always delete the keys and start from scratch (which is not always the case in Android).

Note that you must first register with Apple as an iOS developer (costs $99 per year).
The whole process is done on a Windows computer.

In order to install an app on an iOS device you need to create a certificate and a provisioning profile.

The certificate is used to sign the application. The provisioning profile, which is tied to a specific certificate, includes a list of devices that this app can be installed on.

The video shows the steps required for creating and downloading a certificate and provisioning profile.
There are two steps which are not shown in the video and are also required before you can create a provisioning profile:

- Create an App ID. This step is quite simple. Just make sure that you create a wildcard id.
- Add one or more devices. You will need to find the devices UDID for that.

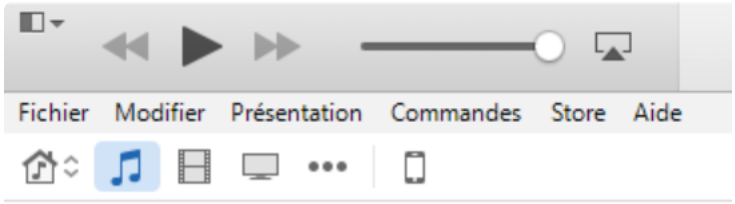
Link to the tutorial in the forum: Creating a certificate and provisioning profile.

3.3.1 UDID

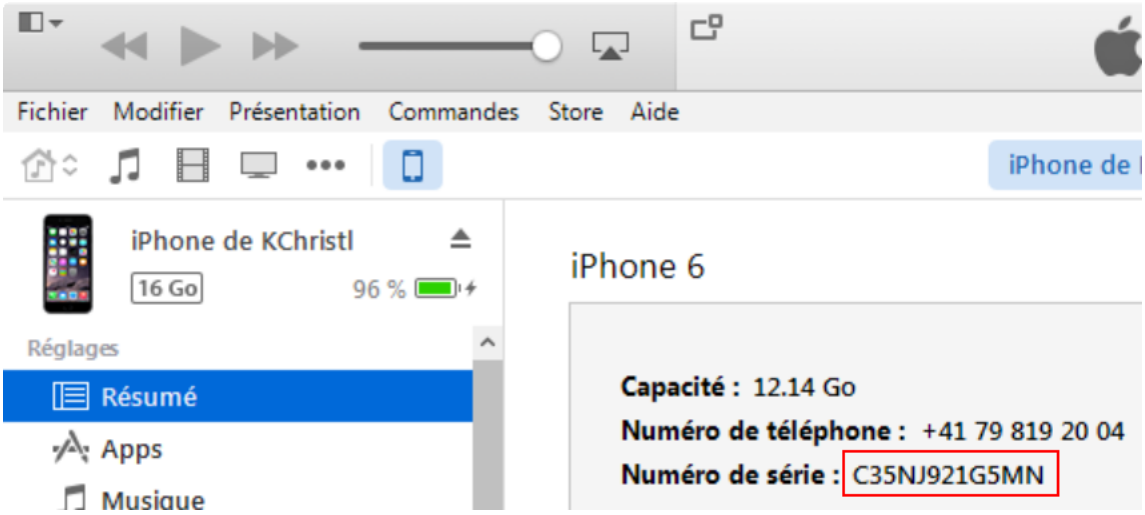
Devices are recognized by their UDIDs. There are two ways to get the device UDID:

1. If iTunes is installed then you can find it in iTunes.
   The first time, connect your device with the USB cable to the computer.

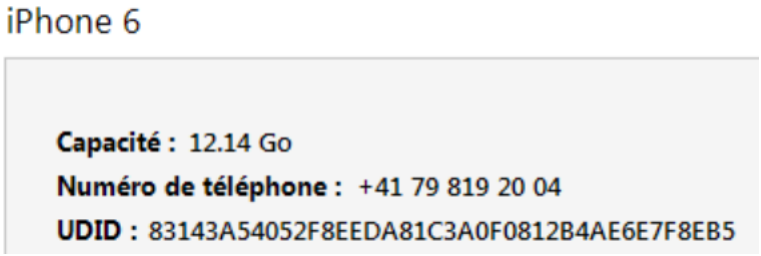
   Run iTunes, you should see on top this icon . It can take a while before you see it.



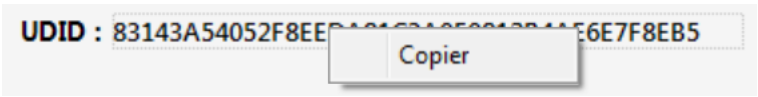
   Click on and you get this screen:



Now click on C35NJ921G5MN to get the UDID.



Right click on 83143A54052F8EEDA81C3A0F0812B4AE6E7F8EB5 to copy the UDID.



2. Use an online service such as this one: http://get.udid.io/

3.3.2 Certificate and Provisioning Profile

Main steps:

1. Set a new keys folder in the IDE.
2. Create a key by choosing Tools - Private Sign Key
3. Create and download the certificate as demonstrated in the video. You will need to upload the CSR file that was created in step 2.

   Note that you can choose either **iOS App Development** or **App Store and Ad Hoc** in the certificate page.
4. Create and download a provisioning profile.

3.4 Installing B4i-Bridge and debugging first app

B4i-Bridge is an application that you install on the device.
It has three purposes:

1. Launch the installation process when needed.
2. Run the installed app (when installation is not needed).
3. The bridge is also the WYSIWYG visual designer.

You need to install B4i-Bridge once. It is done from the device browser.

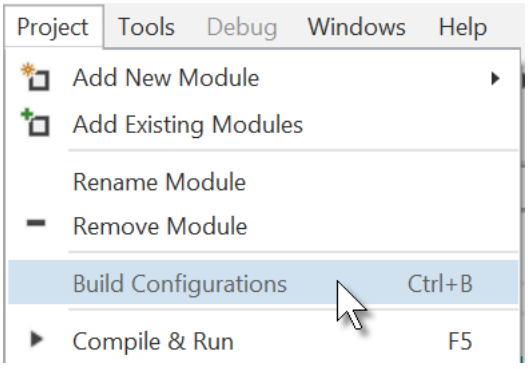
Link to the tutorial in the forum: Installing B4i-Bridge and debugging first app.

3.5 Install the B4I certificate

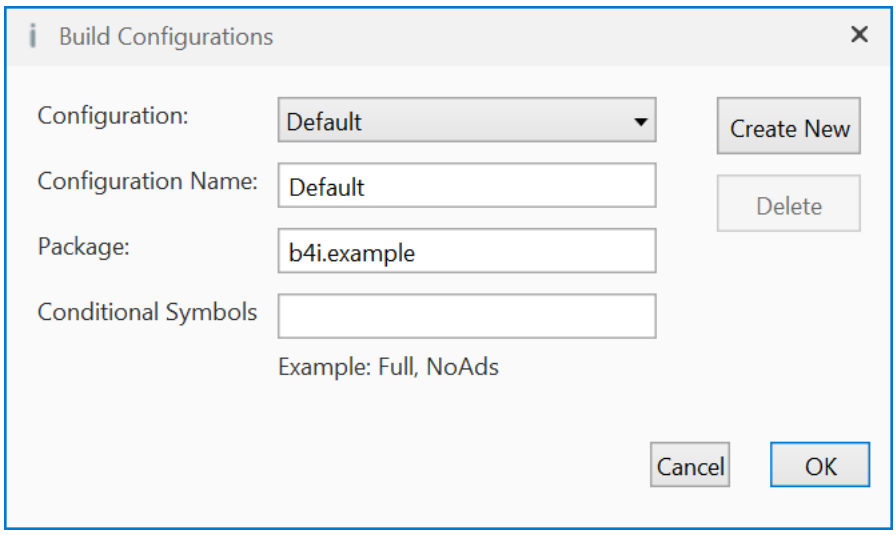
Open Safari (device browser) and navigate to: www.b4x.com/ca.pem
Follow the instructions.
You can see at any time the profile in Settings / General / Profile.

3.6 Set the package name based on the provision app id

Run B4i, load a project or use the default project and set the package name based on the provision app ID.
In the Project menu click on Build Configurations.

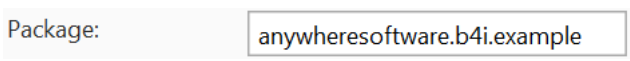


The window below is shown:



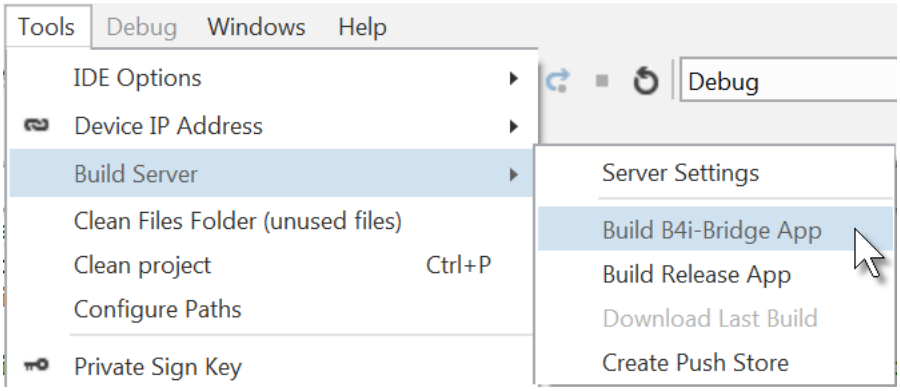
Change the Package name according to the provision app ID.

Example in my case:

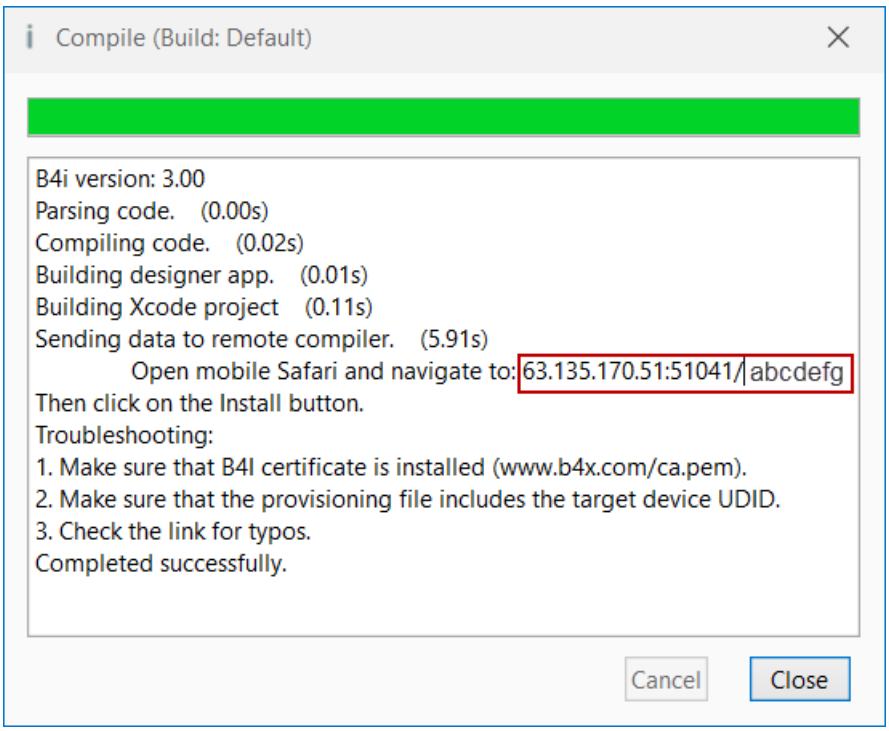


3.6.1 Install Build B4i-Bridge

In the Tools menu click on Build Server and click on Build B4i-Bridge App:



You get the compilation window.



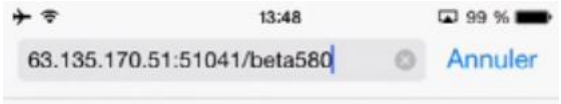
The code you see will be different!

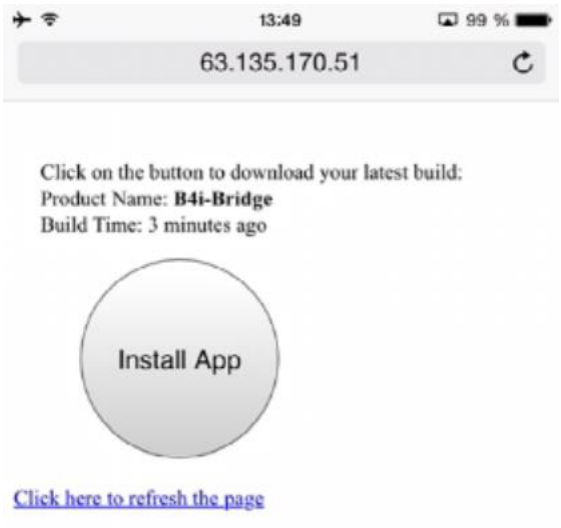
3.6.2 Load B4i-Bridge

Open Safari on the device 

.

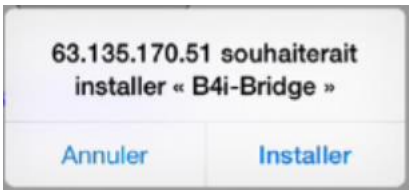
Enter the code in the search box on top.





This screen will appear.

Click on



Close Safari.

3.6.3 Install B4i-Bridge and run it

Click on this B4i-Bridge icon 

on the device, you will see the installing animation and finally the B4i-Bridge icon 

.

Tips:
- You don't need to wait for the installation animation to complete. Once the animation starts you can click on the app icon.
- If the installation is stuck in the "waiting" step for more than 10 or 15 seconds then uninstall it and install it again.
- B4i-Bridge must be in the foreground for it to be able to start an installation or to run the application. In most cases it will be in the foreground automatically. If it is not in the foreground then you need to click on it to bring it to the foreground.

3.7 My first B4i program (MyFirstProgram.b4i)

Let us write our first B4i program. The suggested program is a math trainer for kids.

The project is available in the SourceCode folder shipped with this booklet:
SourceCode\MyFirstProgram\B4i\MyFirstProgram.b4i

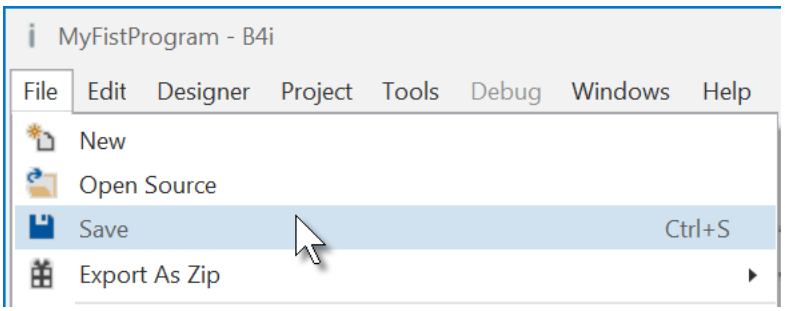
On the screen, we will have:
- 2 Labels displaying randomly generated numbers (between 1 and 9)
- 1 Label with the math sign (+)
- 1 TextField where the user must enter the result
- 1 Button, used to either confirm when the user has finished entering the result or generate a new calculation.
- 1 Label with a comment about the result.

In iOS:
- Label is an object to show text.
- TextField is an object allowing the user to enter text.
- Button is an object allowing user actions.

We will design the layout of the user interface with the Designer, the Abstract Designer and on a Device and go step by step through the whole process.
The Designer manages the different objects of the interface.
The AbstractDesigner shows the positions and sizes of the objects and allows moving or resizing them on the screen.
On the Device we see the real result.

Run the IDE 

Save the project.

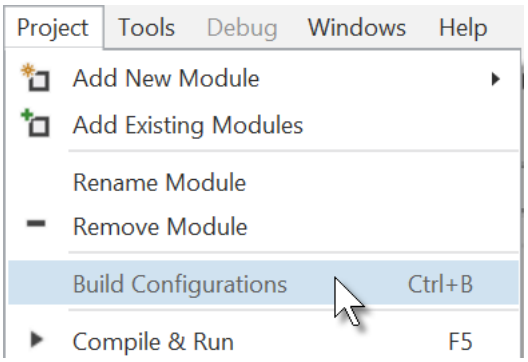


You must save the project before you can run the Designer.

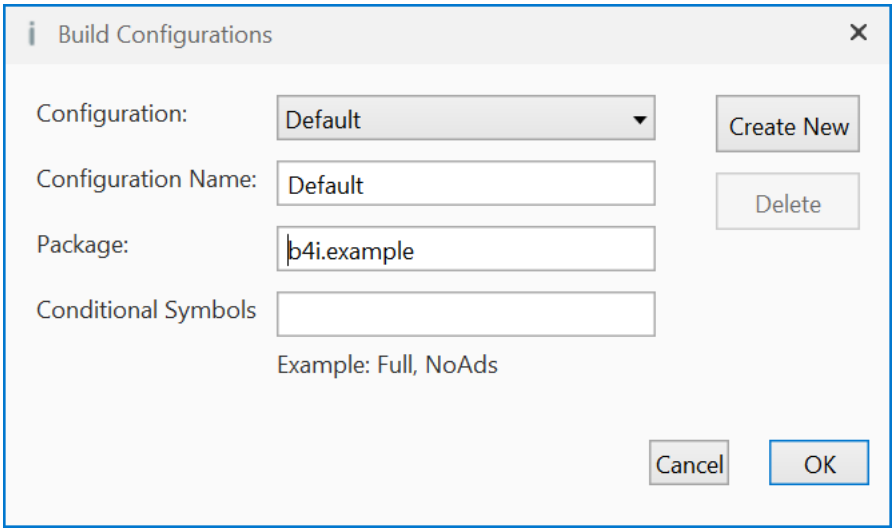
Create a new folder MyFirstProgram and save the project with the name MyFirstProgram.

Set the Package Name.
Each program needs a package name.

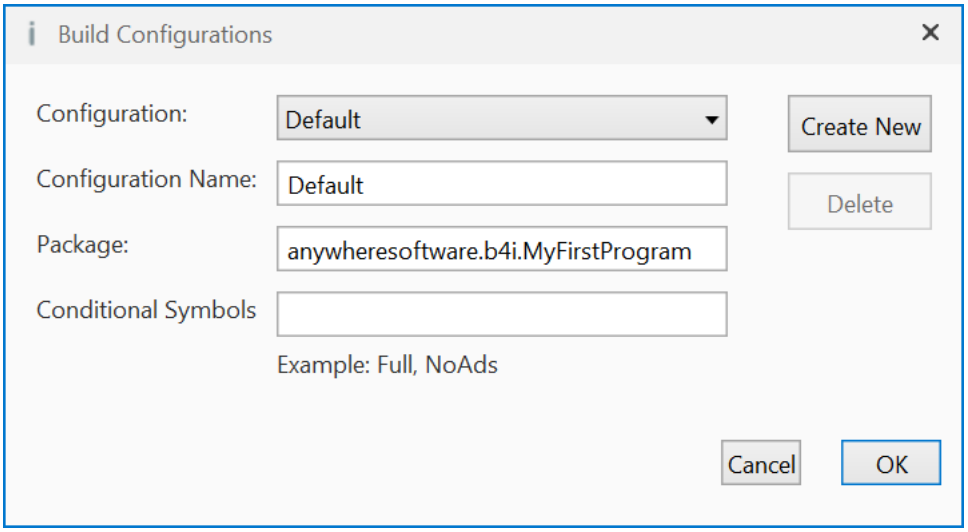
In the menu Project click on Build Configurations.



This window appears:



The default name is b4i.example. We will change it to anywheresoftware.b4i.MyFirstProgram.

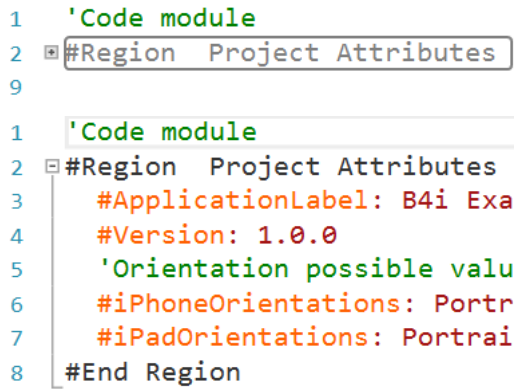


Set the Application Label.

The Application label is the name of the program that will be shown on the device.

On top of the code screen you see the 'region' Project Attributes.

Regions are code parts which can be collapsed



or extended at the right.
Clicking on ⊞ will expand the Region.
Clicking on ⊟ will collapse the Region.
Regions are explained in Collapse a Region.

```
#Region  Project Attributes
	#ApplicationLabel: B4i Example
	#Version: 1.0.0
	'Orientation possible values: Portrait, LandscapeLeft, LandscapeRight and
PortraitUpsideDown
	#iPhoneOrientations: Portrait, LandscapeLeft, LandscapeRight
	#iPadOrientations: Portrait, LandscapeLeft, LandscapeRight, PortraitUpsideDown
#End Region
```

The default name is `B4i Example`, but we will change it to `MyFirstProgram` for naming consistency.

Change this line:

```
  #ApplicationLabel: B4i Example
```

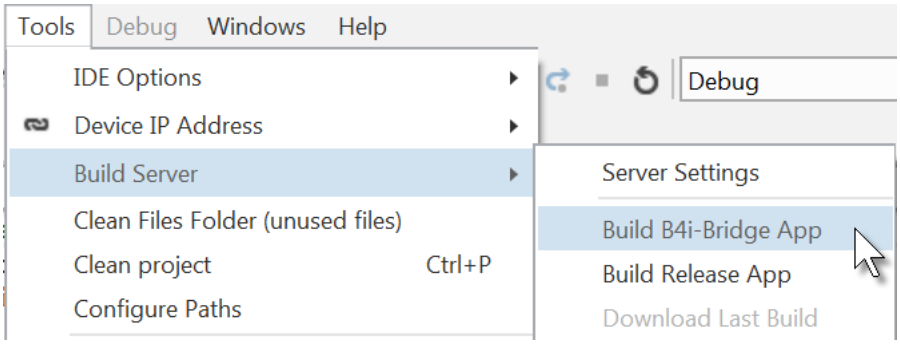
to

```
  #ApplicationLabel: MyFirstProgram
```

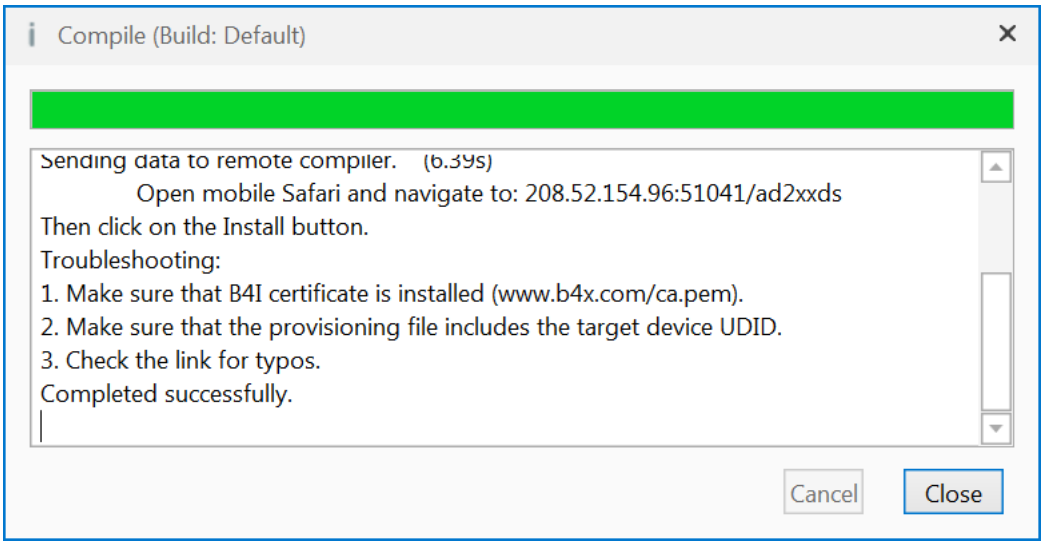
The other lines are explained in Code header Project Attributes / Attributes.

In the IDE run Build B4i-Bridge App.

IDE menu Tools / Build Server / Build B4i-Bridge App



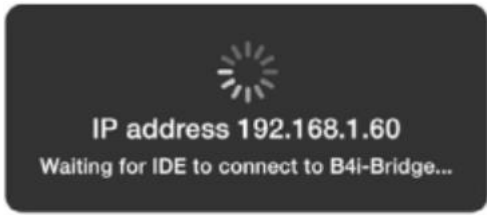
You get this screen.



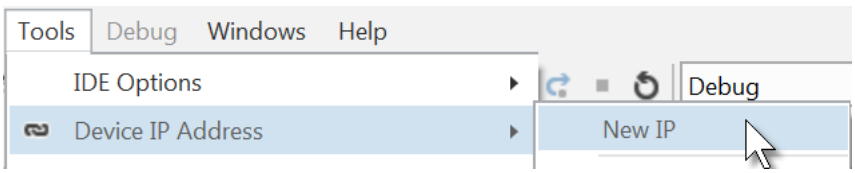
On the device run B4i-Bridge 

.

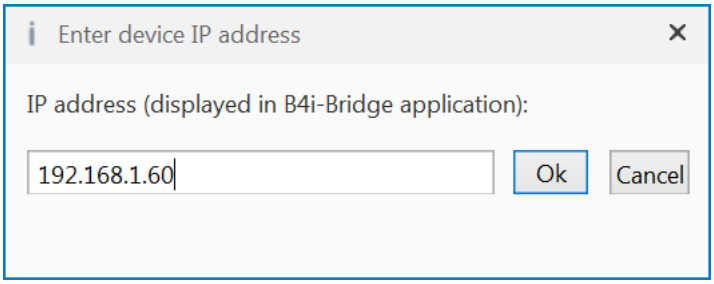
On the screen you see the IP address of the device.



In the IDE click on Tools / Device IP Address / New IP



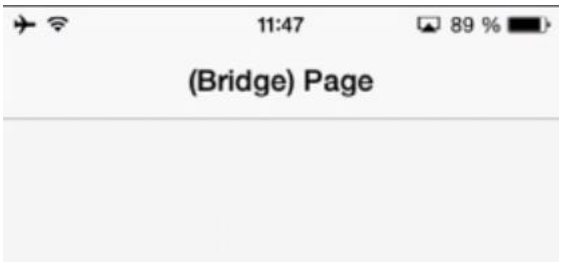
Enter the IP address:



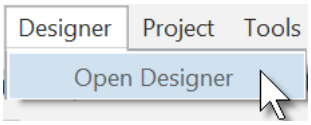
Click on 

.

You will see this screen on the device (only the upper part is shown).

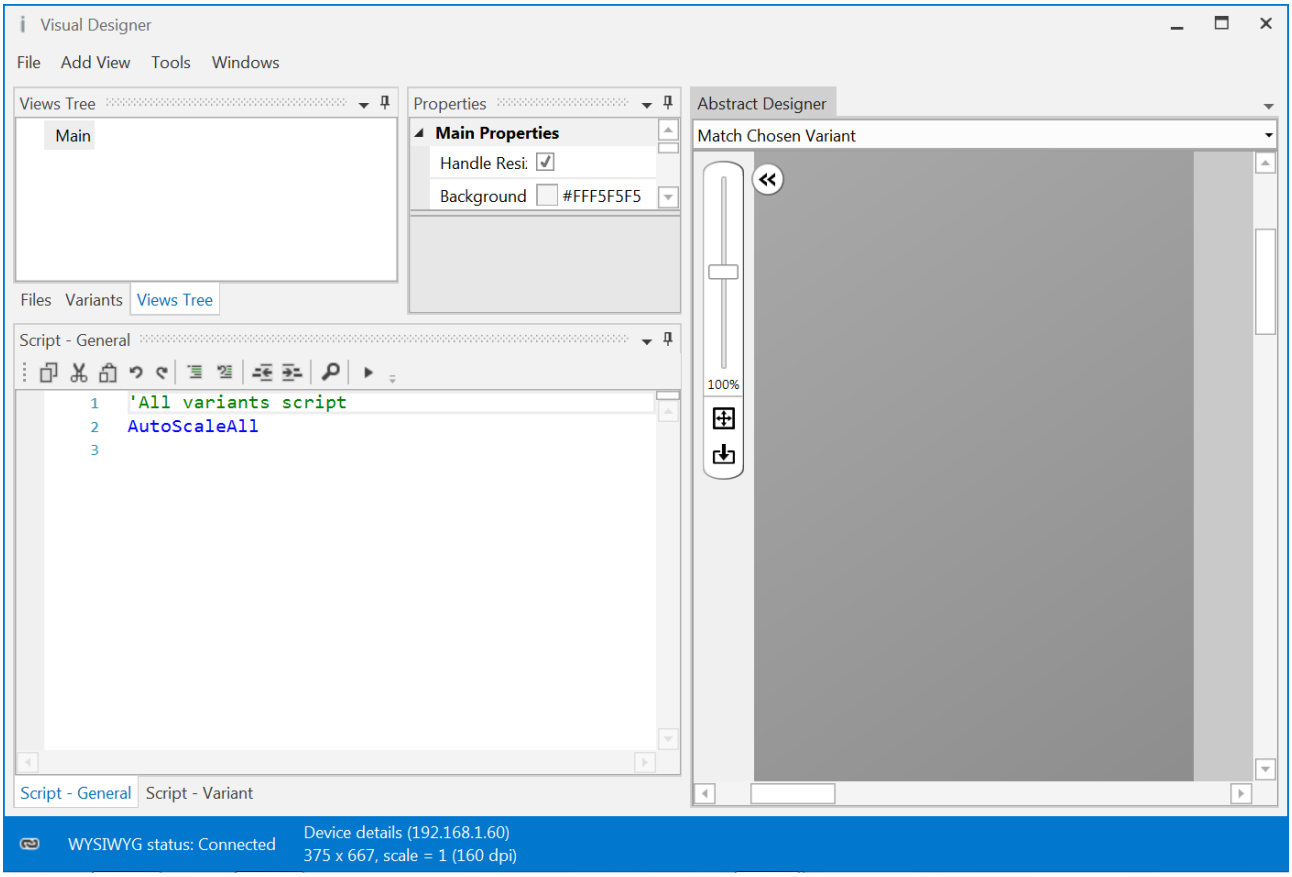


In the IDE open the Designer.

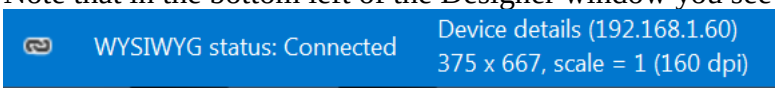


Wait until the Designer is ready.

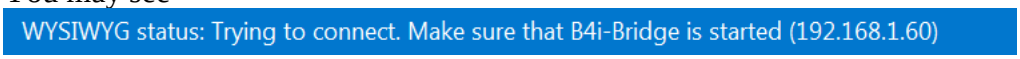
The Designer looks like this.



Note that in the bottom left of the Designer window you see the connection status:

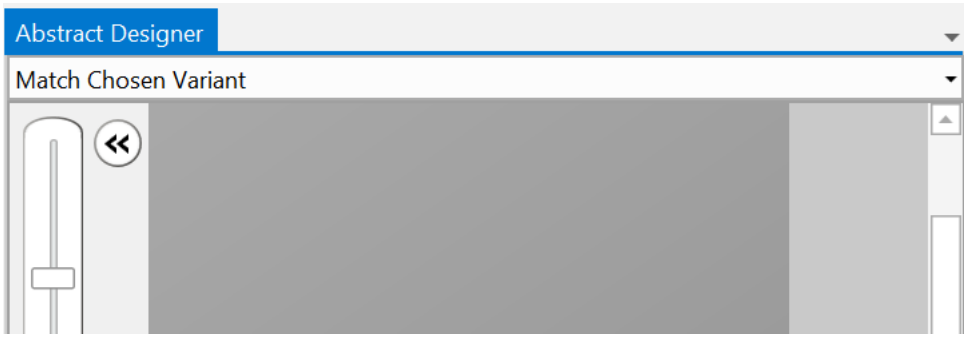


You may see



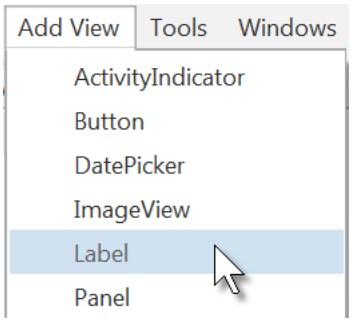
if the device is not connected.

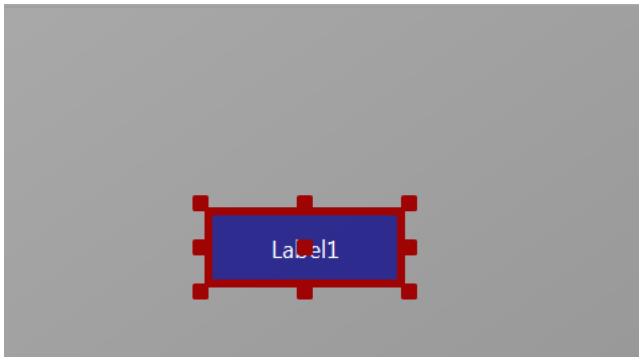
With the Designer we have also the Abstract Designer which shows the layout not exactly WYSIWYG but the positions and size of the different objects.
Only the top of the image is shown.



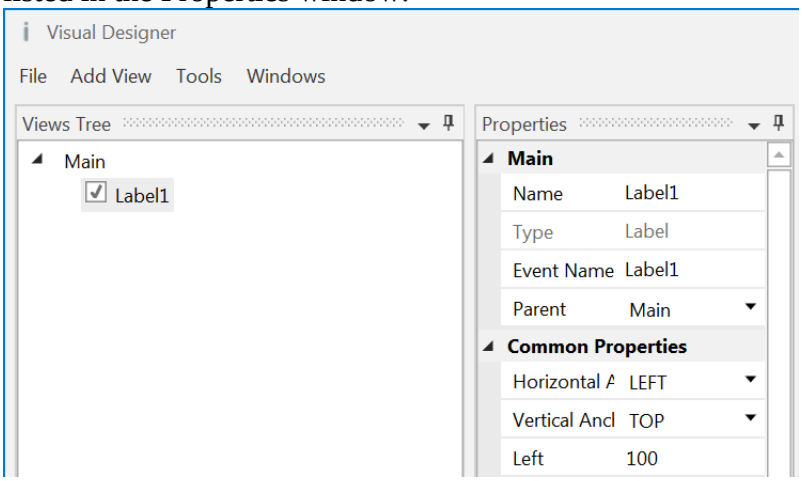
The dark gray area represents the screen area of the connected device.

Now we will add the 2 Labels for the numbers.
In the Designer, add a Label.





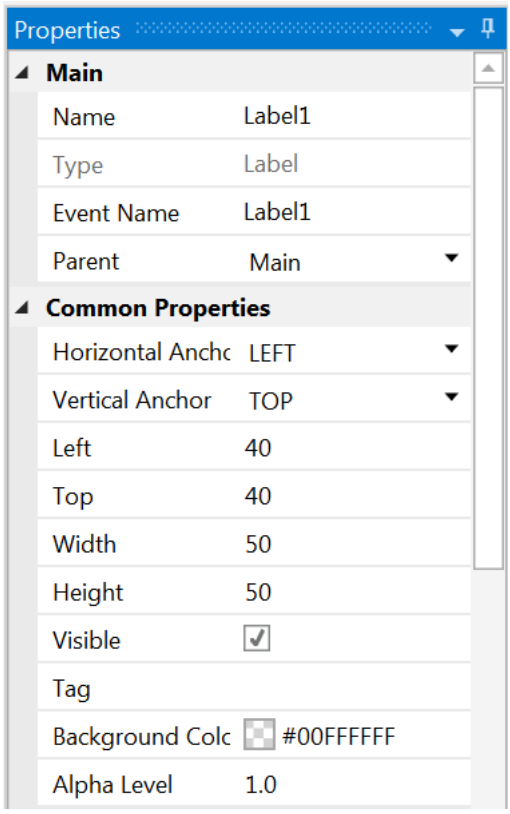
The label appears in the Abstract Designer, in the Views Tree window and its default properties are listed in the Properties window.



Resize and move the Label with the red squares like this.

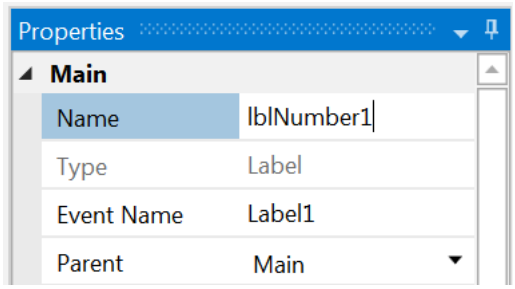
You can follow the layout on the device.

At the moment we see only Lab…
The background color is by default transparent.
Lab… stays for Label1



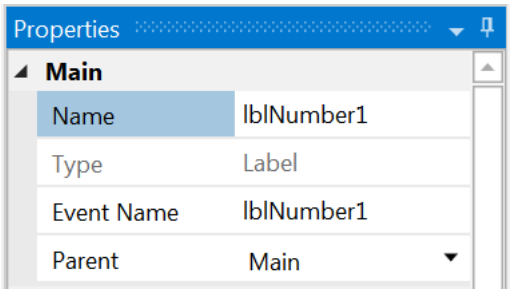
The new properties Left, Top, Width and Height are directly updated in the Properties window.
You can also modify the Left, Top, Width and Height properties directly in the Properties window.

Let us change the properties of this first Label according to our requirements.
By default, the name is Label with a number, here Label1, let us change its name to lblNumber1.
The three letters 'lbl' at the beginning mean 'Label', and 'Number1' means the first number.
It is recommended to use significant names for views so we know directly what kind of view it is and its purpose.



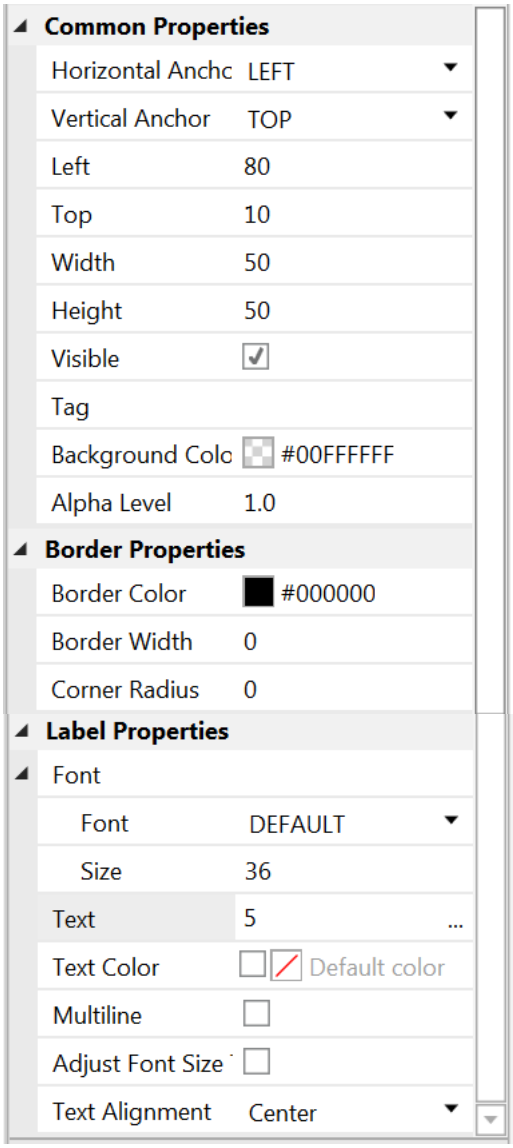
Pressing the 'Return' key or clicking elsewhere will also change the Event Name property.

We have now:



Main : main module.
Name : name of the view.
Type : type of the view. In this case, Label, which is not editable.
Event Name : generic name of the routines that handle the events of the Label.
Parent : parent view the Label belongs to.

Let us check and change the other properties:



Set Left, Top, Width and Height to the values in the picture.

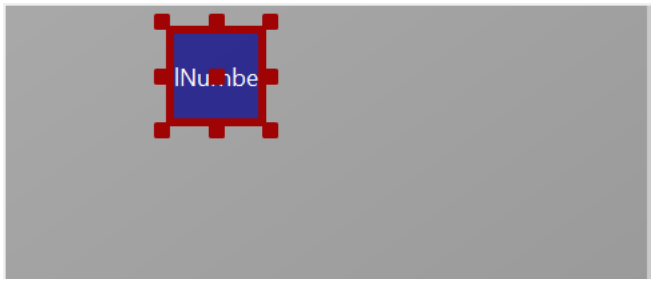
Visible is checked.

We leave the default colors.

We leave the default Font.
Text Size, we set it to 36.

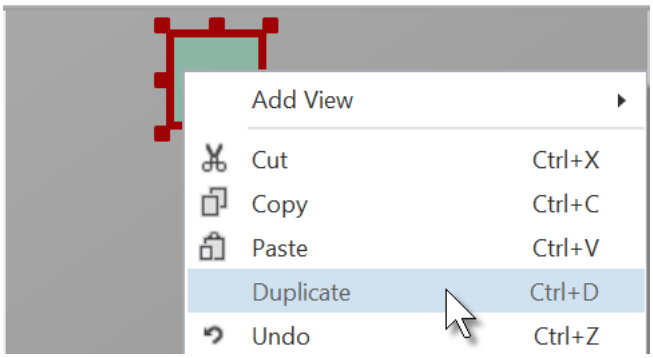
Text set to 5

Set Text Alignment to Center.



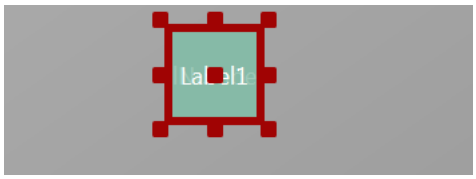
And the result in the Abstract Designer and on the device.

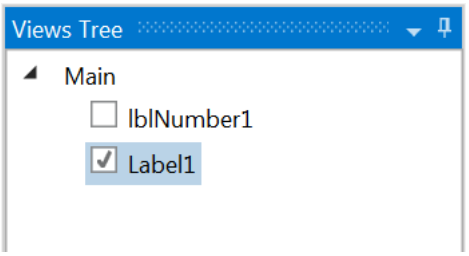
We need a second Label similar to the first one. Instead of adding a new one, we copy the first one with the same properties. Only the Name and Left properties will change.



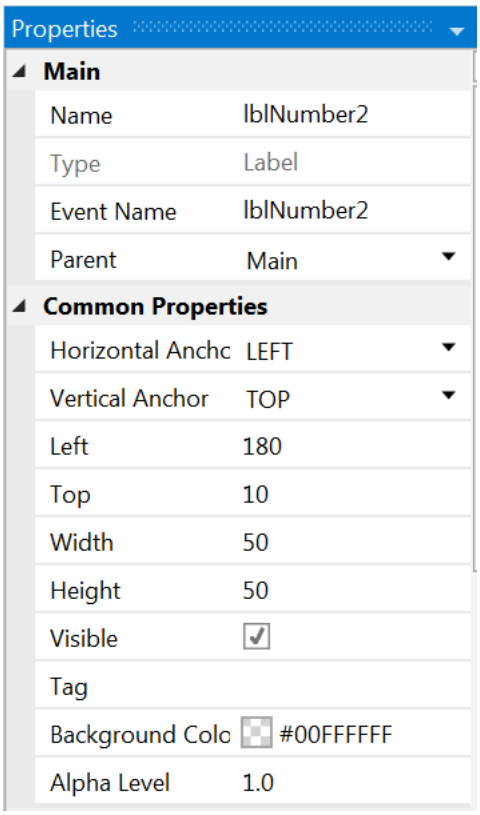
Right click on lblNumber1 and click on Duplicate in the popup menu.

The new label covers the previous one.





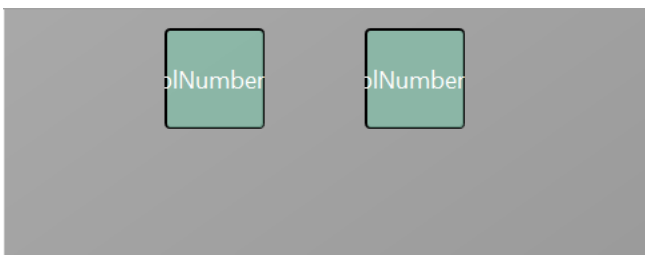
In the left part, in the Views Tree, you see the different views. The new label Label1 is added.

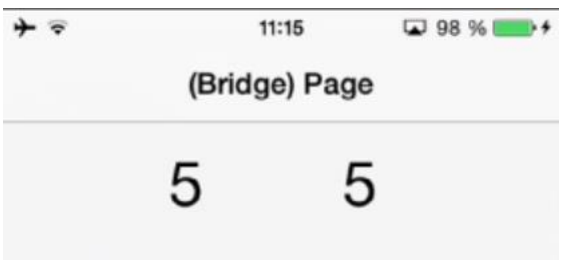


Let us position the new Label and change its name to lblNumber2.

Change the name to lblNumber2.

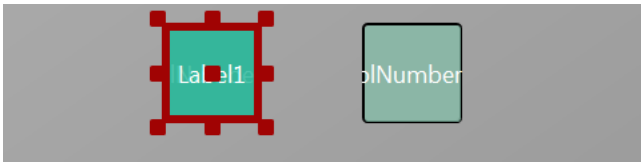
The Left property to 180.





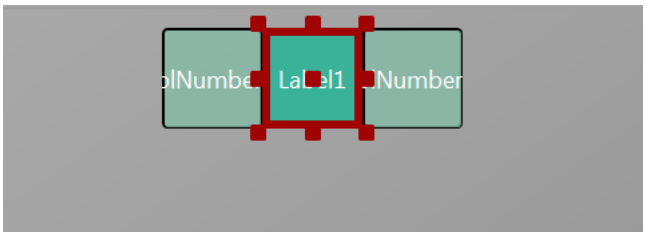
And the result in the Abstract Designer and on the device.

Let us now add a 3rd Label for the math sign. We copy once again lblNumber1.
Right click on lblNumber1 in the Abstract Designer and click Duplicate on in the popup menu.

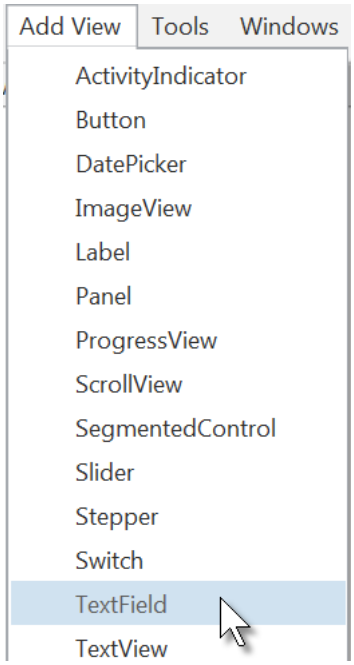


The new label covers lblNumber1.

Position it between the first two Labels and change its name to lblMathSign, its Text property to '+'.

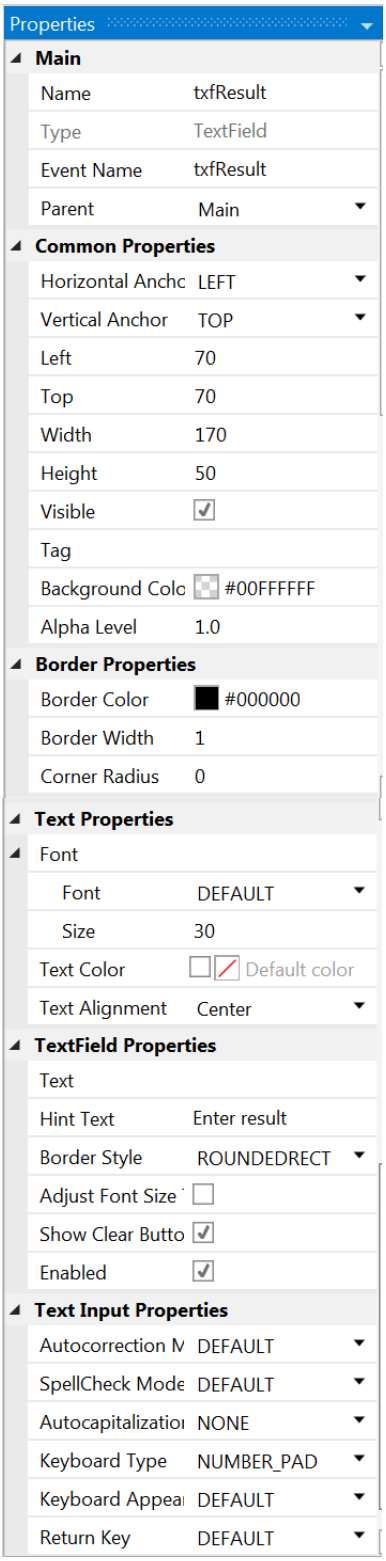


And the result in the Abstract Designer and on the device.



Now let us add a TextField view.
In the Designer Add View menu click on TextField.

Position it below the three Labels and change its name to txfResult.
'txf' means TextField and 'Result' for its purpose.



Change these properties.

Name to txfResult

Left, Top, Width and Height.

Border Width to 1

Text Size to 30

Text Alignment to Center

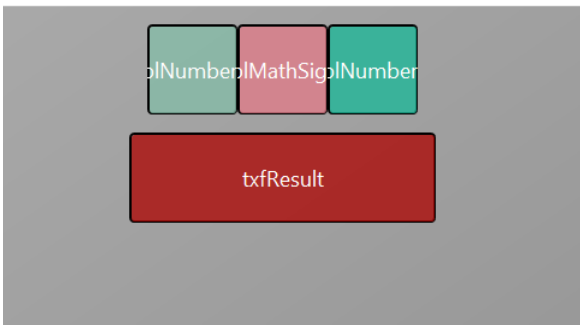
Hint Text to Enter result

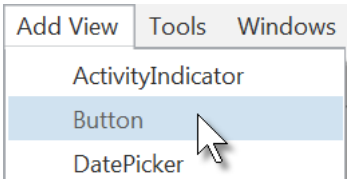
Hint Text represents the text shown in the TextField view if no text is entered.

Keyboard Type to NUMBER_PAD

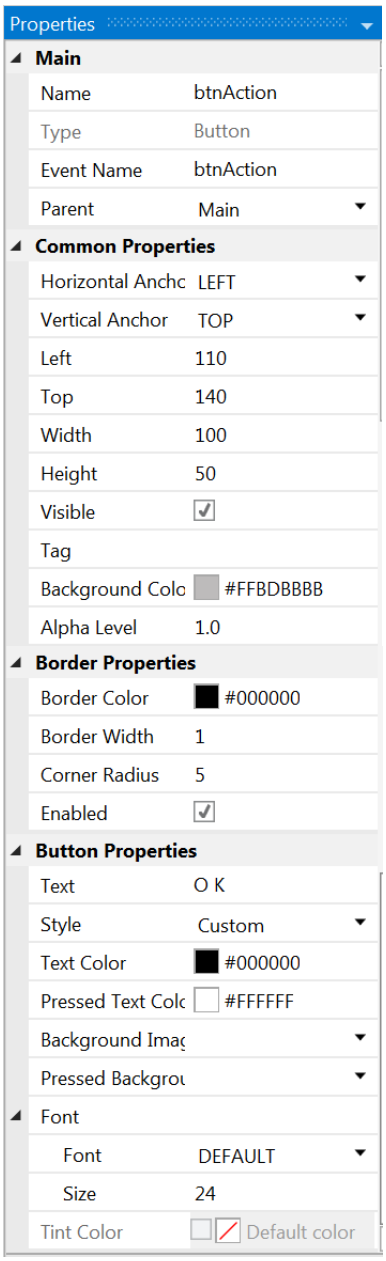
Setting Input Type to NUMBER_PAD lets the user enter only numbers.

After making these changes, you should see something like this.





Now, let's add the Button which, when pressed, will either check the result the user supplied as an answer, or will generate a new math problem, depending on the user's input.



Position it below the TextField view. Resize it and change following properties:

Name to btnAction

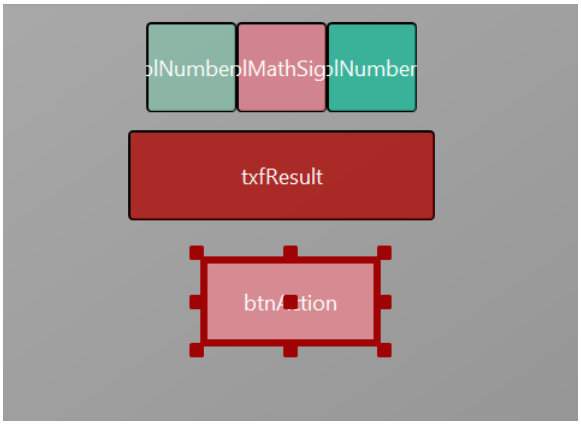
Left, Top, Width and Height.

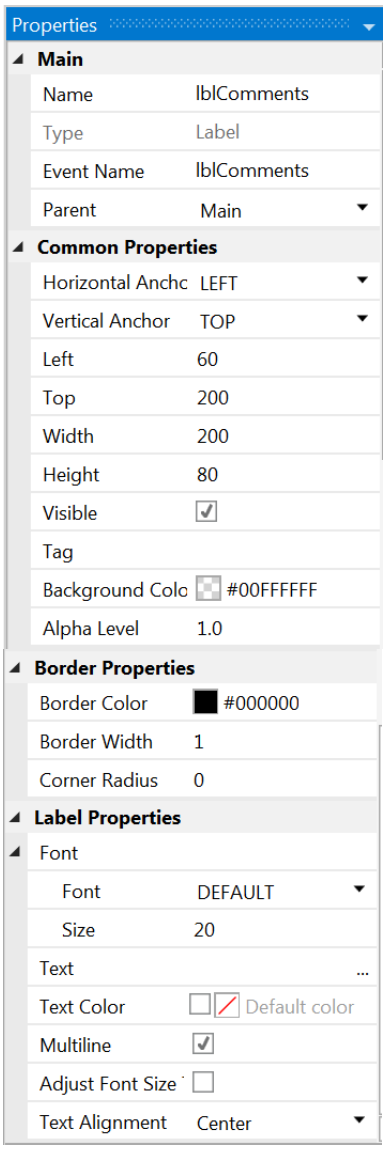
Background Color to #FFBDBBBB

Border Width to 1

Text to O K (with a space between O and K)

Text Size to 24





Let us add the last Label for the comments. Position it below the Button and resize it.

Change the following properties:

Name to lblComments

Left, Top, Width and Height.

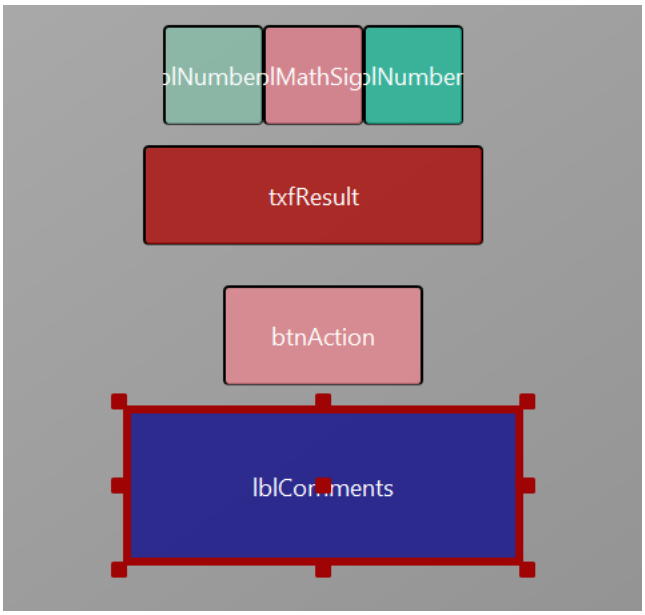
Border Width to 1

Text Size to 20

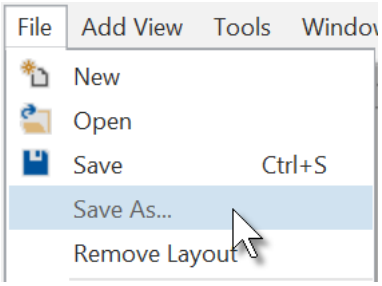
Multiline to True (checked)

Text Alignment to Center

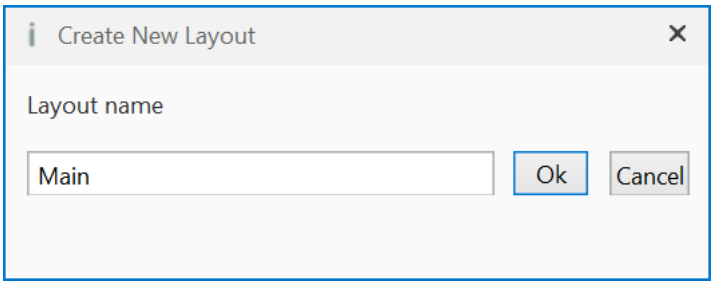
And the result.



Now we save the layout in a file.

Click on 

and save it with the name 'Main'.

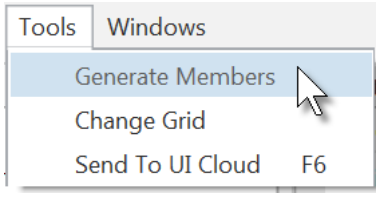


Click on 

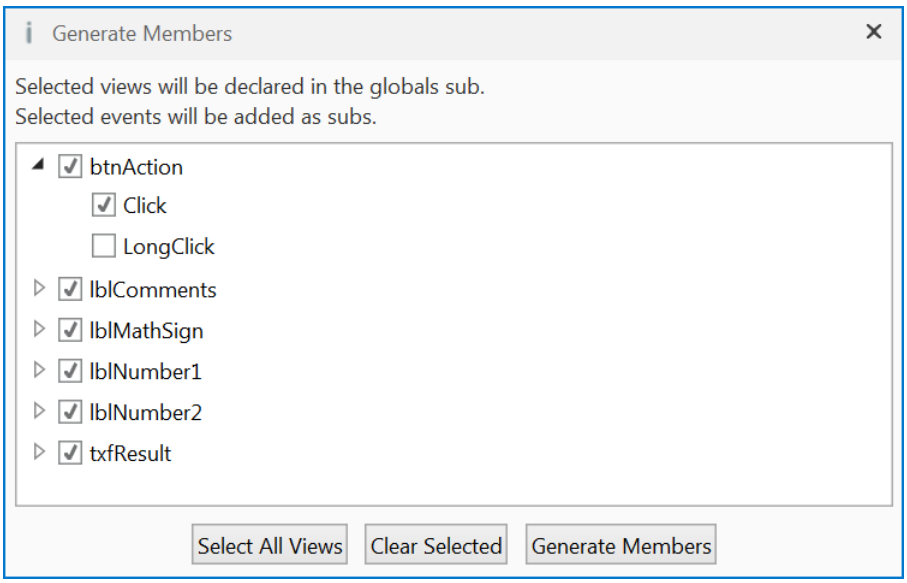
.

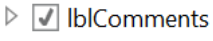
To write the routines for the project, we need to reference the Views in the code.
This can be done with the *Generate Members* tool in the Designer.

The *Generate Members* tool automatically generates references and subroutine frames.

Click on 

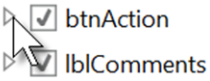
to open the generator.

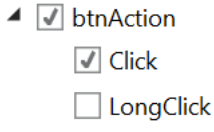


Here we find all the views added to the current layout.
We check all views and check the Click event for the btnAction Button.
Checking a view 

generates its reference in the Globals Sub routine in the code.
This is needed to make the view recognized by the system and allow the autocomplete function.

```
Private btnAction As Button
Private lblComments As Label
Private lblMathSign As Label
Private lblNumber1 As Label
Private lblNumber2 As Label
Private txfResult As TextField
```

Clicking on 

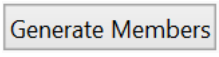
shows all events for the selected view 

.
Clicking on an event of a view 

generates the Sub frame for this event.

```
Sub btnAction_Click

End Sub
```

Click on 

to generate the references and Sub frames, then close the window 

.

Now we go back to the IDE to enter the code.

On the top of the program code we have:

```
Sub Process_Globals
   'These global variables will be declared once when the application starts.
   'Public variables can be accessed from all modules.
   Public App As Application
   Public NavControl As NavigationController
   Private Page1 As Page

   Private btnAction As Button
   Private lblComments As Label
   Private lblMathSign As Label
   Private lblNumber1 As Label
   Private lblNumber2 As Label
   Private txfResult As TextField
End Sub
```

These lines are automatically in the project code.

```
   Public App As Application
   Public NavControl As NavigationController
   Private Page1 As Page
```

iOS needs an Application, a NavigationControl and at least one Page, the details are explained in the chapter Process life cycle.

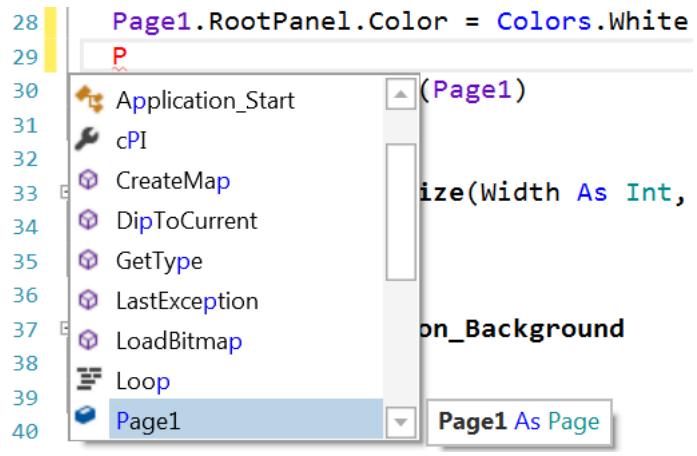
Below the code above we have the `Application_Start` routine which is the first routine called when the program starts.
The content below is also added automatically in each new project.

```
Private Sub Application_Start (Nav As NavigationController)
   NavControl = Nav
   Page1.Initialize("Page1")
   Page1.Title = "Page 1"
   Page1.RootPanel.Color = Colors.White
   NavControl.ShowPage(Page1)
End Sub
```

`NavControl = Nav` > Sets `NavControl` as the `NavigationController`
`Page1.Initialize("Page1")` > Initializes `Page1`, `"Page1"` is the generic EventName of Page1.
`Page1.Title = "Page 1"` > Sets the Page Title
`Page1.RootPanel.Color = Colors.White` > Sets the background color to white.
`NavControl.ShowPage(Page1)` > Shows Page1 on the device.

First, we need our program to load the layout file we defined in the previous pages.
The file must be loaded onto the RootPanel of Page1, we load it just before
`NavControl.ShowPage(Page1)`
We take advantage of the autocomplete and in-line help features of B4i.

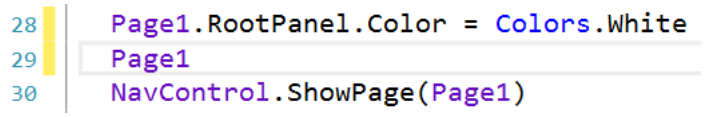
Enter P in a new line 29.



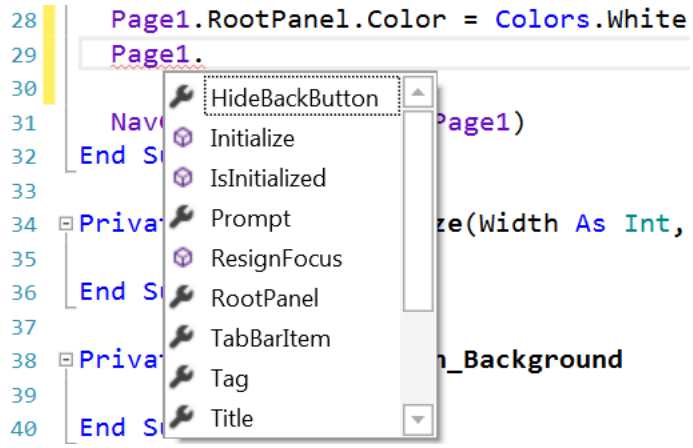
A drop-down list appears with all available keywords, views, routines etc.

The first item beginning with the letter 'P' is highlighted.

Click on Return to validate.

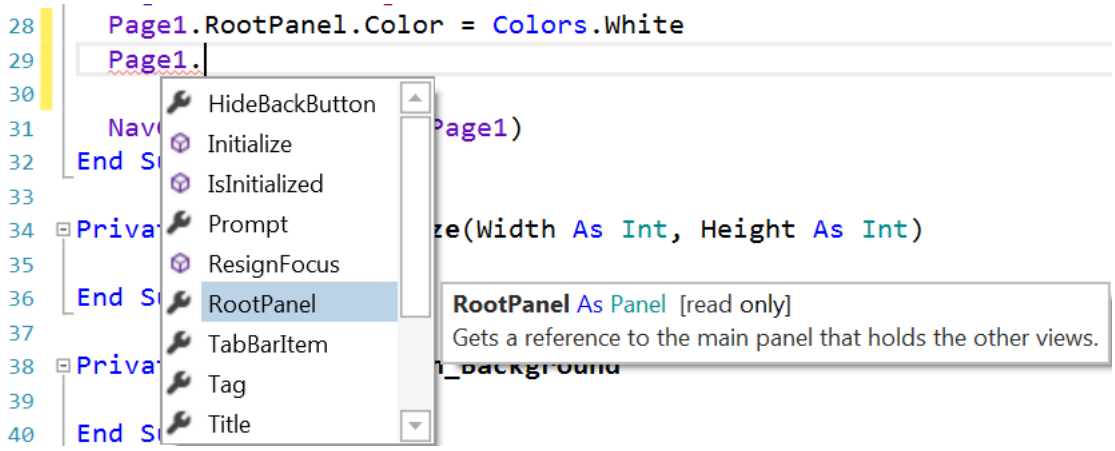


`P` is completed to `Page1`.
Now enter a dot "."



The drop-down list contains all properties of a Page view.

With the Down key go down to RootPanel.



We see RootPanel highlighted, and besides the list the in-line help with the syntax for the property and an explanation.

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel
30    NavControl.ShowPage(Page1)
```

Click on Return to validate.

Enter a dot “.”

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel.
30    NavControl.Sh
31  End Sub
32
33  Private Sub Pag
34
35  End Sub
36
37  Private Sub App
38
39  End Sub
40
```

ACTION_DOWN
ACTION_MOVE
ACTION_UP
AddView
Alpha
BringToFront
Color
GetAllViewsRecursive
GetView

Again we get a drop-down list with the properties of a Panel.

With the Down key go down to LoadLayout.

Again we see the syntax and the explanation.

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel.
30    NavControl.Sh
31  End Sub
32
33  Private Sub Pag                    , Height As Int)
34
35  End Sub
36
37  Private Sub App
38
39  End Sub
40
```

Height
Initialize
IsFocused
IsInitialized
Left
LoadLayout
NumberOfViews
RemoveAllViews
RemoveViewAt

LoadLayout (LayoutFile As String) As LayoutValues
Loads a layout file to the panel.

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel.LoadLayout
30    NavControl.ShowPage(Page1)
```

Click on Return to validate.

Enter “(”.

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel.LoadLayout(
30    NavControl.ShowPage(Page1)
31  End Sub
32
```

LoadLayout (**LayoutFile** As String) As LayoutValues
Loads a layout file to the panel.

The in-line help shows what to do and the explanation.

```
28    Page1.RootPanel.Color = Colors.White
29    Page1.RootPanel.LoadLayout("Main")
30    NavControl.ShowPage(Page1)
```

Complete the line with the layout file name.
The file extension is not needed.
The file name `"Main"` is between quotes because it is a String.

The yellow line in the left border shows that a modification was made in the code.
As soon as you save the code the yellow line will be changed to a green line.

We want to generate a new problem as soon as the program starts. Therefore, we add a call to the New subroutine in Application_Start.

```
Private Sub Application_Start (Nav As NavigationController)
   NavControl = Nav
   Page1.Initialize("Page1")
   Page1.Title = "Page 1"
   Page1.RootPanel.Color = Colors.White
   NavControl.ShowPage(Page1)

   New
End Sub
```

New is displayed in red because the 'New' routine has not yet been defined.
Generating a new problem means generating two new random values between 1 and 9 (inclusive) for Number1 and Number2, then showing the values using the lblNumber1 and lblNumber2 'Text' properties.

To do this we enter following code:
In Sub Process_Globals we add two variables for the two numbers.

```
   Private Number1, Number2 As Int
End Sub
```

And the 'New' Subroutine:

```
Private Sub New
   Number1 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1           ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2           ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   txfResult.Text = ""                 ' Sets edtResult.Text to empty
End Sub
```

The following line of code generates a random number from '1' (inclusive) to '10' (exclusive) :
`Rnd(1, 10)`

In this line `Number1 = Rnd(1, 10)  ' Generates a random number between 1 and 9`
The text after the quote, `' Generates...`, is considered as a comment.
It is good practice to add comments explaining the purpose of the code.

The following line displays the comment in the lblComment view:
`lblComments.Text = "Enter the result" & CRLF & "and click on OK"`
`CRLF` is the LineFeed character.

Now we add the code for the Button click event.

We have two cases:
- When the Button text is equal to "O K", it means that a new problem is displayed, and the program is waiting for the user to enter a result and press the Button.
- When the Button text is equal to "NEW", it means that the user has entered a correct answer and when the user clicks on the Button a new problem will be generated.

```
Private Sub btnAction_Click
   If btnAction.Text = "O K" Then
      If txfResult.Text="" Then
         Msgbox("No result entered","E R R O R")
      Else
         CheckResult
      End If
   Else
      New
      btnAction.Text = "O K"
   End If
End Sub
```

`If btnAction.Text = "O K" Then` checks if the Button text equals "O K"
If yes then we check if the TextField is empty.
If yes, we display a MessageBox telling the user that there is no result in the TextField view.
If no, we check if the result is correct or if it is wrong.
If no then we generate a new problem, set the Button text to "O K" and clear the TextField view.

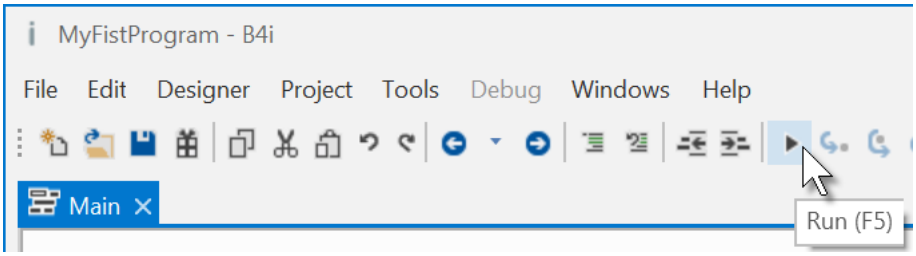
The last routine checks the result.

```
Private Sub CheckResult
   If txfResult.Text = Number1 + Number2 Then
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      btnAction.Text = "N E W"
   Else
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
   End If
End Sub
```

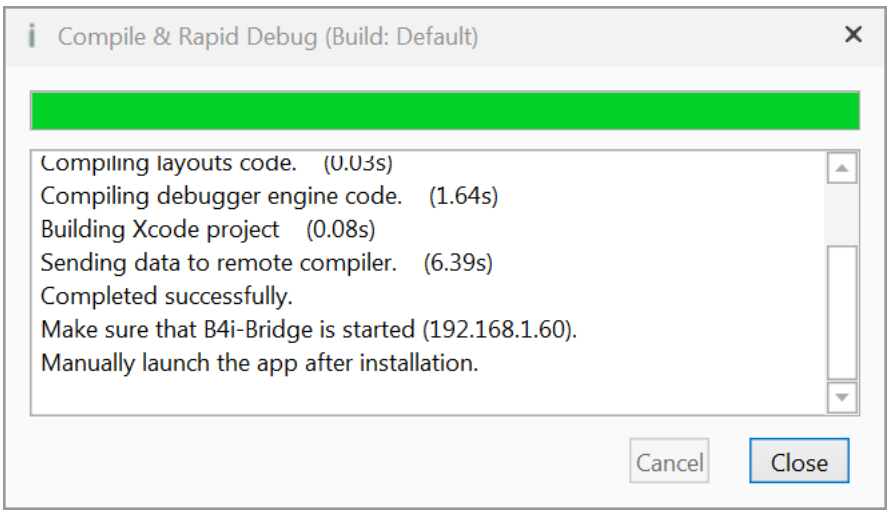
With `If txfResult.Text = Number1 + Number2 Then` we check if the entered result is correct.

If yes, we display in the lblComments label the text below:
'G O O D result'
'Click on NEW'
and we change the Button text to "N E W ".
If no, we display in the lblComments label the text below:
W R O N G result
Enter a new result
and click OK

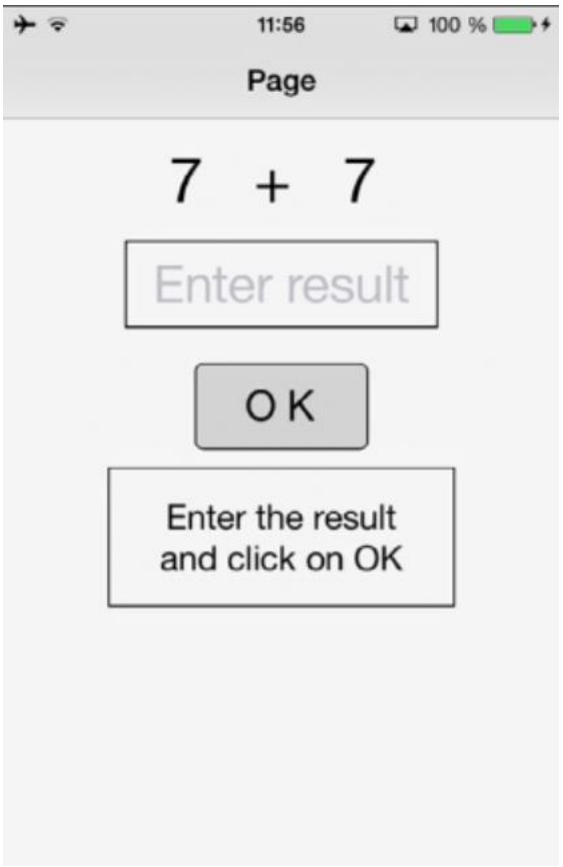
Let us now compile the program and transfer it to the Device.
In the IDE on top click on  :



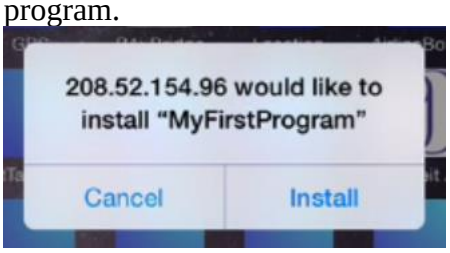
The program is going to be compiled.



When you see `'Completed successfully.'` as in the message box, the compiling and transfer is finished.



Looking at the device, you should see something similar to the image below when you first run the program.



Touch on Install.
Then you will see somewhere on the device the icon of the program  , touch it to run the program.

Then you should see something similar to the image on the left, with different numbers.

Of course, we could make aesthetic improvements in the layout, but this was not the main issue for the first program.

Touch on 

to activate the keyboard

Enter 14

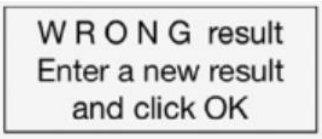
You will see this screen.

Click on 

to confirm the result entry.

If the result is correct you will see the screen on the left.

If the result is wrong the message is:

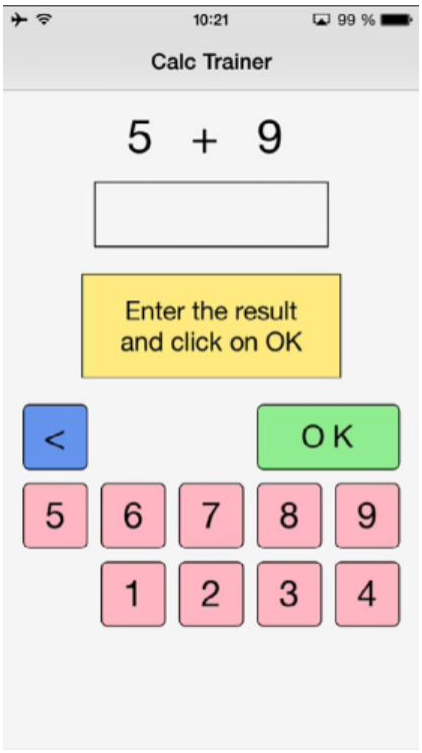


3.8 Second B4i program (SecondProgram.b4i)

The project is available in the SourceCode folder:
SourceCode\SecondProgram\B4i\SecondProgram.b4i.

Improvements to “My first program”.
- Independent numeric keyboard to avoid the use of the virtual keyboard.
- Colors in the comment label.

Create a new folder called “SecondProgram”. Copy all the files and folders from MyFirstProgram to the new SecondProgram folder and rename the program file MyFirstProgram.b4i to SecondProgram.b4i and MyFirstProgram.meta to SecondProgram.meta.

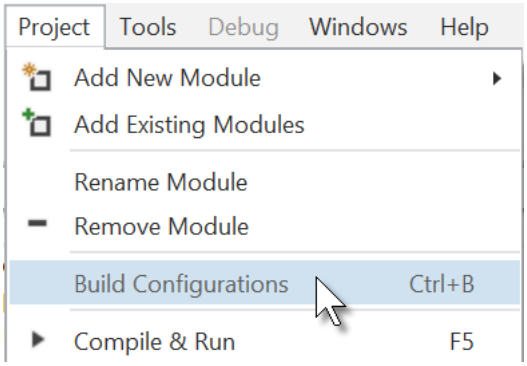


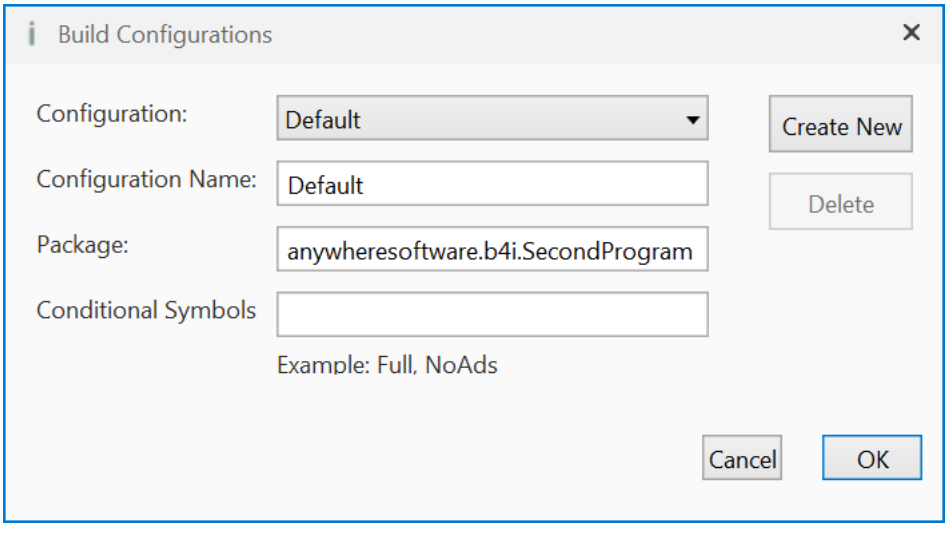
Load this new program in the IDE.

Run the Designer.

We need to change the Package Name.

In the IDE Project menu.
Click on Build Configurations.



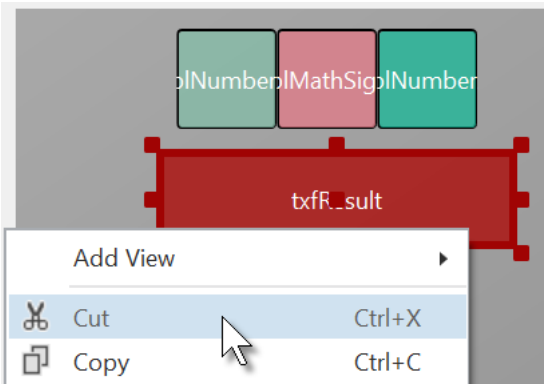


Change the Package name to anywheresoftware.b4i.SecondProgram and click on OK.

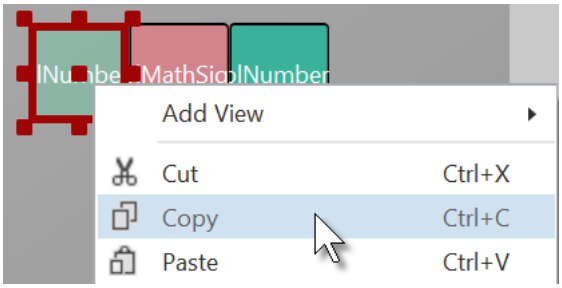
Then we must change the ApplicationLabel on the very top of the code.

```
#Region  Project Attributes
   #ApplicationLabel: SecondProgram
```

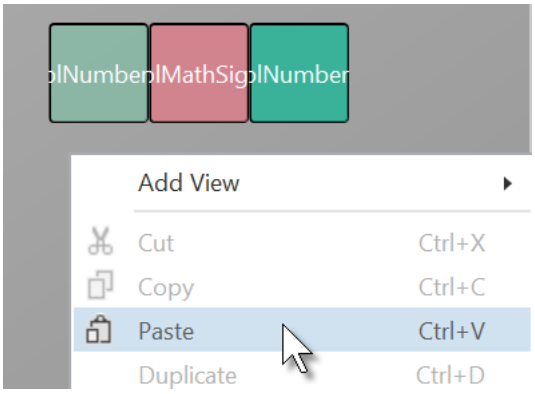
We want to replace the txfResult TextField view by a new Label.
In the Abstract Designer, click on the txfResult view.



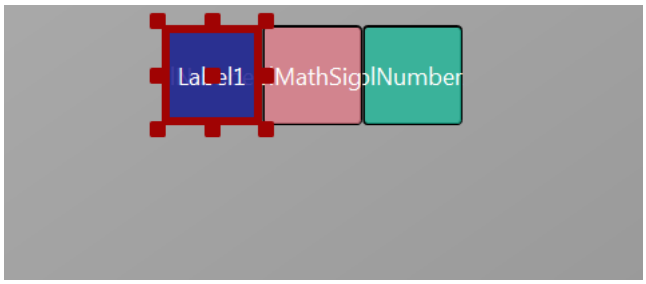
Right click on txfResult
and click on Cut.



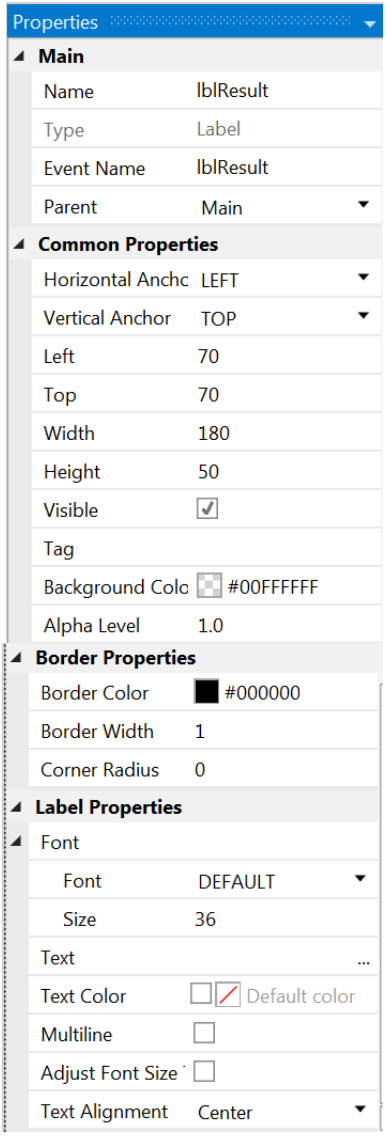
Right click on lblNumber1
and click on Copy.



Right click somewhere else
and click on Paste.



The new label covers lblNumber1.



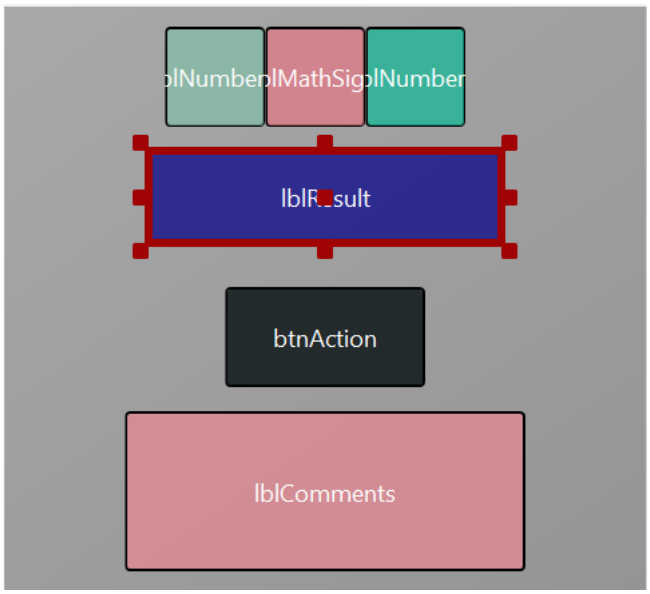
Modify the following properties:

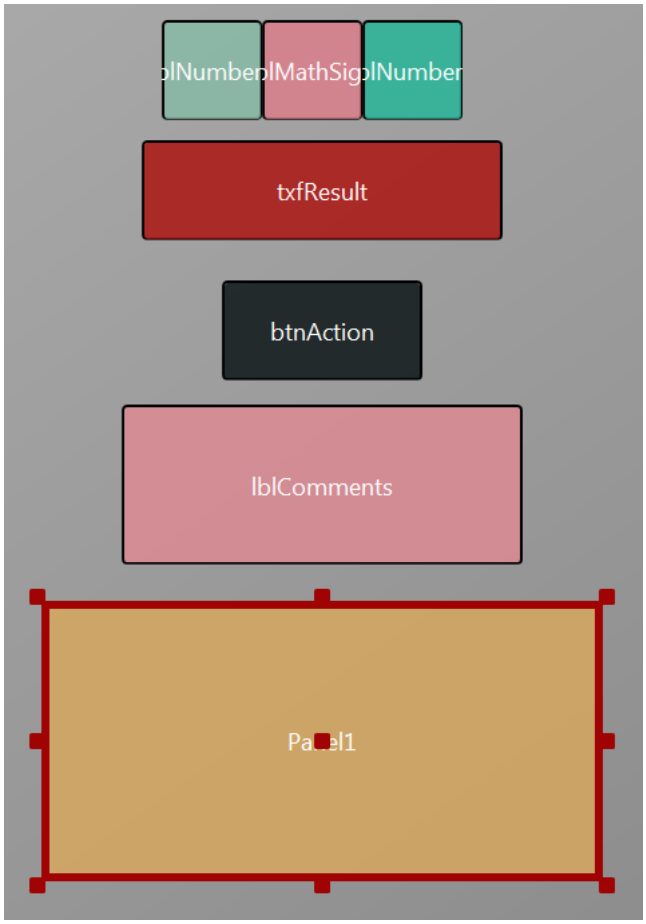
Name to lblResult

Left, Top, Width, Height

Boarder Width to 1

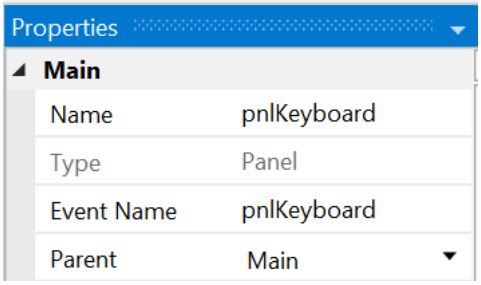
Text to "" no character



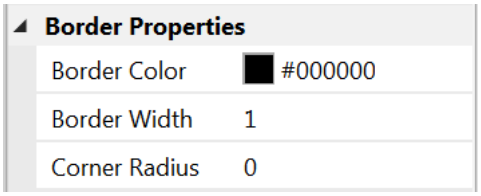


Let us add a Panel for the keyboard buttons.

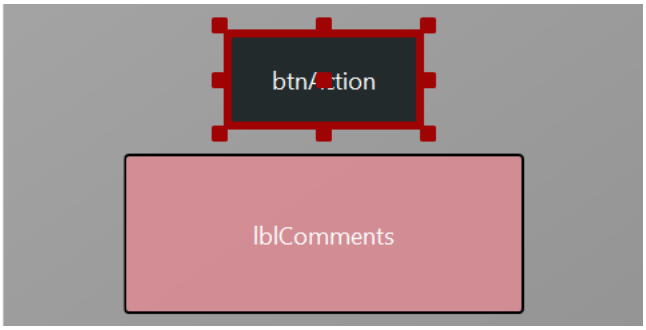
Position and resize it as in the image.



Change its Name to pnlKeyboard
"pnl" for Panel, the view type.

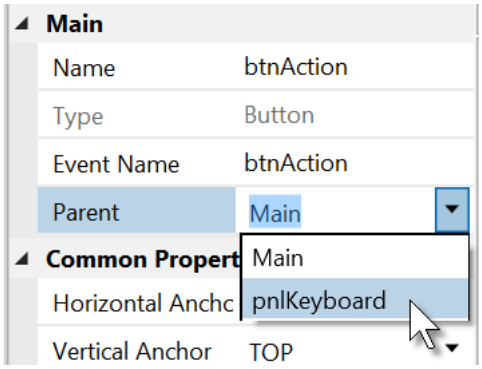


Change
Corner radius to 0

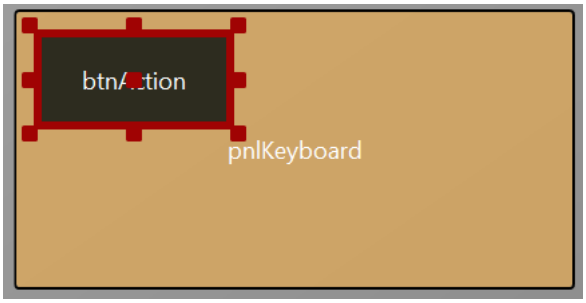


We will move btnAction from Main to the pnlKeyboard Panel.

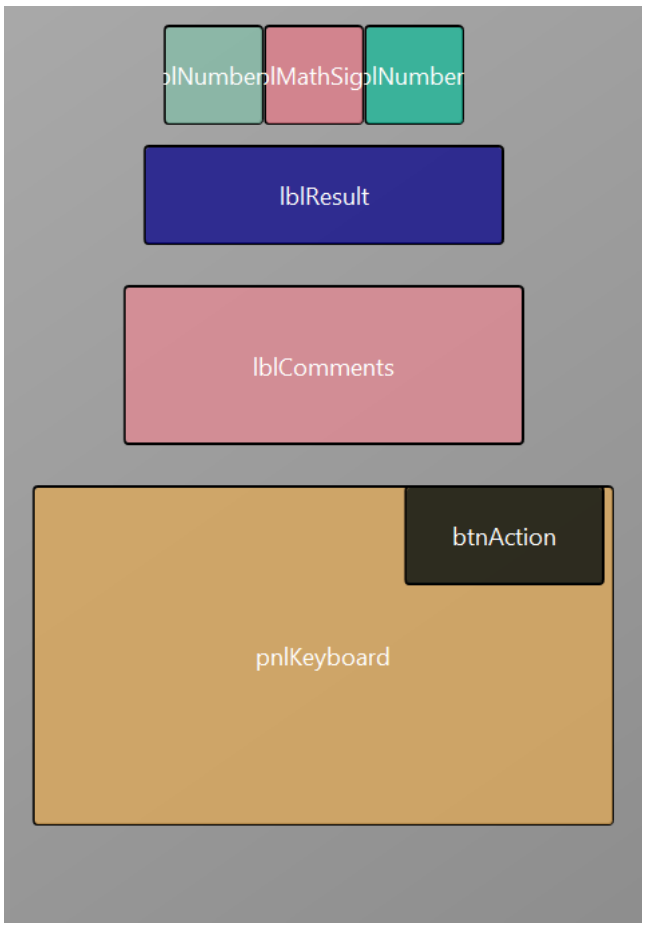
Click on btnAction.



In the Parent list click on pnlKeyboard.



The button now belongs to the Panel.



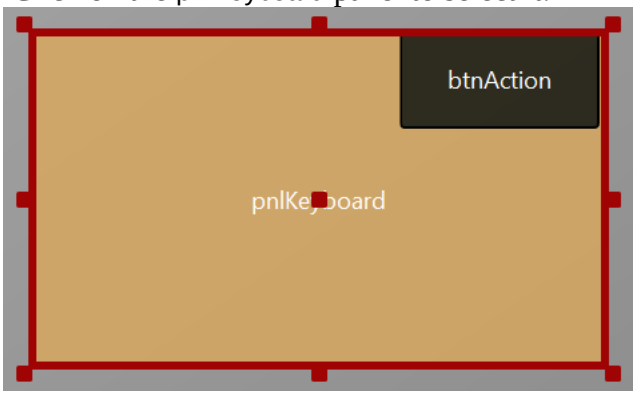
Now we rearrange the views to get some more space for the keyboard.

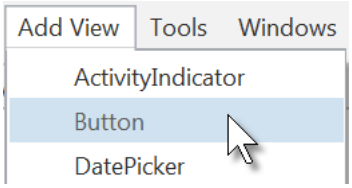
Set the properties below:

lblComments	Top = 140
pnlKeyboard	Left = 15
pnlKeyboard	Top = 240
pnlKeyboard	Width = 290
pnlKeyboard	Height = 170
pnlKeyboard	BorderWidth = 0

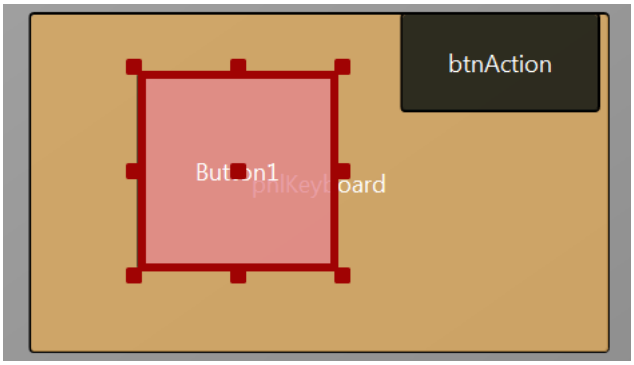
Move btnAction to the upper right corner of pnlKeyboard.

Click on the pnlKeyboard panel to select it.

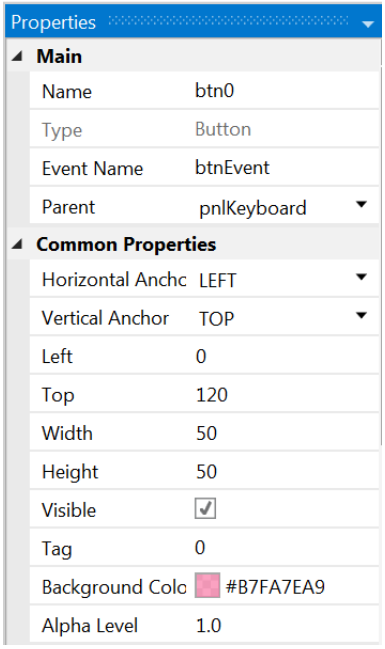


Click on 

to add a new button.



The new button is added.

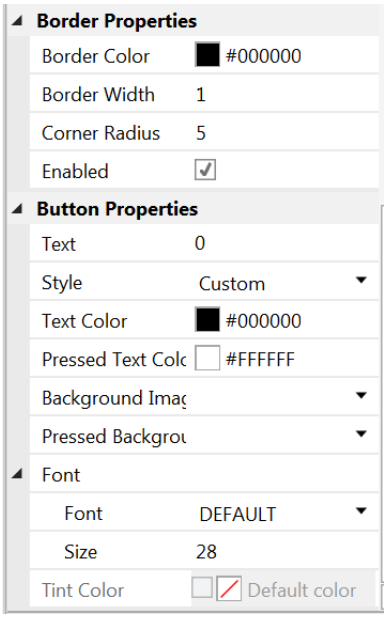


Change following properties:

Name to btn0
Event name to btnEvent

Left to 0
Top to 120
Width to 50
Height to 50

Tag to 0
Background Color to #B7FA7EA9



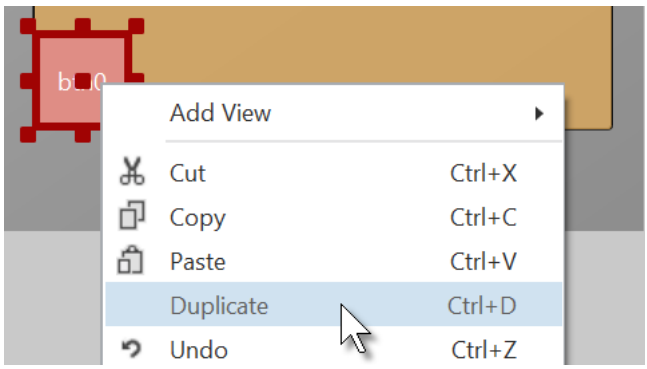
Boarder Width to 1
Corner Radius to 5

Text to 0

Size to 28

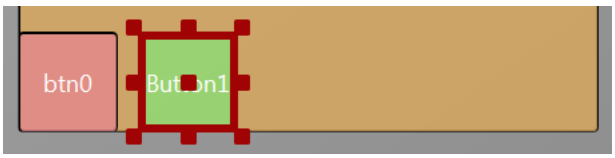
The button looks now like this.

Let us duplicate btn0 and position the new one beside button btn0.

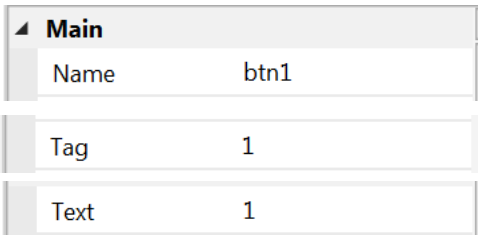


Select the Button btn0.

Right click on btn0
and click on Duplicate.



Move the new Button next to the previous one with a space.

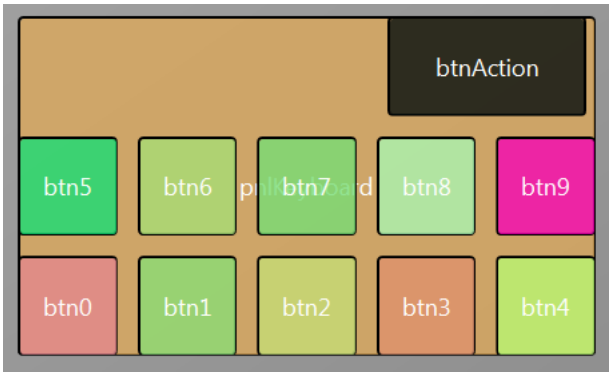


Change the following properties:
Name to btn1

Tag to 1

Text to 1

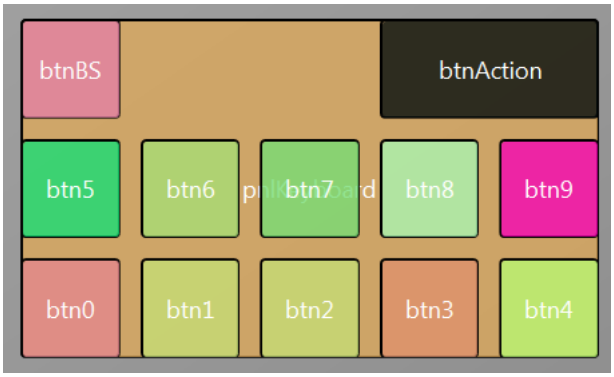
And the result.



Add 8 more Buttons and position them like in the image.

Change following properties:

Name	btn2,	btn3,		btn4	etc.
Tag	2 ,	3	,	4	etc.
Text	2 ,	3	,	4	etc.

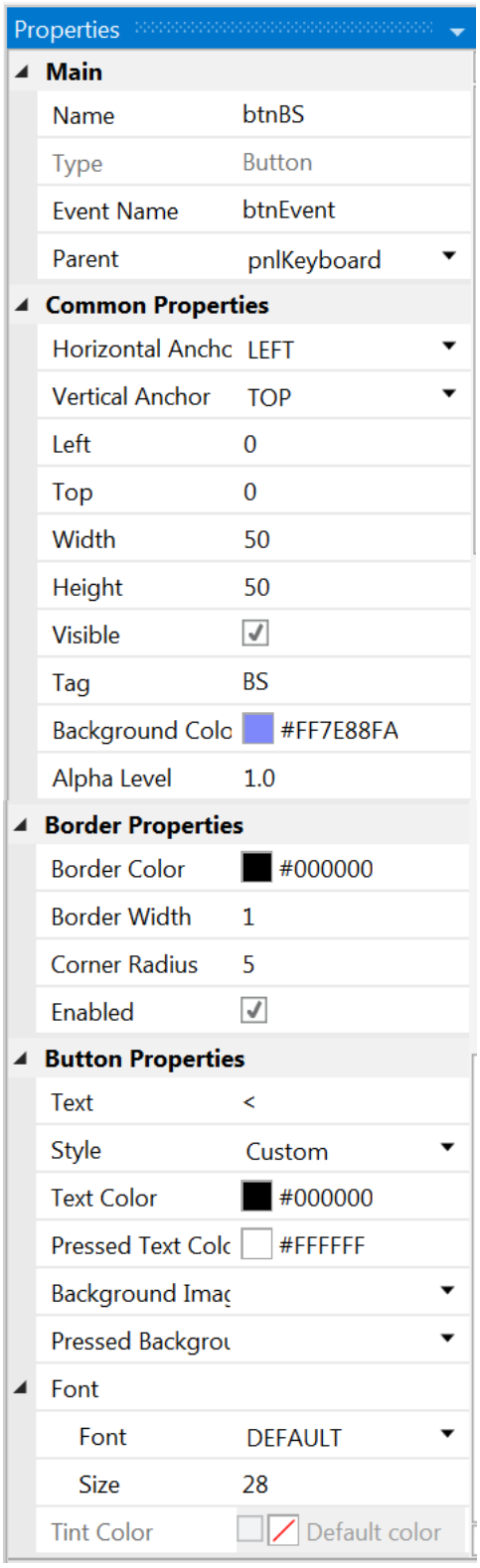


To create the BackSpace button, duplicate one of the number buttons, and position it in the top left corner.

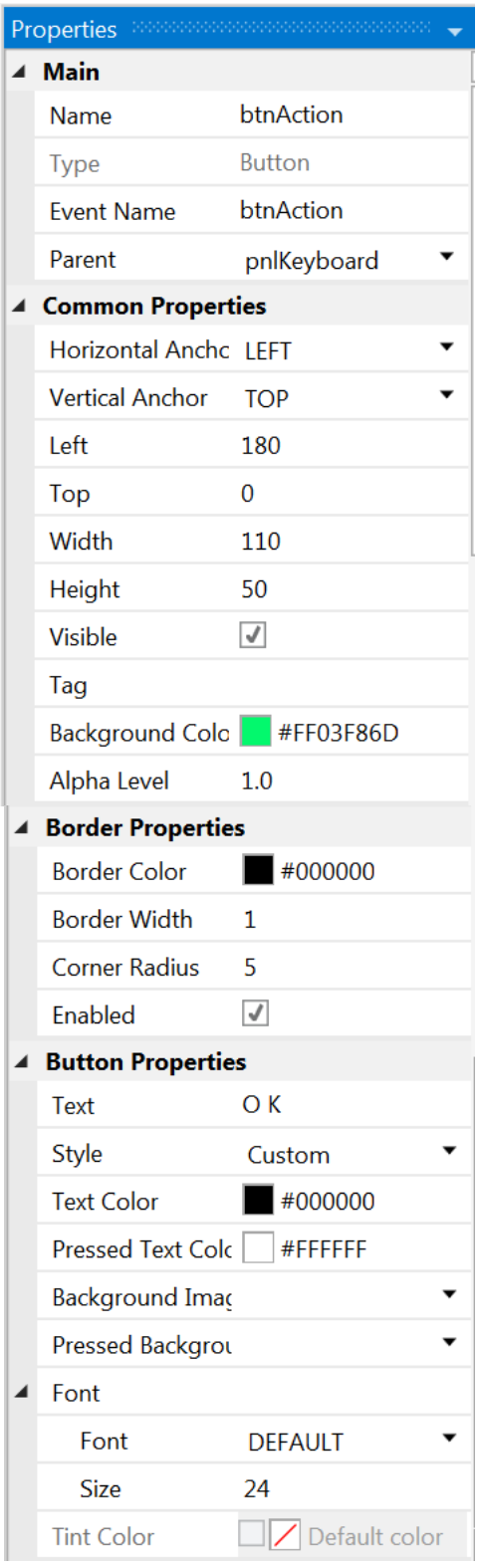
Resize and position btnAction.

Change their Name, Tag, Text and Color properties as below.

btnBS <



btnAction O K



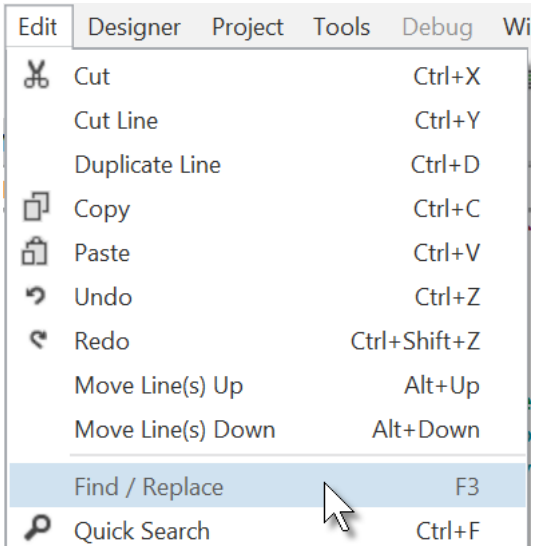
The finished new layout on the device.

If you had connect the device since the beginning you could have followed all the evolutions of the layout on the device.

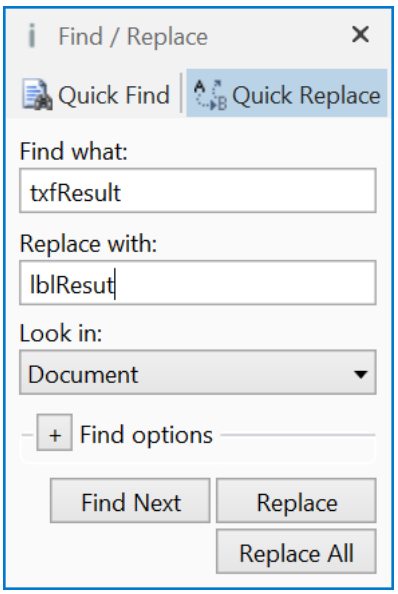
Now we will update the code.
First, we must replace the txfResult by lblResult because we replaced the TextField view by a Label.

```
17    Private btnAction As Button
18    Private lblMathSign As Label
19    Private lblComments As Label
20    Private lblNumber1 As Label
21    Private lblNumber2 As Label
22    Private txfResult As TextField
```

Double click on txfResult to select it.



Click on Find / Replace



The Find / Replace window is displayed.

Click on Replace All and close the window.

We also need to change its view type form TextField to Label.

```
Private lblResult As Label
```

Now we write the routine that handles the Click events of the Buttons.
The Event Name for all buttons, except btnAction, is "btnEvent".
The routine name for the associated click event will be btnEvent_Click.
Enter the following code:

```
Private Sub btnEvent_Click

End Sub
```

We need to know what button raised the event. For this, we use the Sender object which is a special object that holds the object reference of the view that generated the event in the event routine.

```
Private Sub btnEvent_Click
   Dim btnSender As Button

   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
   Case Else
   End Select
End Sub
```

To have access to the properties of the view that raised the event we declare a local variable `Dim btnSender As Button`.
And set `btnSender = Sender`.

Then, to differentiate between the backspace button and the numeric buttons we use a Select / Case / End Select structure and use the Tag property of the buttons.
Remember, when we added the different buttons we set their Tag property to BS, 0, 1, 2 etc.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
```

Select sets the variable to test.
Checks if it is the button with the "BS" tag value.
Handles all the other buttons.

Now we add the code for the numeric buttons.
We want to add the value of the button to the text in the lblResult Label.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

This is done in this line
```
   lblResult.Text = lblResult.Text & btnSender.Text
```

The "&" character means concatenation, so we just append to the already existing text the value of the Text property of the button that raised the event.

Now we add the code for the BackSpace button.
```
   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length > 0 Then
         lblResult.Text = lblResult.Text.SubString2(0, lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

When clicking on the BS button we must remove the last character from the existing text in lblResult. However, this is only valid if the length of the text is bigger than 0. This is checked with:
```
If lblResult.Text.Length > 0 Then
```

To remove the last character we use the SubString2 function.
```
lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
```

SubString2(BeginIndex, EndIndex) extracts a new string beginning at BeginIndex (inclusive) until EndIndex (exclusive).

Now the whole routine is finished.
```
Private Sub btnEvent_Click
   Private btnSender As Button

   btnSender = Sender
   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

In `Sub btnAction_Click` we add, at the end, `lblResult.Text = ""` to clear the text.
```
   Else
      New
      btnAction.Text = "O K"
      lblResult.Text = ""
   End If
End Sub
```

We can try to improve the user interface of the program by adding some colors to the lblComments Label.
Let us set:
- Yellow for a new problem
- Light Green for a GOOD answer
- Light Red for a WRONG answer.

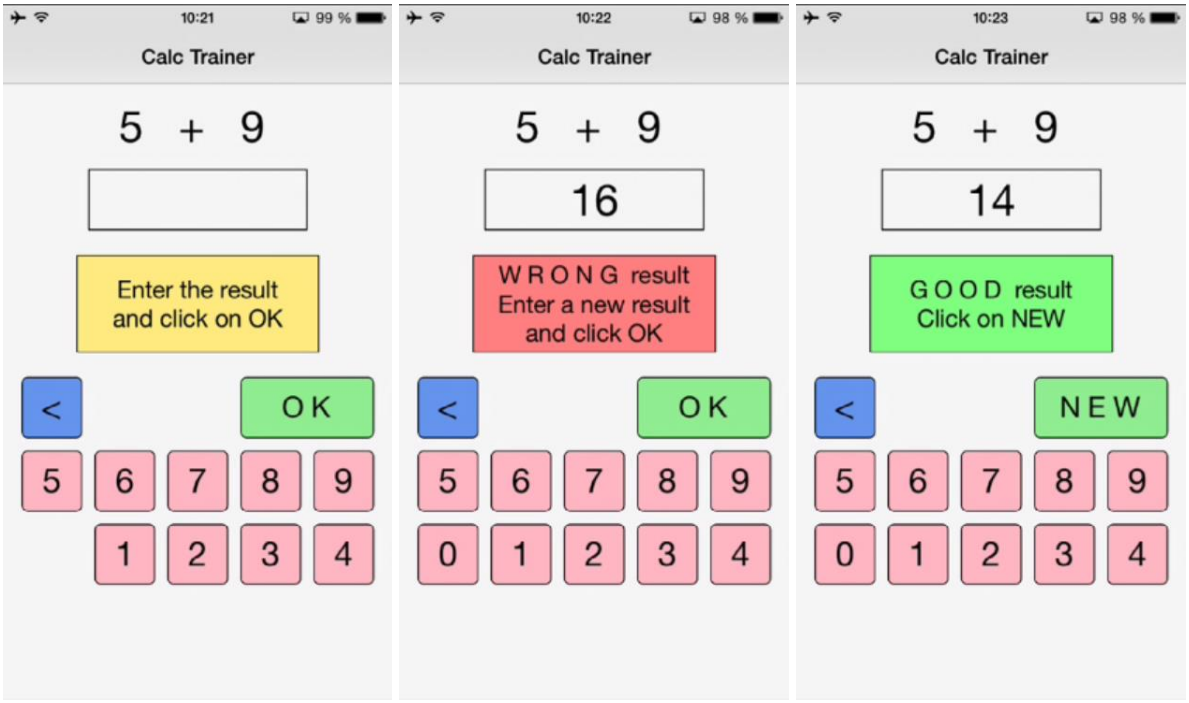
We first modify the New routine, where we add this line
`lblComments.Color = Colors.RGB(255,235,128)`

```
Private Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   lblComments.Color = Colors.RGB(255,235,128)      ' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
End Sub
```

And in the CheckResult routine we add the two lines with `lblComments.Color =...`

```
Private Sub CheckResult
   If lblResult.Text = Number1 + Number2 Then
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      lblComments.Color = Colors.RGB(128,255,128)       ' light green color
      btnAction.Text = "N E W"
   Else
      lblComments.Color = Colors.RGB(255,128,128)       ' light red color
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
   End If
End Sub
```

And we give the program a more meaningful title by adding `Page1.Title = "Calc Trainer"` in Application_Start just before `NavControl.ShowPage(Page1)`.



Another improvement would be to hide the '0' button to avoid entering a leading '0'.
For this, we hide the button in the New subroutine with line `btn0.Visible = False`.

```
Private Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   lblComments.Color = Colors.RGB(255,235,128)      ' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
   btn0.Visible = False
End Sub
```

We see that `btn0` is in red, this means that this object is not recognized by the IDE.

```
   btn0.Visible = False
```

So we must declare it, by adding `btn0` into line 17:

```
   Private btnAction, btn0 As Button
```

Now `btn0` is no more in red.

```
btn0.Visible = False
```

In addition, in the btnEvent_Click subroutine, we hide the button if the length of the text in lblResult is equal to zero and show it if the length is greater than zero.

```
Private Sub btnEvent_Click
   Dim btnSender As Button

   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & Send.Tag
   End Select

   If lblResult.Text.Length = 0 Then
      btn0.Visible = False
   Else
      btn0.Visible = True
   End If
End Sub
```

4 Getting started B4J

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

The B4J language is like Visual Basic and B4A language.

B4J includes a powerful GUI designer, built-in support for multiple screens and orientations.

What you need:

- The B4J program, this is a Windows program running on a PC.
- The Java SDK on the PC, free.

4.1 Installing B4J

4.1.1 Installing Java JDK

B4J depends on the free Java JDK component.

If you are already using B4A or B4i you can skip this chapter.

Installation instructions:

The first step should be to install the **Java JDK**, as B4J requires it.
Note that there is no problem with having several versions of Java installed on the same computer.

- Open the Java 8 JDK download link.
- Check the Accept License Agreement radio button.
- Select **"Windows x86"** or **"Windows x64"** (for 64 bit machines) in the platforms list.
- Download the file and install it.

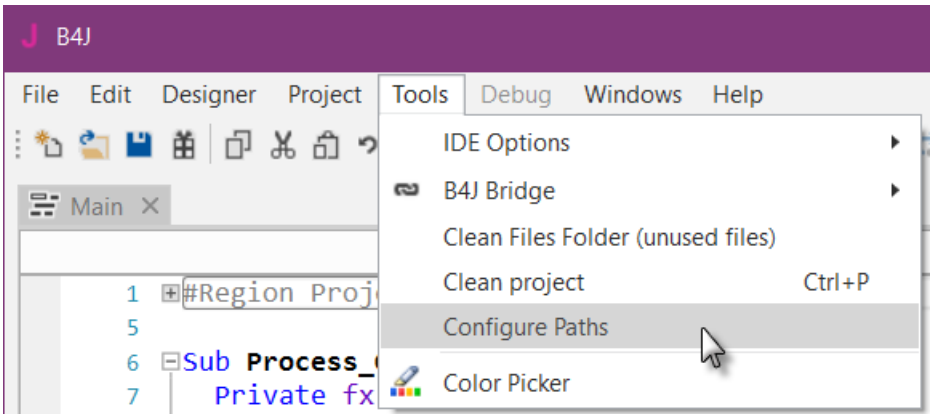
4.1.2 Installing B4J

Download and install the B4J file on your computer.

As B4J is for free, there is no license file needed like for B4A and B4i.

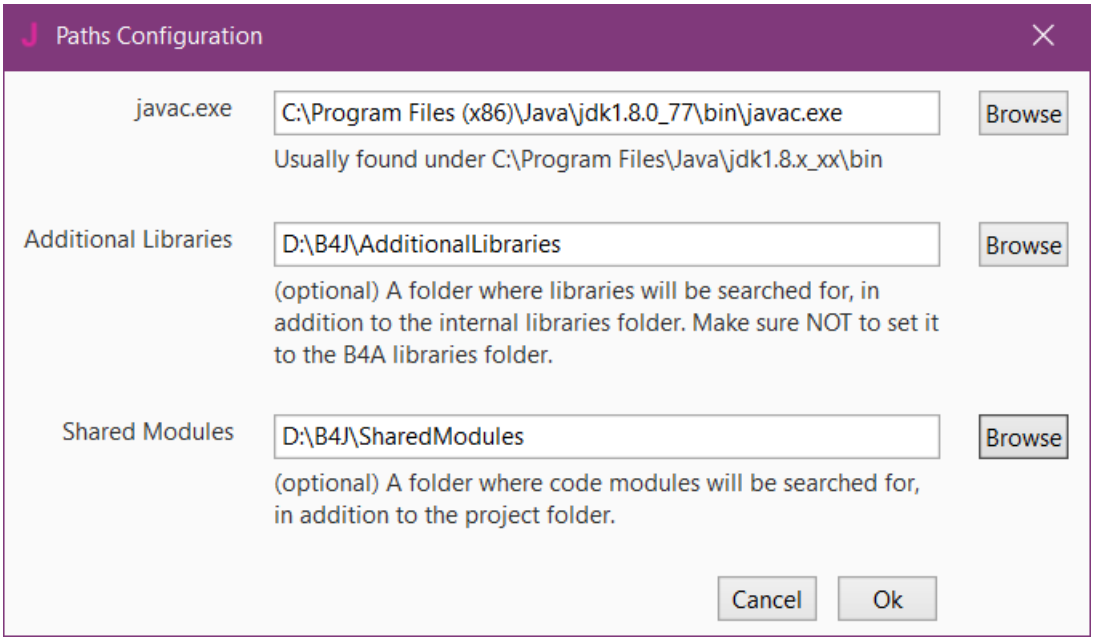
4.2 Configure Paths in the IDE

Then you need to configure the different paths in the IDE.



Run the IDE.

In the Tools menu click on Configure Paths.



javac.exe:
Enter the folder of the javac.exe file.

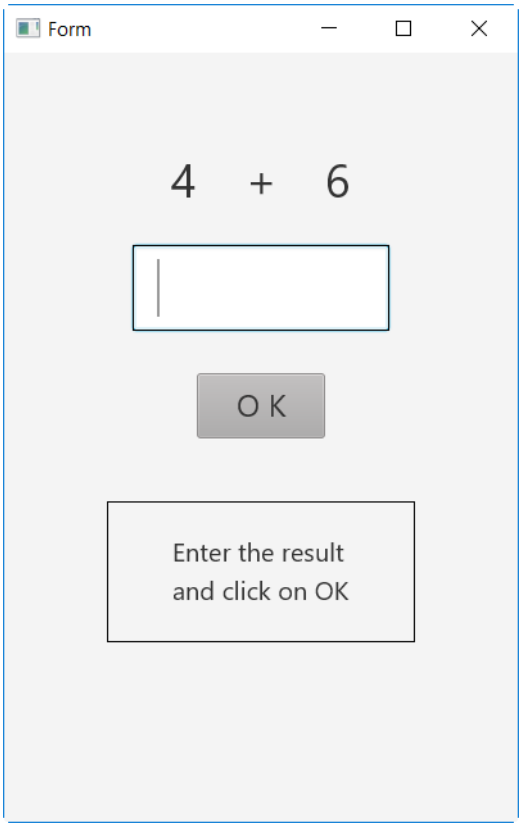
Additional libraries:
Create s specific folder for additional libraries, for example C:\B4J\AdditionalLibraries.

Shared Modules:
Create s specific folder for shared modules, for example C:\B4J\SharedModules.

4.3 My first program (MyFirstProgram.b4j)

Let us write our first program. The suggested program is a math trainer for kids.

The project is available in the SourceCode folder shipped with this booklet:
SourceCode\MyFirstProgram\B4J\MyFirstProgram.b4j



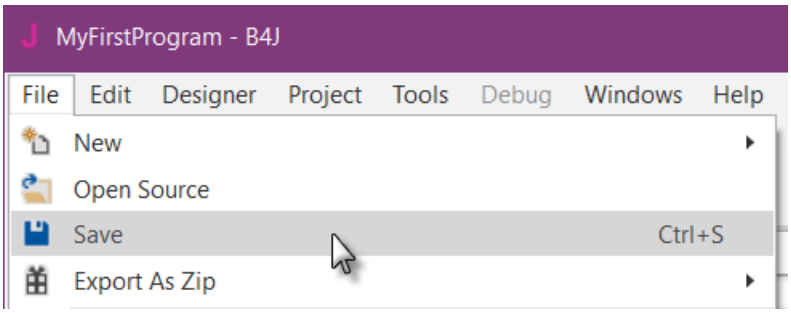
On the screen, we will have:
- 2 Labels displaying randomly generated numbers (between 1 and 9)
- 1 Label with the math sign (+)
- 1 TextField where the user must enter the result
- 1 Button, used to either confirm when the user has finished entering the result or generate a new calculation.
- 1 Label with a comment about the result.

In B4J:
- Label is an object to show text.
- TextField is an object allowing the user to enter text, like EditText in B4A.
- Button is an object allowing user actions.

We will design the layout of the user interface with the Designer, the Abstract Designer and a Form on the screen.
We will go step by step through the whole process.
The Designer manages the different objects of the interface.
The Abstract Designer shows the positions and sizes of the objects and allows moving or resizing them on the screen.
On the Form, we see the real result.

Run the IDE 

Save the project.

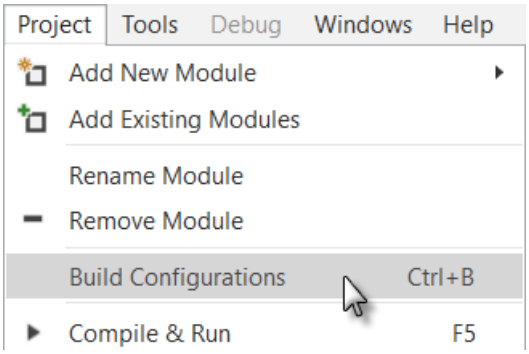


You must save the project before you can run the Designer.

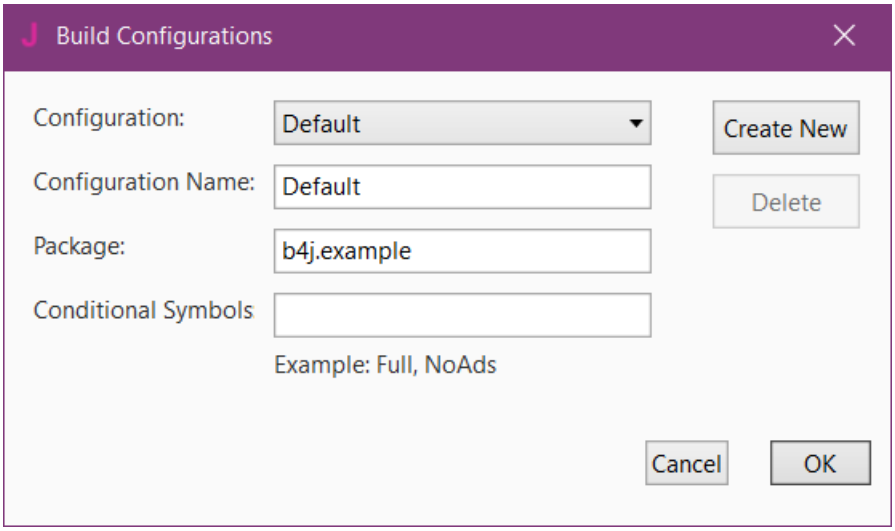
Create a new folder MyFirstProgram and save the project with the name MyFirstProgram.

Set the Package Name.
Each program needs a package name.

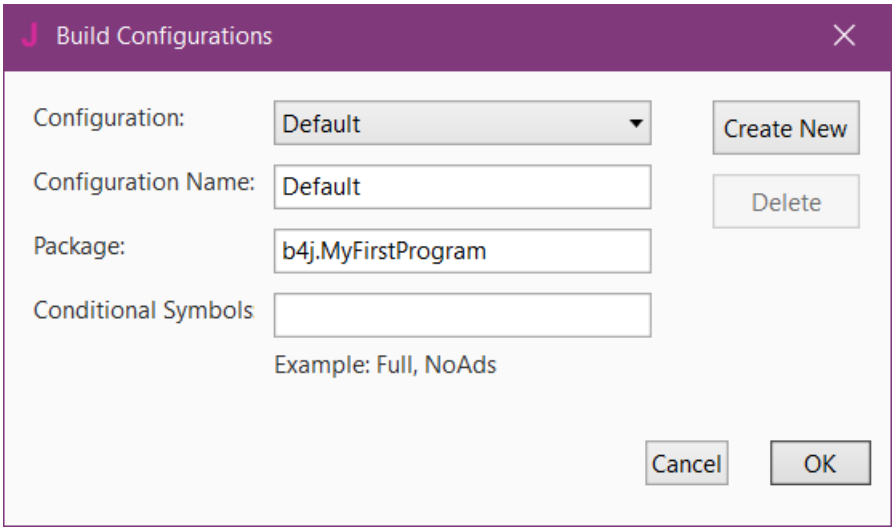
In the menu Project click on Build Configurations.



This window appears:



The default name is b4j.example. We will change it to b4j.MyFirstProgram.

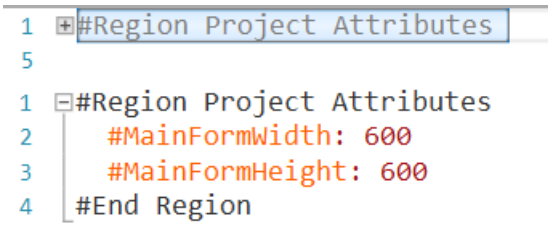


Set the Form size.

The Form size is the size of the main window, called Form, shown on the PC screen.

On top of the code you see `Region Project Attributes`.

Regions are code parts which can be collapsed or extended.
Clicking on ⊞ will expand the Region.
Clicking on ⊟ will collapse the Region.
Regions are explained in Collapse a Region.

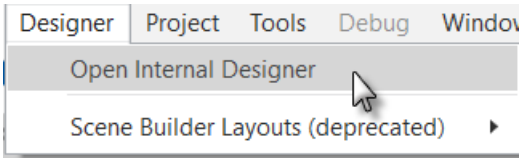


Here we can define the size of the project main window called Form.
The default values are Width = 600 and Height = 400.

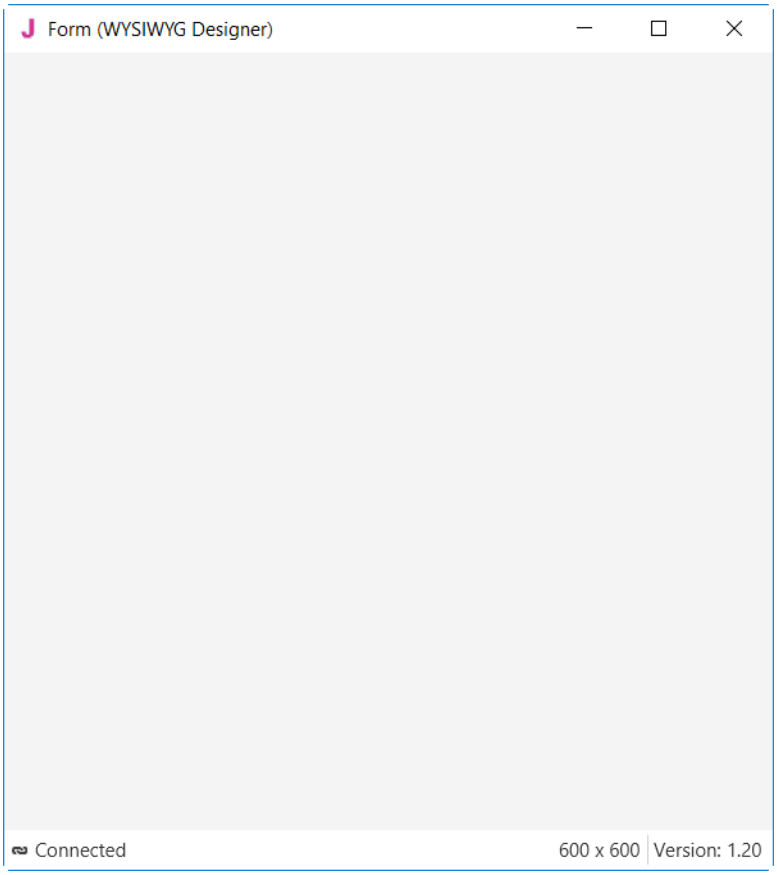
We change these to Width = 400 and Height = 600.

```
#Region Project Attributes
   #MainFormWidth: 400
   #MainFormHeight: 600
#End Region
```

In the IDE open the Designer.

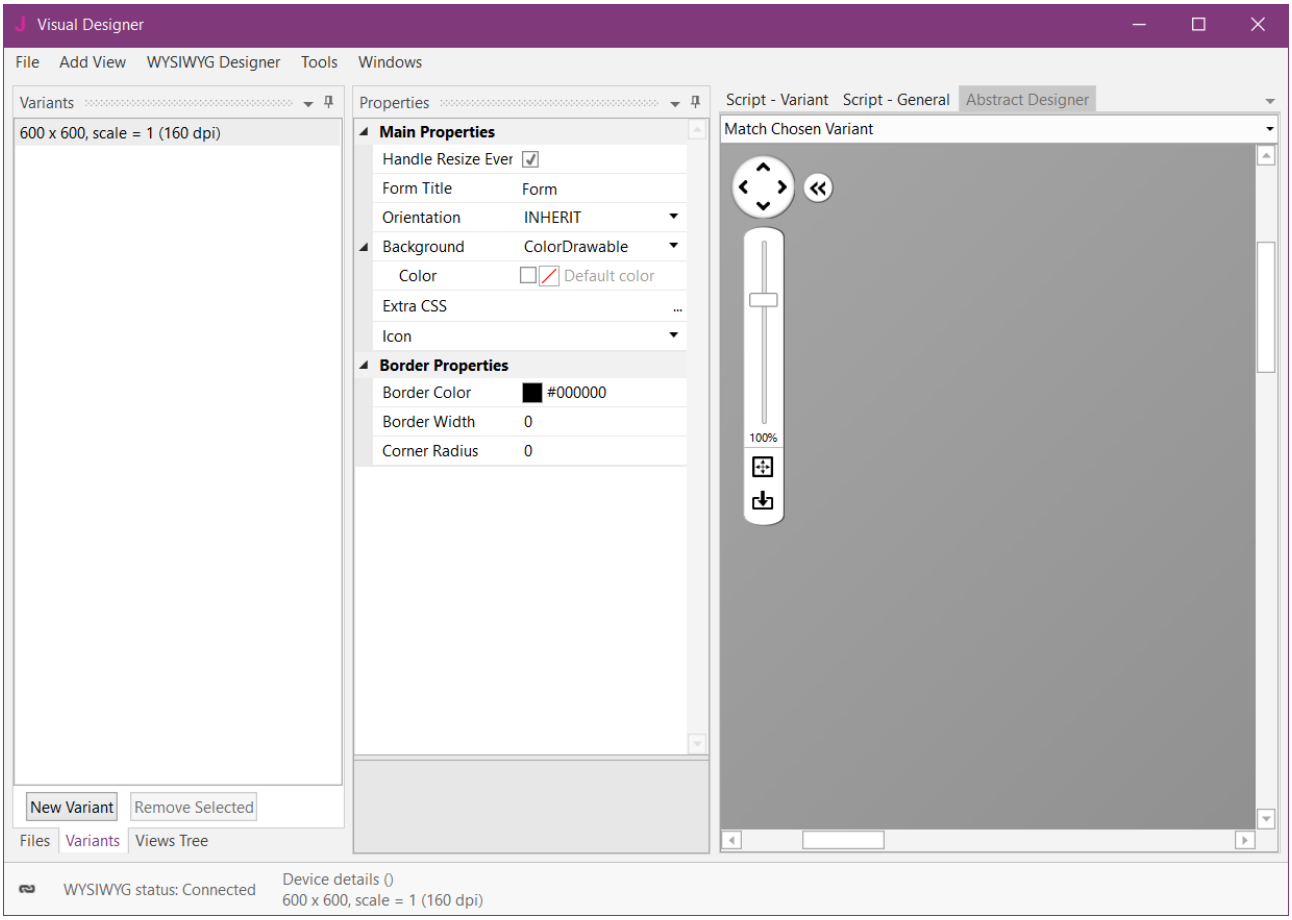


Wait until the Designer is ready.



We get a WYSIWYG Form.

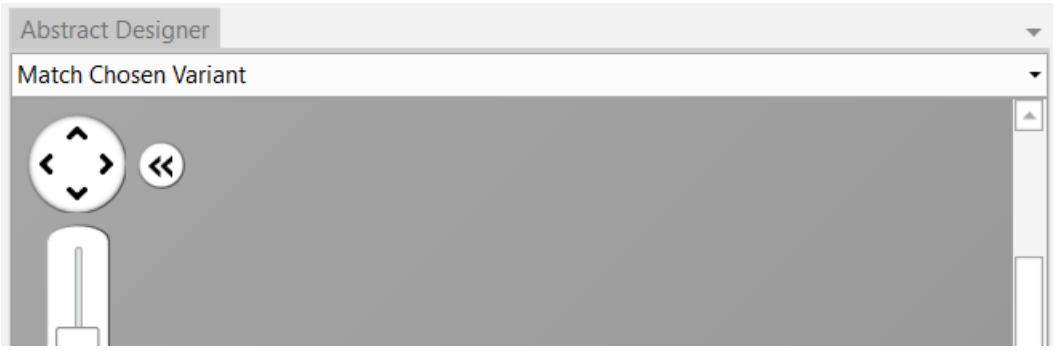
And the Designer looks like this.



Note that in the bottom left of the Designer window you see the connection status to the WYSIWYG Form:

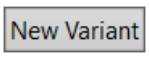
If you close the WYSIWYG Form you will see this status:

With the Designer, we have also the Abstract Designer which shows the layout not exactly WYSIWYG but the positions and size of the different objects.
Only the top of the image is shown.

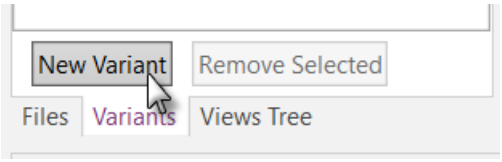


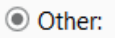
The dark gray area represents the screen area of the connected 'device' which is the WYSIWYG Form.

The default screen size variant is 600 * 600, we define a new variant with the same values, 400 * 600, as in the Project Attributes.

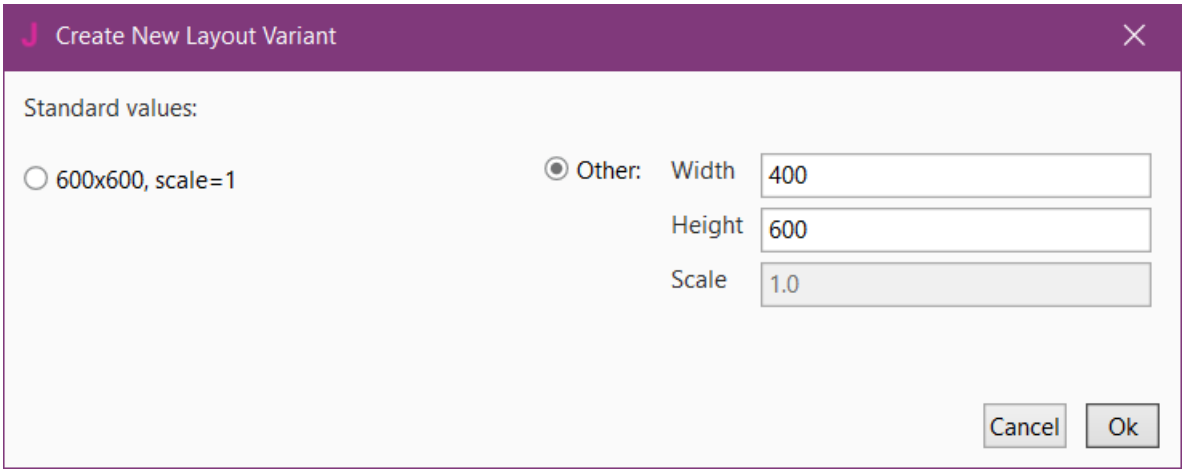
Click on 

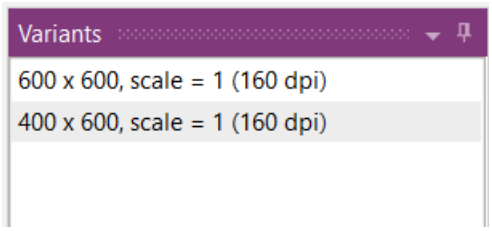
.



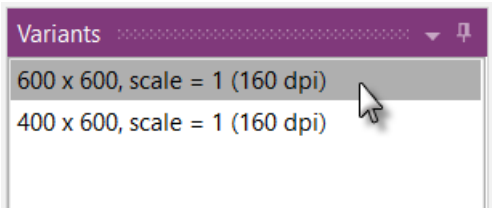
In this window click on 

and enter the two values.



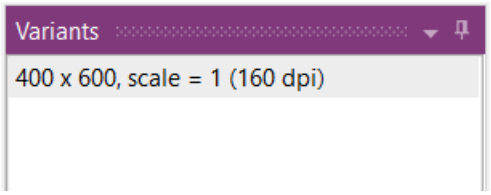


On top of the Variants window we see the new variant.

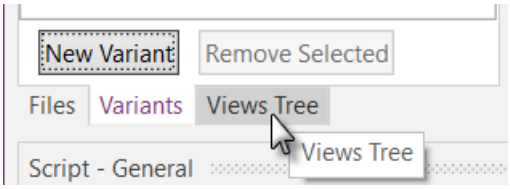


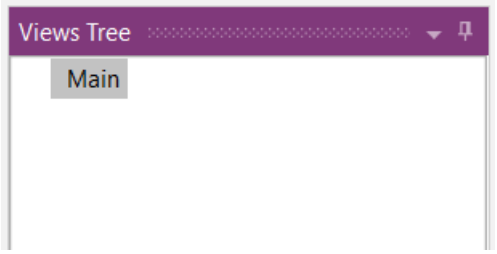
Click on 600 x 600, scale = 1 (160 dpi) to select the 600 * 600 variant.

Click on Remove Selected to remove the 600 * 600 variant.

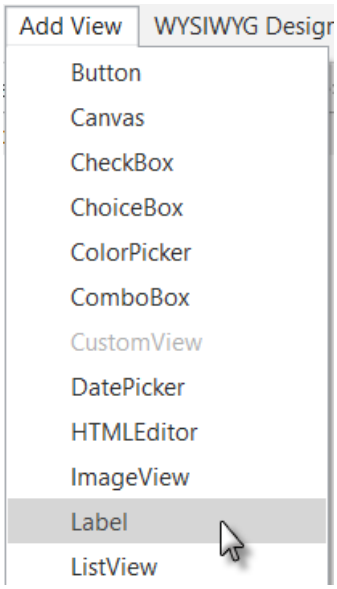


Click on Views Tree to show the Views Tree window.

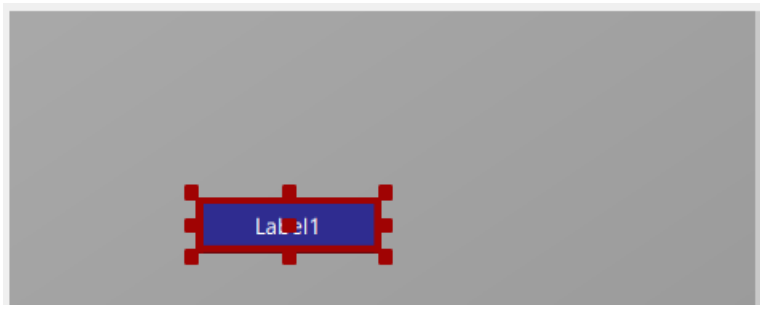




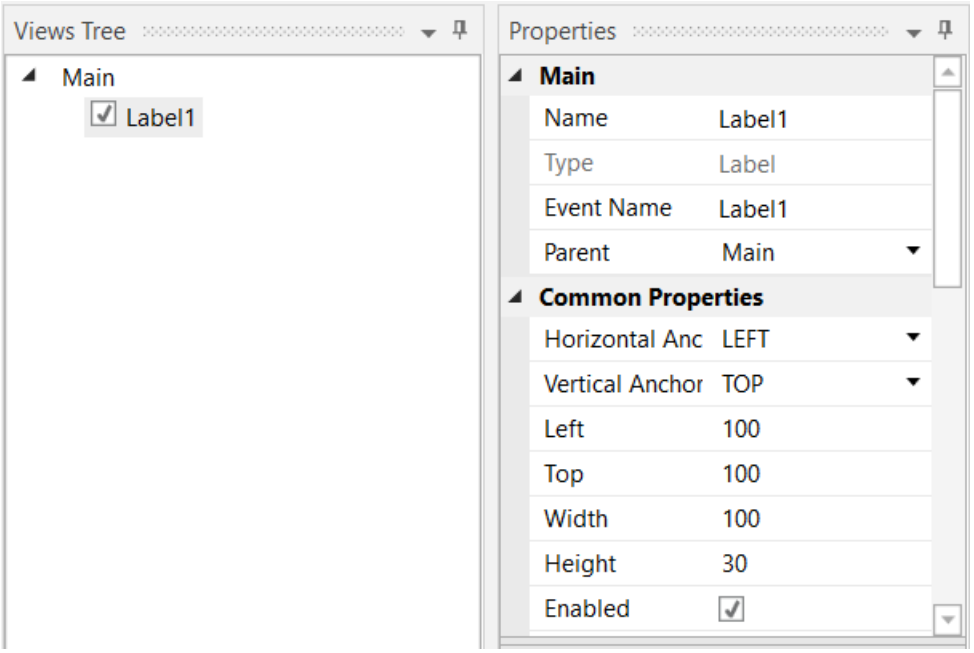
Now we will add the 2 Labels for the numbers.
In the Designer, add a Label.



In the IDE menu Add View click on Label.

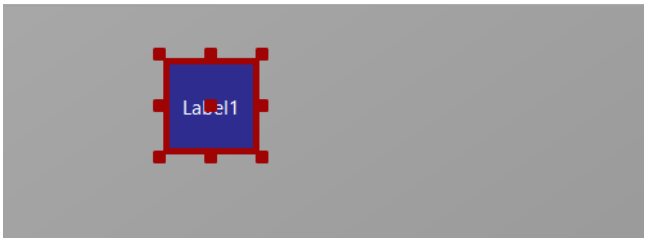


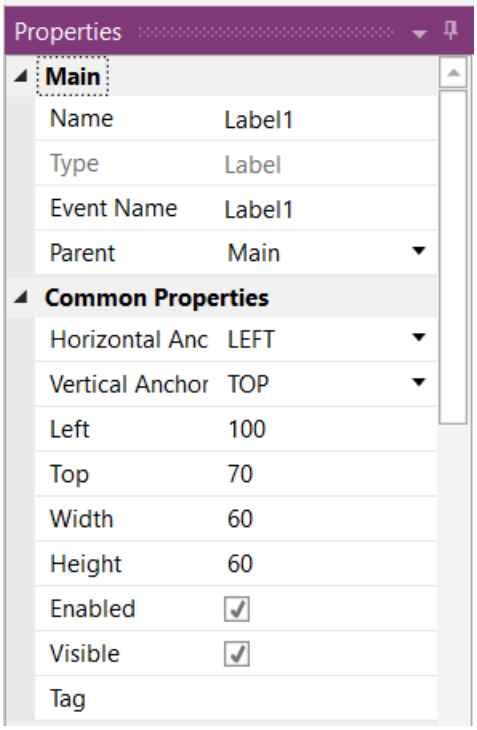
The Label appears in the Abstract Designer, in the Views Tree window and its default properties are listed in the Properties window.



And in the WYSIWYG Form.

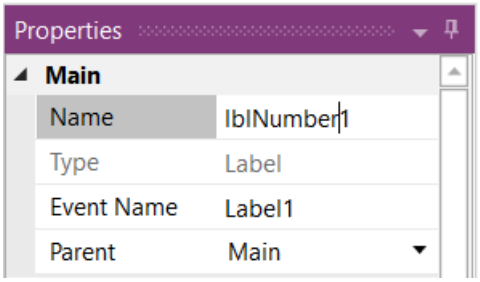
Resize and move the Label with the red squares like this.



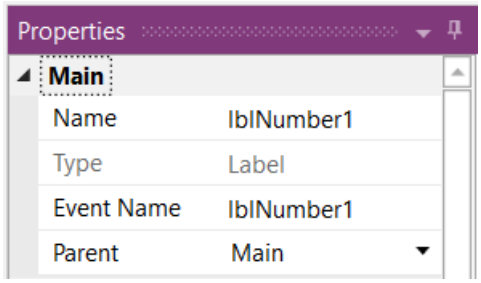


The new properties Left, Top, Width and Height are directly updated in the Properties window.
You can also modify the Left, Top, Width and Height properties directly in the Properties window.

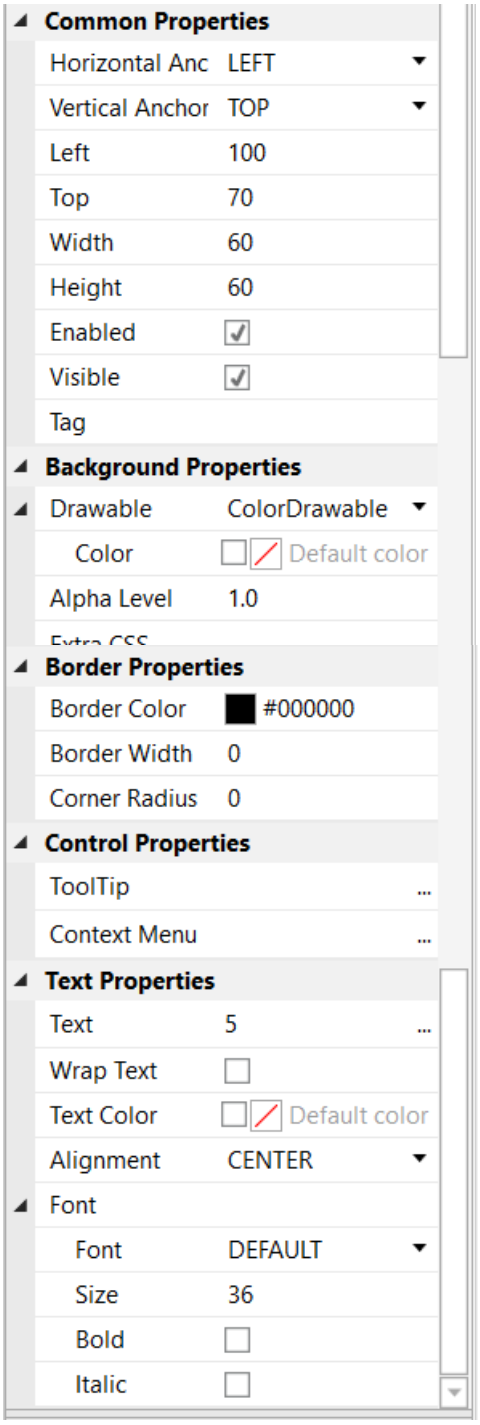
Let us change the properties of this first Label, per our requirements.
By default, the name is Label with a number, here Label1.
Let us change its name to lblNumber1.
The three letters 'lbl' at the beginning mean 'Label', and 'Number1' means the first number.
It is recommended to use significant names for nodes so we know directly what kind of node it is and its purpose.



Pressing the 'Return' key or clicking elsewhere will also change the Event Name property.



Main :	Main module.
Name :	name of the node.
Type :	type of the node. In this case, Label, which is not editable.
Event Name :	generic name of the routines that handle the events of the Label.
Parent :	parent node the Label belongs to.



Let us check and change the other properties:

Set Left, Top, Width and Height to the values in the picture.

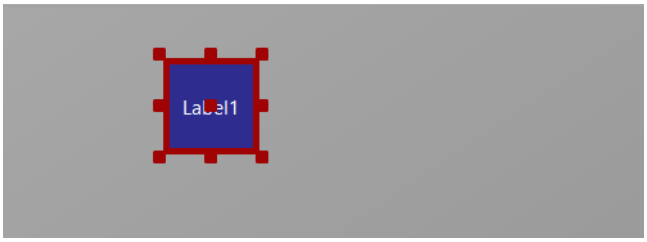
Visible is checked.

We leave the default colors.

Text set to 5

Set Text Alignment to Center.

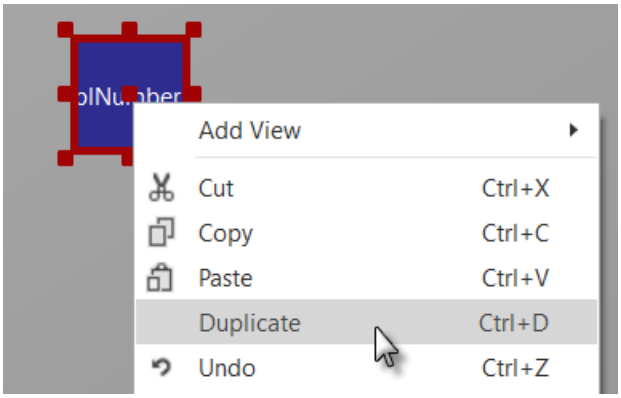
We leave the default Font.
Text Size, we set it to 36.



And the result in the Abstract Designer

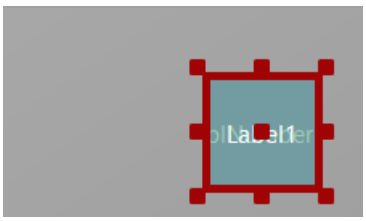
and on the WYSIWYG Form.

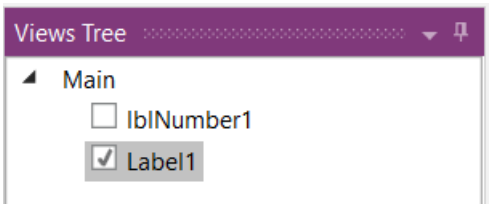
We need a second Label, like the first one. Instead of adding a new one, we copy the first one with the same properties. Only the Name and Left properties will change.



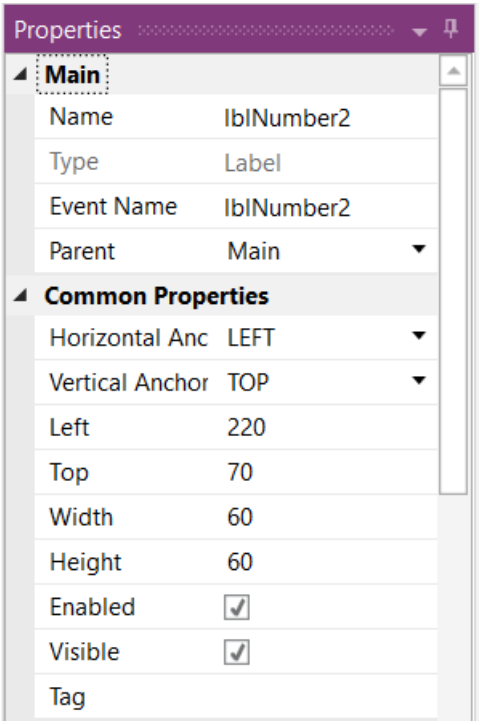
Right click on lblNumber1 and click on Duplicate in the popup menu.

The new label covers the previous one.





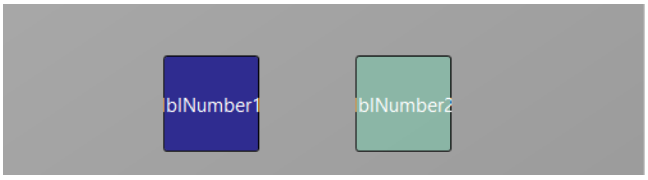
In the left part, in the Views Tree, you see the different nodes.
The new label Label1 is added.

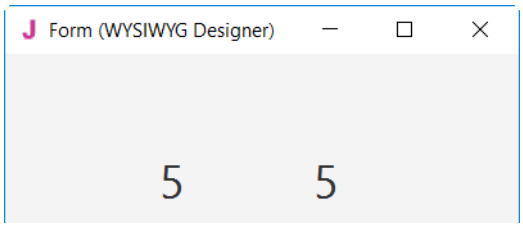


Let us position the new Label and change its name to lblNumber2.

Change the name to lblNumber2.

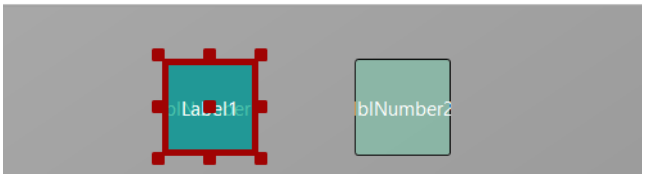
The Left property to 220.





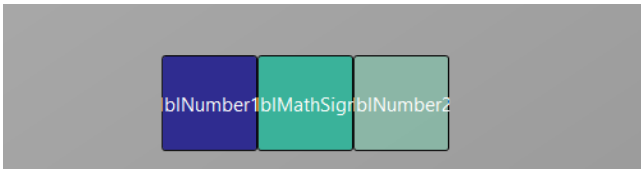
And the result in the Abstract Designer and on the WYSIWYG Form.

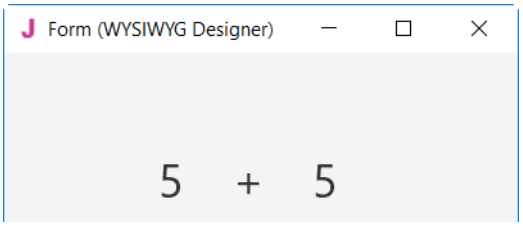
Let us now add a 3rd Label for the math sign. We copy once again lblNumber1.
Right click on lblNumber1 in the Abstract Designer and click on Duplicate in the popup menu.



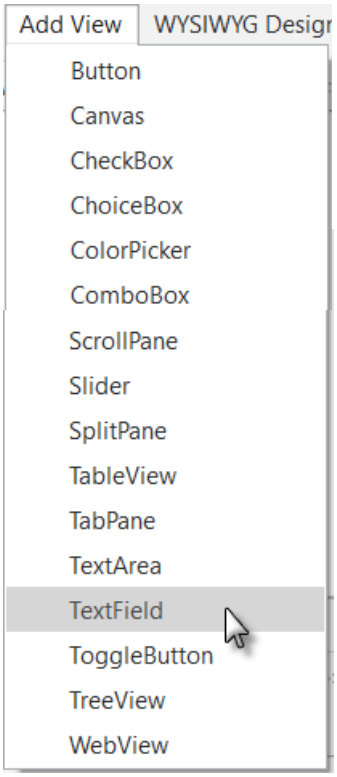
The new label covers lblNumber1.

Position it between the first two Labels and change its name to lblMathSign and its Text property to '+'.



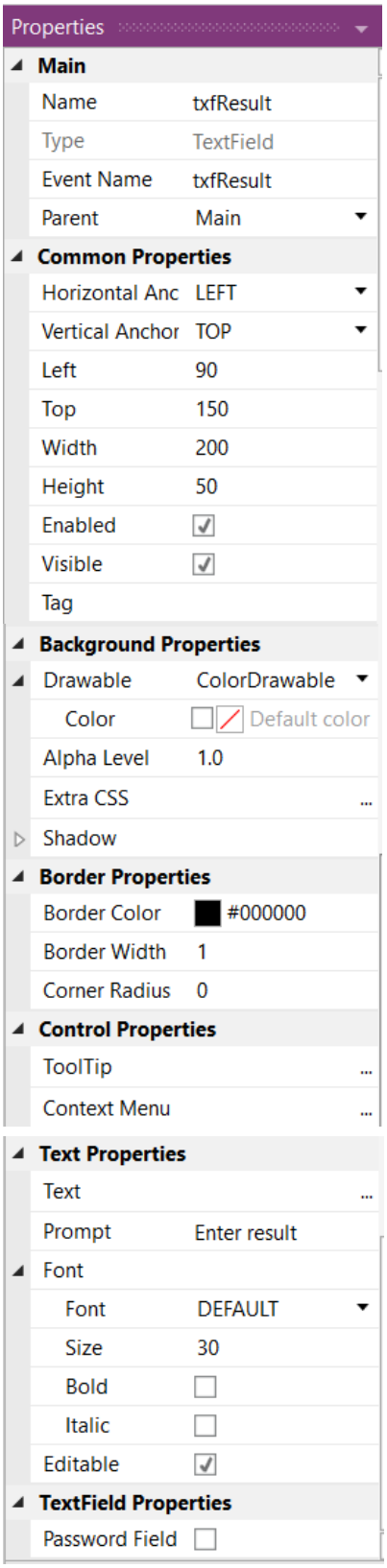


And the result in the Abstract Designer and on the WYSIWYG Form.



Now let us add a TextField node.
In the Designer Add View menu click on TextField.

Position it below the three Labels and change its name to txfResult.
'txf' means TextField and 'Result' for its purpose.



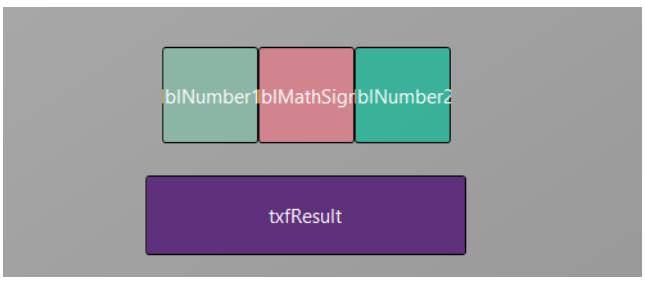
Change these properties.
Name to txfResult

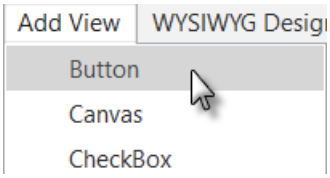
Left, Top, Width and Height.

Border Width to 1

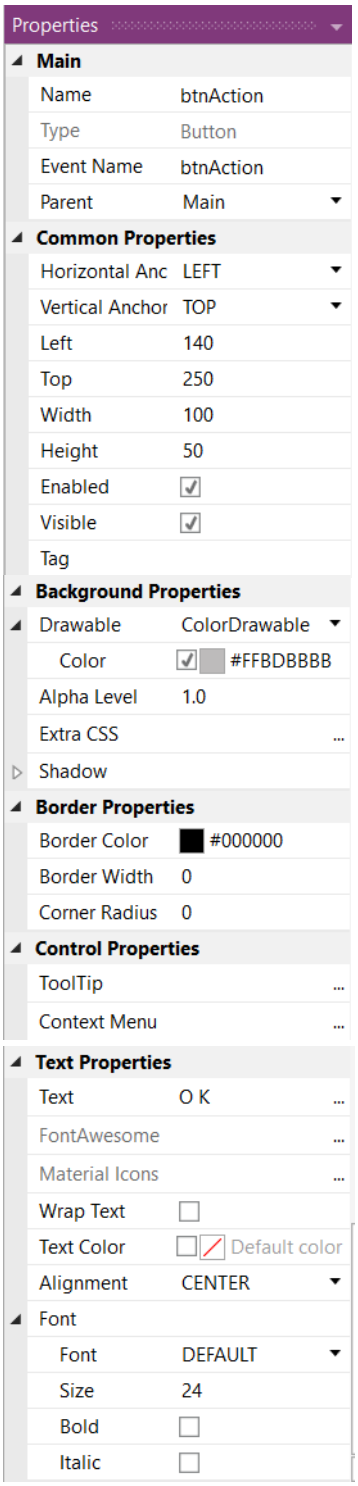
Prompt to Enter result
Prompt represents the text shown in the TextField node if no text is entered.
Size to 30

After making these changes, you should see something like this.





Now, let's add the Button which, when pressed, will either check the result the user supplied as an answer, or generate a new math problem, depending on the user's input.



Position it below the TextField node. Resize it and change following properties:

Name to btnAction

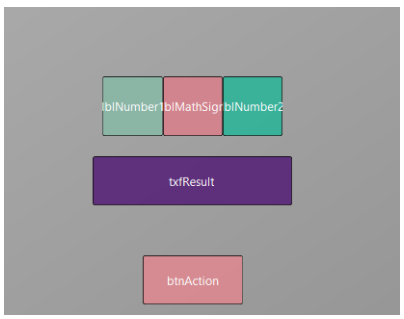
Left, Top, Width and Height.

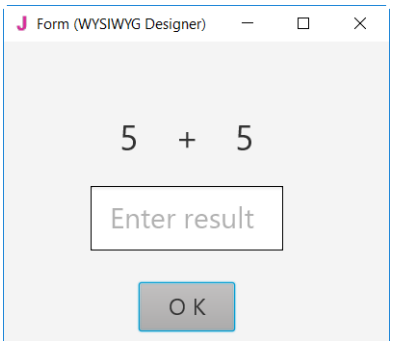
Background Color to #FFBDBBBB

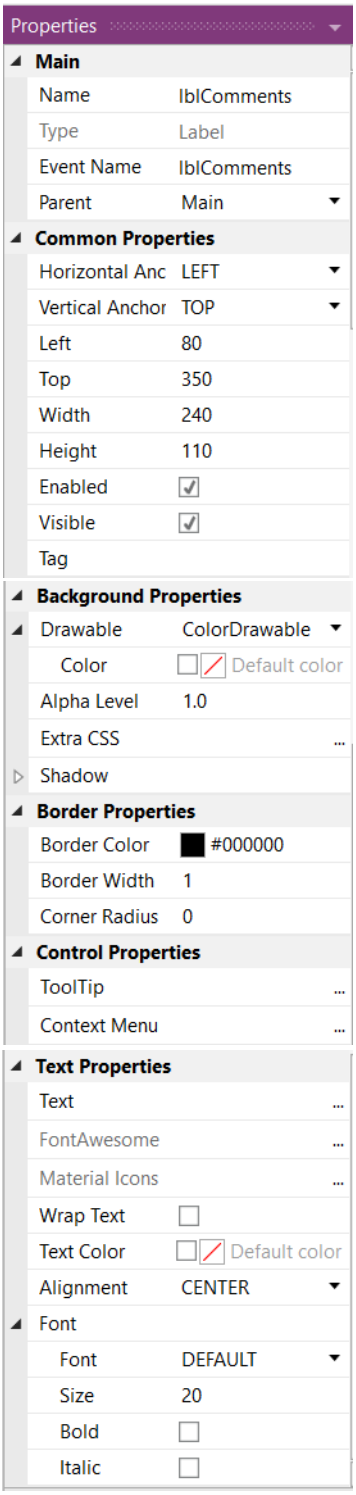
Text to O K (with a space between O and K)

Font Size to 24

Result (pictures reduced size)







Let us add the last Label for the comments. Position it below the Button and resize it.

Change the following properties:

Name to lblComments

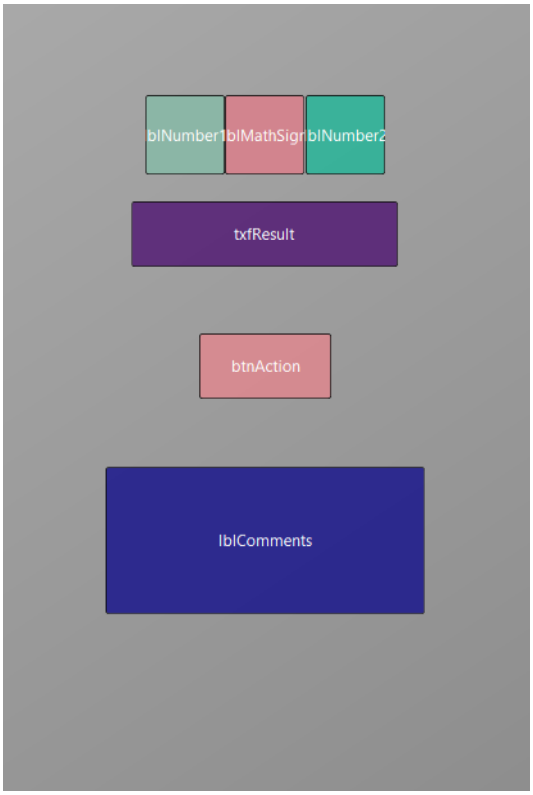
Left, Top, Width and Height.

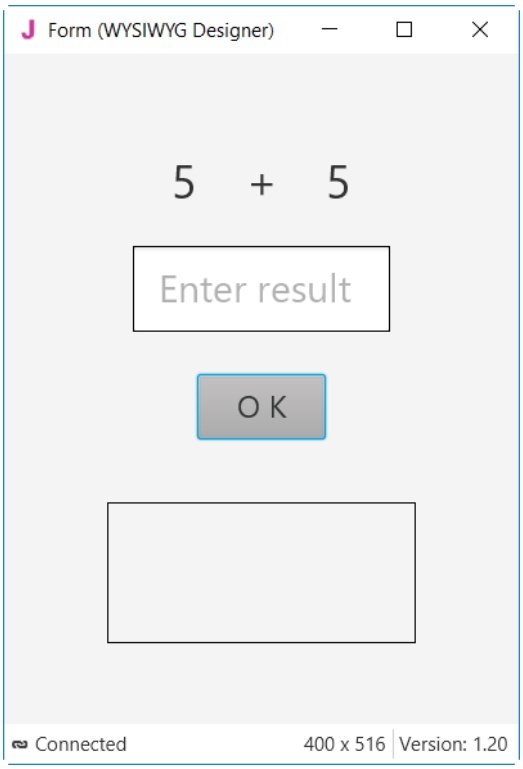
Border Width to 1

Text empty

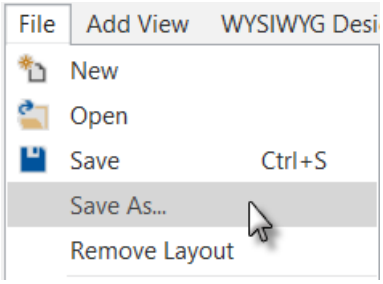
Font Size to 20

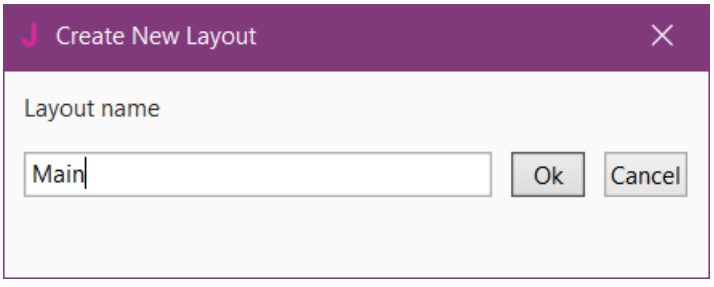
And the result.





Now we save the layout in a file.

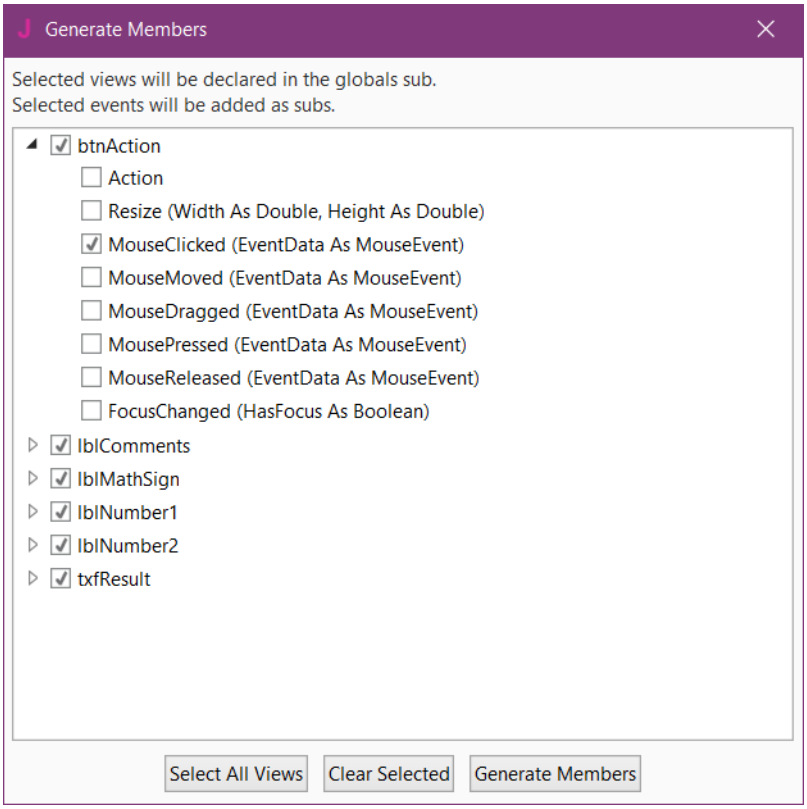
Click on  and save it with the name 'Main'.



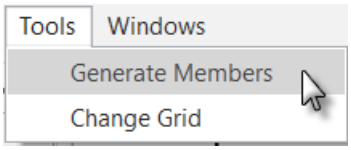
Click on  .

To write the routines for the project, we need to reference the Views in the code.
This can be done with the *Generate Members* tool in the Designer.

The *Generate Members* tool automatically generates references and subroutine frames.



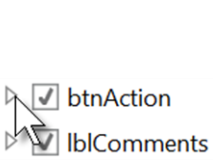
Click on



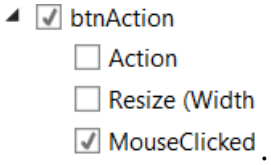
to open the generator.

Here we find all the nodes added to the current layout.
We check all nodes and check the MouseClicked event for the btnAction Button.
Checking a node  generates its reference in the Globals Sub routine in the code.
This is needed to make the node recognized by the system and allow the autocomplete function.

```
Private btnAction As Button
Private lblComments As Label
Private lblMathSign As Label
Private lblNumber1 As Label
Private lblNumber2 As Label
Private txfResult As TextField
```



Clicking on shows all events for the selected node



.

Clicking on an event of a node  generates the Sub frame for this event.

```
Sub btnAction_Click

End Sub
```

Click on  to generate the references and Sub frames, then close the window  .

Now we go back to the IDE to enter the code.

On the top of the program code we have:

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form
   Private btnAction As Button
   Private lblComments As Label
   Private lblMathSign As Label
   Private lblNumber1 As Label
   Private lblNumber2 As Label
   Private txfResult As TextField
End Sub
```

These lines are automatically in the project code.

```
   Private fx As JFX
   Private MainForm As Form
```

B4J needs a MainForm, and the JFX library for the nodes, details in chapter Process life cycle.

Below the code above we have the `AppStart` routine which is the first routine executed when the program starts.
The content below is also added automatically in each new project.

```
Sub AppStart (Form1 As Form, Args() As String)
   MainForm = Form1
   'MainForm.RootPane.LoadLayout("Layout1") 'Load the layout file.
   MainForm.Show
End Sub
```

`MainForm = Form1` > Sets Form1 to the variable MainForm.
`'MainForm.RootPane.LoadLayout("Layout1")` > Loads a layout file if needed.
`MainForm.Show` > Shows the MainForm

First, we need our program to load the layout file we defined in the previous pages.
The file must be loaded onto the MainForm.RootPane, we uncomment the line
`'MainForm.RootPane.LoadLayout("Layout1")`
and change the layout file name.

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

We want to generate a new problem as soon as the program starts. Therefore, we add a call to the New subroutine in AppStart.

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

   New
End Sub
```

New is displayed in red because the ‘New’ routine has not yet been defined.
Generating a new problem means generating two new random values between 1 and 9 (inclusive) for Number1 and Number2, then showing the values using the lblNumber1 and lblNumber2 ‘Text’ properties.

To do this we enter following code:
In Sub Process_Globals we add two variables for the two numbers.

```
   Private Number1, Number2 As Int
End Sub
```

And the 'New' Subroutine:

```
Private Sub New
   Number1 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)                ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1           ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2           ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   txfResult.Text = ""                 ' Sets edtResult.Text to empty
End Sub
```

The following line of code generates a random number from '1' (inclusive) to '10' (exclusive) :
`Rnd(1, 10)`

In this line `Number1 = Rnd(1, 10)  ' Generates a random number between 1 and 9`
The text after the quote, `' Generates...`, is considered as a comment.
It is good practice to add comments explaining the purpose of the code.

The following line displays the comment in the lblComment node:
`lblComments.Text = "Enter the result" & CRLF & "and click on OK"`
`CRLF` is the LineFeed character.

Now we add the code for the Button click event.

We have two cases:
- When the Button text is equal to "O K", it means that a new problem is displayed, and the program is waiting for the user to enter a result and press the Button.
- When the Button text is equal to "NEW", it means that the user has entered a correct answer and when the user clicks on the Button a new problem will be generated.

```
Private Sub btnAction_Click
   If btnAction.Text = "O K" Then
      If txfResult.Text="" Then
         lblComments.Text = "No result entered" & CRLF & "Enter a result" & CRLF & "and click on OK"
      Else
         CheckResult
      End If
   Else
      New
      btnAction.Text = "O K"
   End If
End Sub
```

`If btnAction.Text = "O K" Then` checks if the Button text equals "O K".
If yes, we check if the TextField is empty.
    If yes, we display a message in the comment field telling the user that there is no result in the TextField node.
    If no, we check if the result is correct or if it is wrong.
If no, we generate a new problem, set the Button text to "O K" and clear the TextField node.

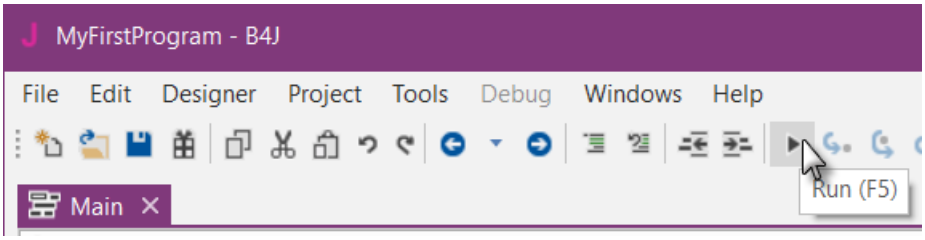
The last routine checks the result.

```
Private Sub CheckResult
   If txfResult.Text = Number1 + Number2 Then
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      btnAction.Text = "N E W"
   Else
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
   End If
End Sub
```

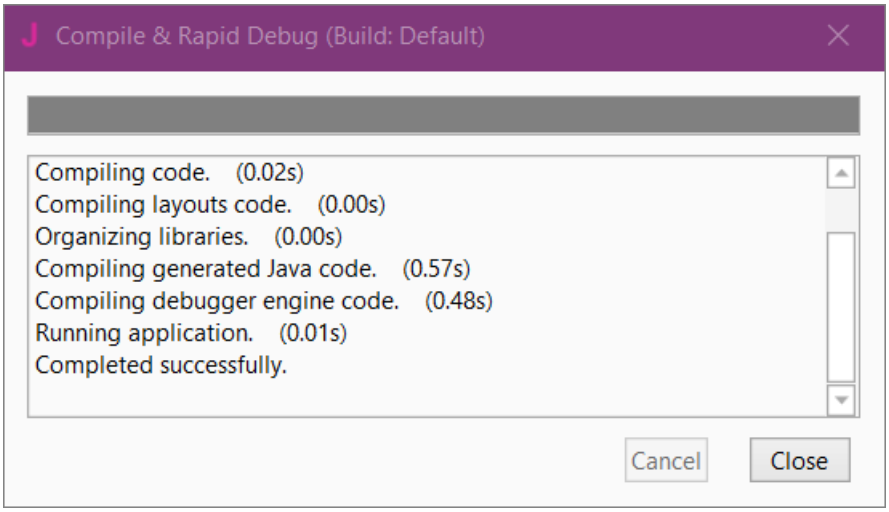
With `If txfResult.Text = Number1 + Number2 Then` we check if the entered result is correct.

If yes, we display in the lblComments label the text below:
        'G O O D result'
        'Click on NEW'
        and we change the Button text to "N E W ".
If no, we display in the lblComments label the text below:
        W R O N G result
        Enter a new result
        and click OK

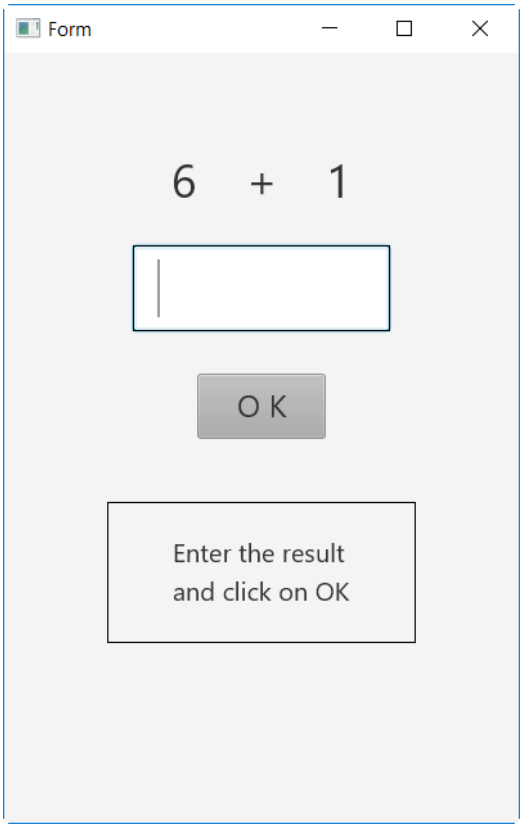
Let us now compile and run the program.
In the IDE on top click on  :



The program is going to be compiled.

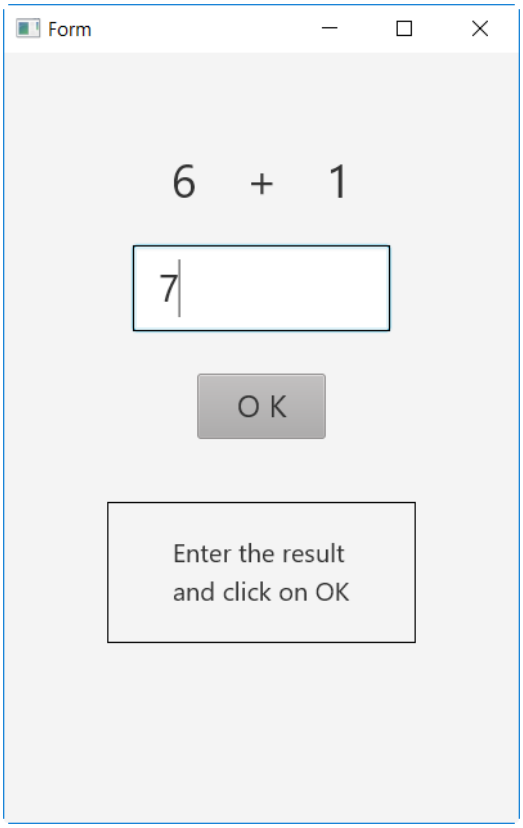


When you see `'Completed successfully.'` as in the message box, the compiling and transfer is finished.



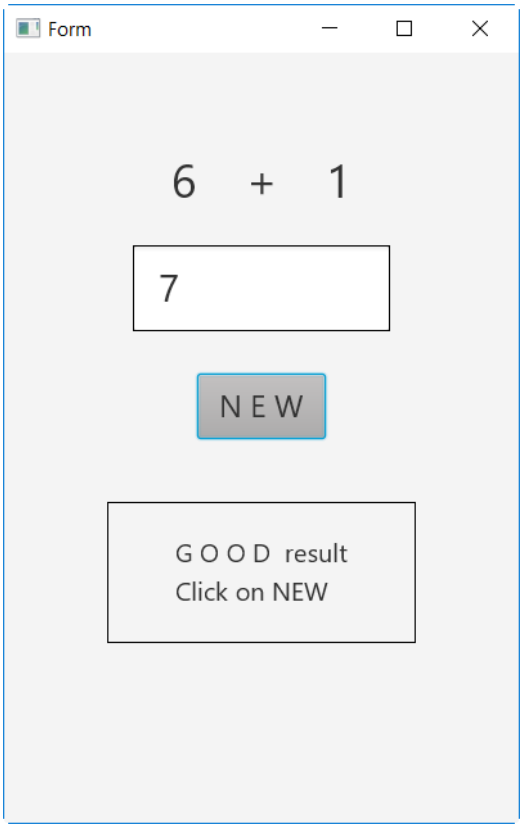
Then you should see something like the image on the left, with different numbers.

Of course, we could make aesthetic improvements in the layout, but this was not the main issue for the first program.



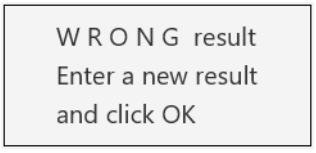
Enter 7

Click on  to confirm the result entry.



If the result is correct you will see the screen on the left.

If the result is wrong the message is:



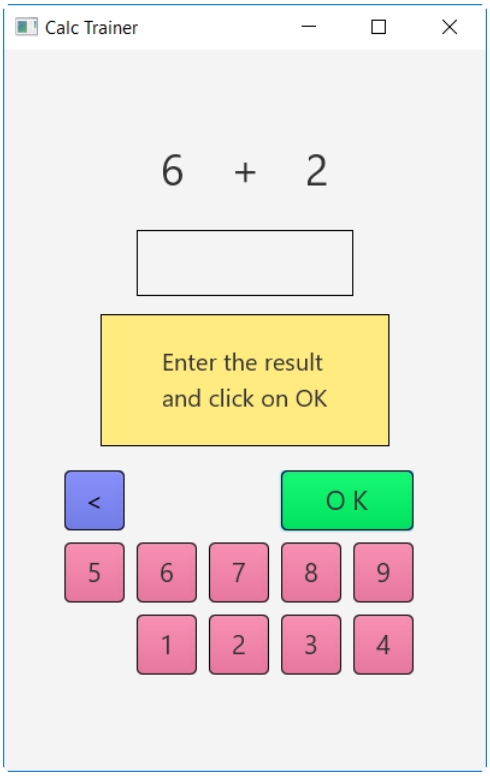
Click on  to define a new problem.

4.4 Second B4J program (SecondProgram.b4j)

The project is available in the SourceCode folder: SourceCode\SecondProgram\SecondProgram.b4j.

Improvements to “My first program”.
- Independent numeric keyboard to avoid the use of the PC keyboard.
- Colors in the comment label.

Create a new folder called “SecondProgram”. Copy all the files and folders from MyFirstProgram to the new SecondProgram folder and rename the program file MyFirstProgram.b4j to SecondProgram.b4j and MyFirstProgram.b4j.meta to SecondProgram.b4j.meta.

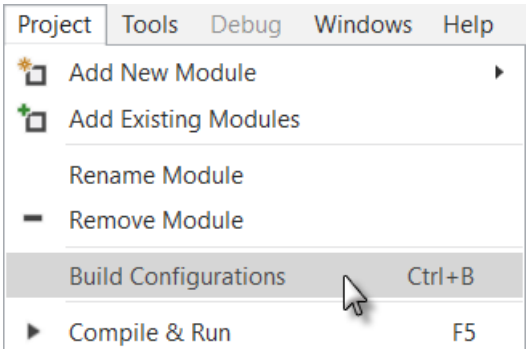


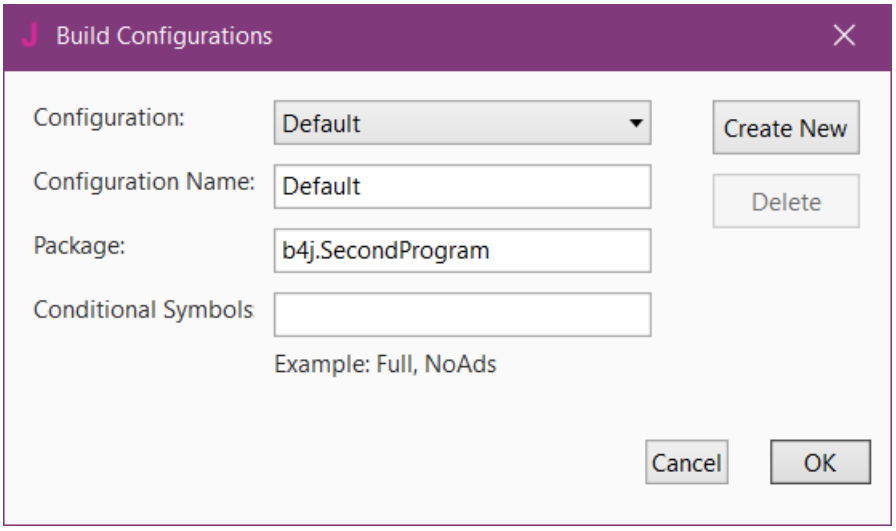
Load this new program in the IDE.

Run the Designer.

We need to change the Package Name.

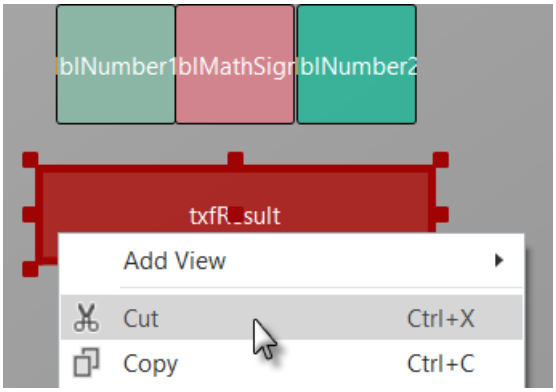
In the IDE Project menu.
Click on Build Configurations.



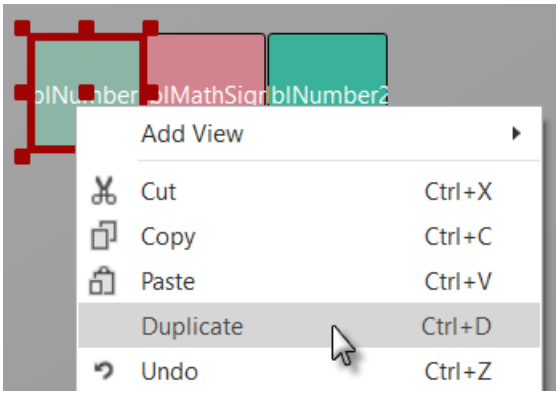


Change the Package name to b4j.SecondProgram and click on OK.

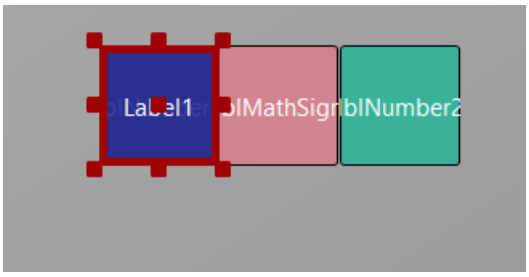
We want to replace the txfResult TextField node by a new Label.
In the Abstract Designer, click on the txfResult node.



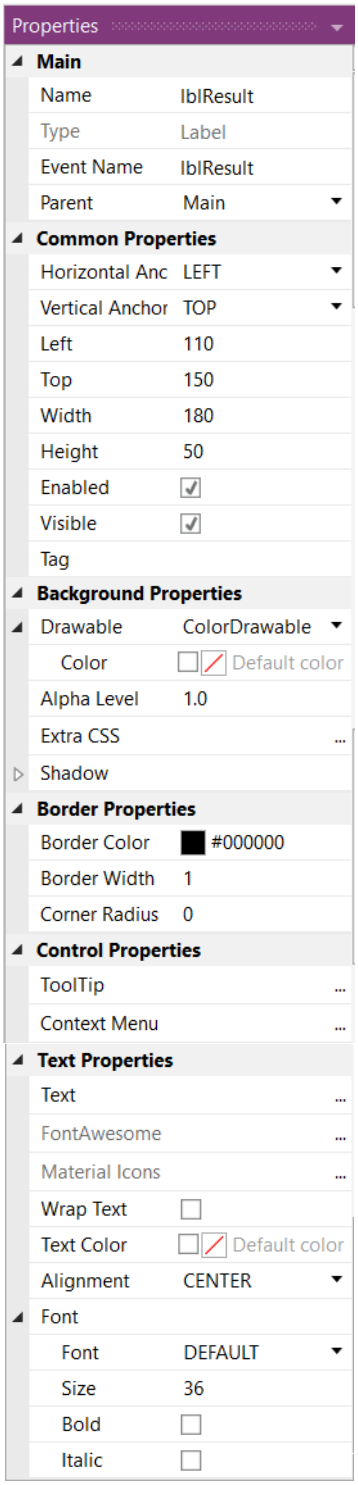
Right click on txfResult and click on Cut.



Right click on lblNumber1 and click on Duplicate.



The new label covers lblNumber1.



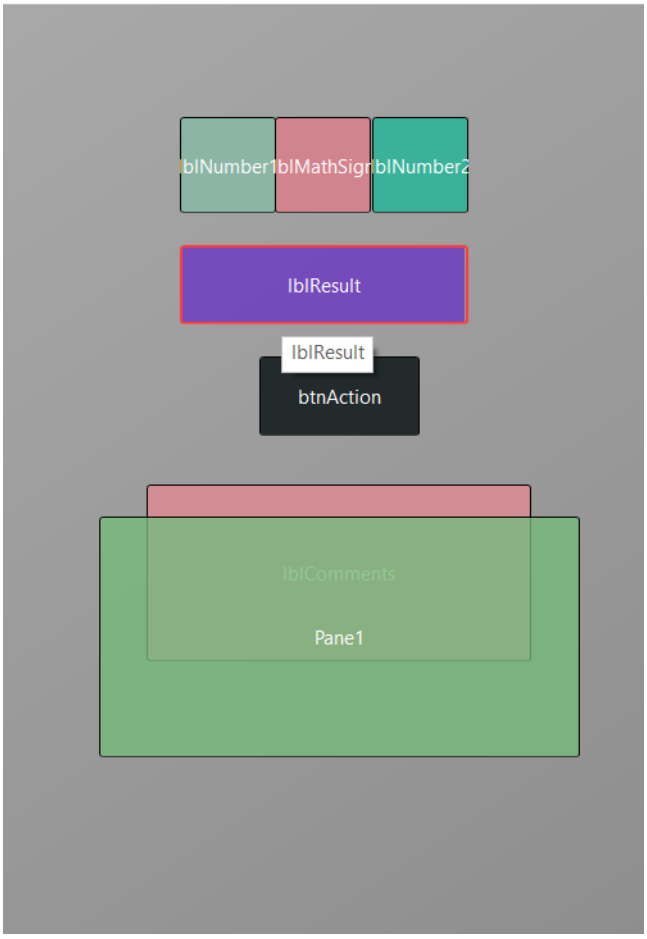
Modify the following properties:

Name to lblResult

Left, Top, Width, Height

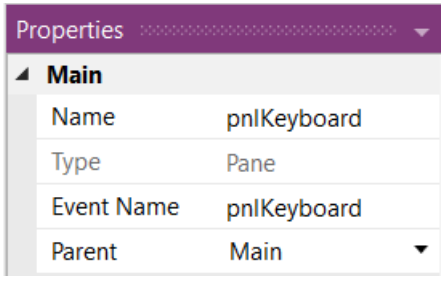
Border Width to 1

Text to "" no character

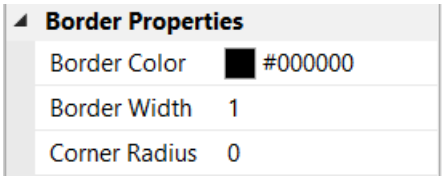


Let us add a Pane for the keyboard buttons.

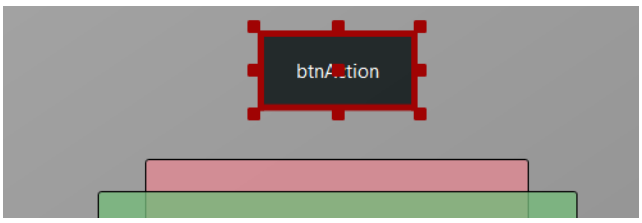
Position and resize it as in the image.



Change its Name to pnlKeyboard
"pnl" for Pane (B4A habit pnl for Panel), the node type.

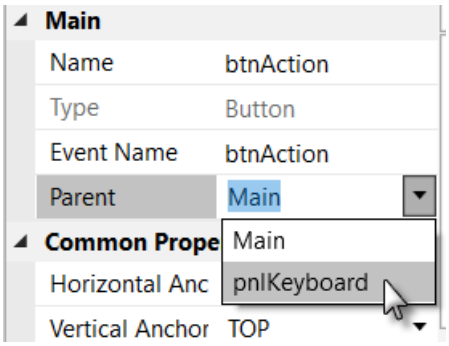


Change
Corner radius to 0

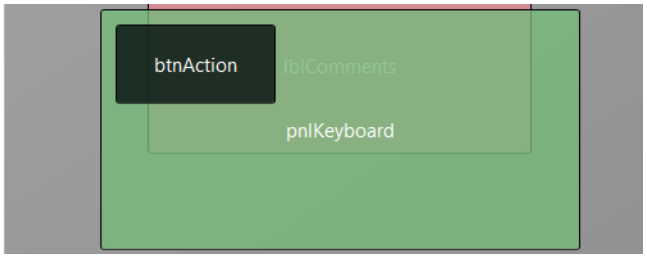


We will move btnAction from Main to the pnlKeyboard Panel.

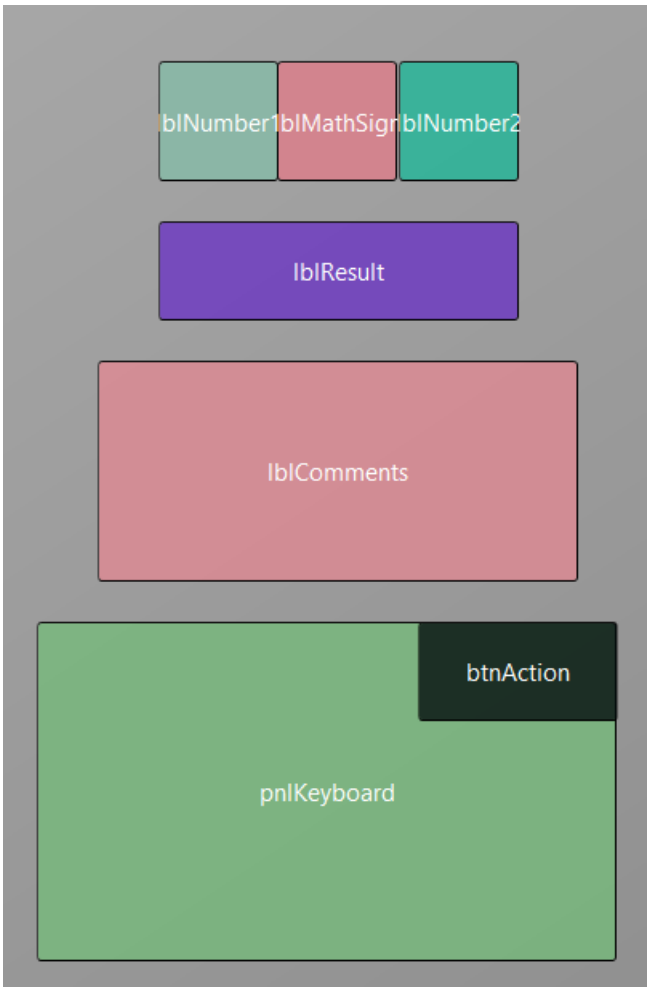
Click on btnAction.



In the Parent list click on pnlKeyboard.



The button now belongs to the Pane.

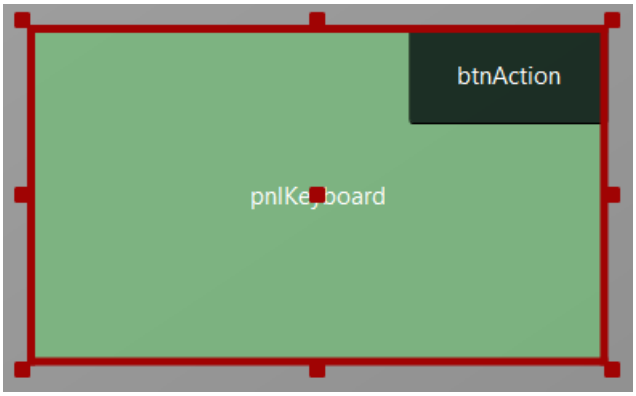


Now we rearrange the nodes to get some more space for the keyboard.

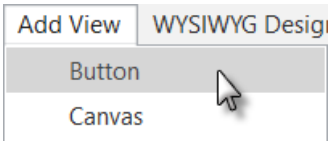
Set the properties below:

lblComments	Top = 220
pnlKeyboard	Left = 30
pnlKeyboard	Top = 350
pnlKeyboard	Width = 290
pnlKeyboard	Height = 170
pnlKeyboard	BorderWidth = 0

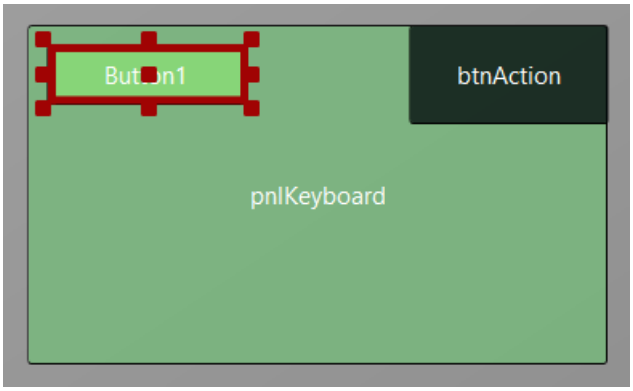
Move btnAction to the upper right corner of pnlKeyboard.



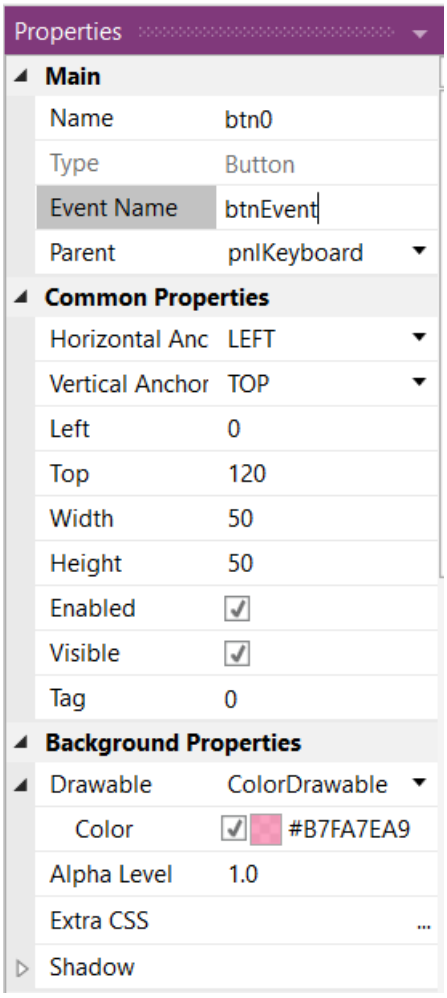
Click on the pnlKeyboard panel to select it.

Click on 

to add a new button.



The new button is added.



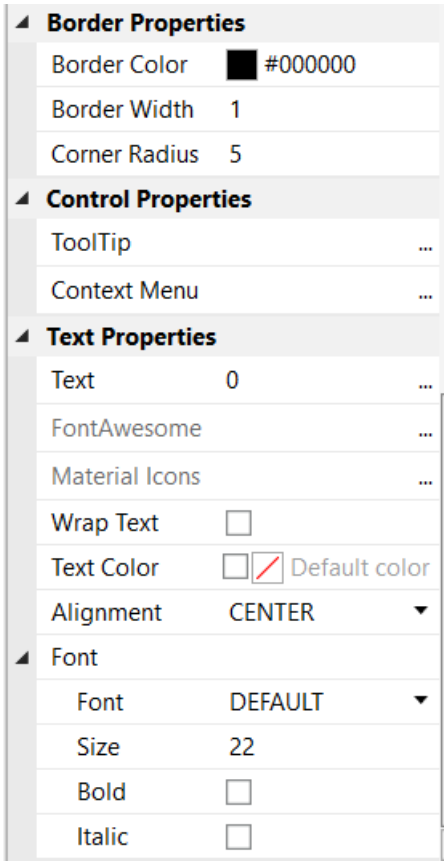
Change following properties:

Name to btn0
Event name to btnEvent

Left to 0
Top to 120
Width to 50
Height to 50

Tag to 0

Background Color to #B7FA7EA9

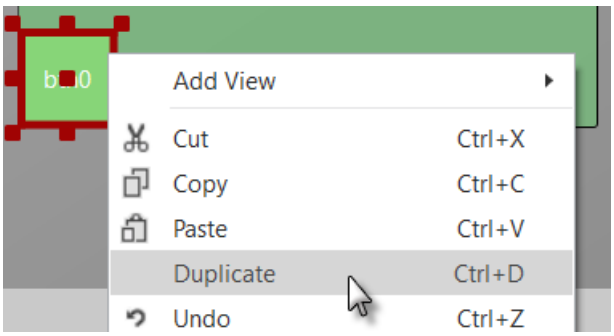


Border Width to 1
Corner Radius to 5

Text to 0

Size to 22

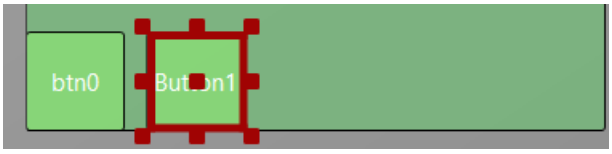
The button looks now like this.



Let us duplicate btn0 and position the new one beside button btn0.

Select the Button btn0.

Right click on btn0
and click on Duplicate.



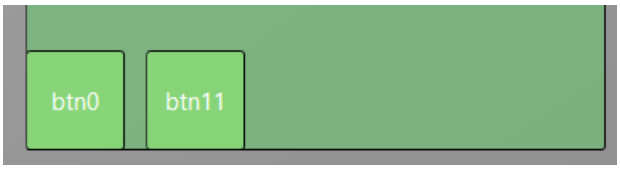
Move the new Button next to the previous one with a space.

Main	
Name	btn11
Tag	1
Text	1 ...

Change the following properties:
Name to btn1

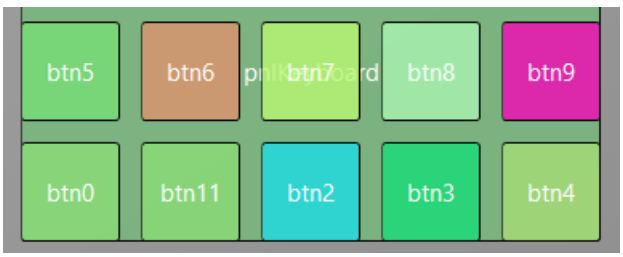
Tag to 1

Text to 1



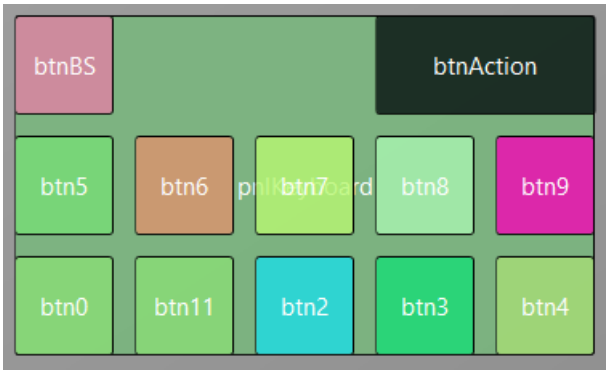
And the result.

Add 8 more Buttons and position them like in the image.



Change following properties:

Name	btn2,	btn3,	btn4	etc.
Tag	2 ,	3 ,	4	etc.
Text	2 ,	3 ,	4	etc.

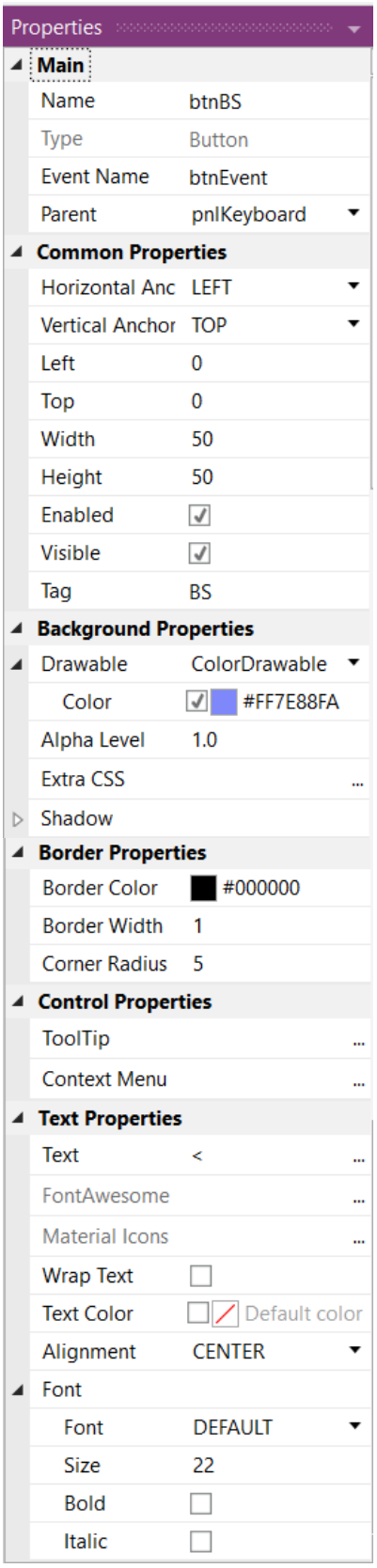


To create the BackSpace button, duplicate one of the number buttons, and position it in the top left corner.

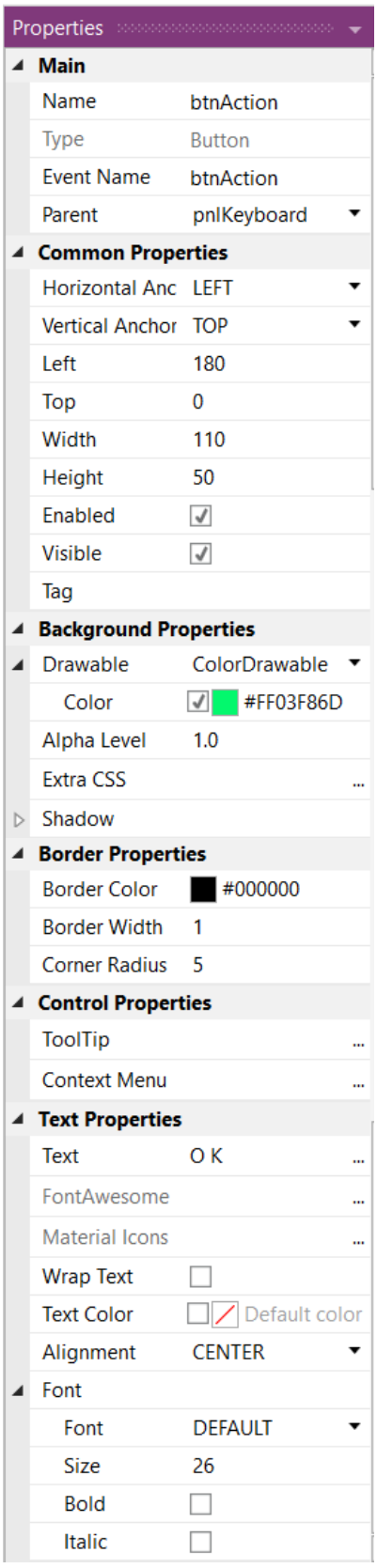
Resize and position btnAction.

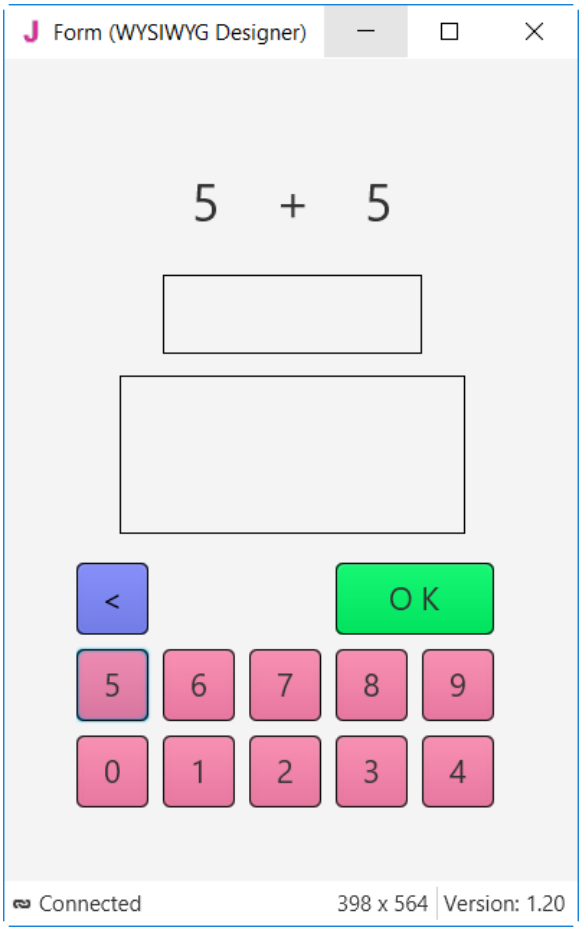
Change their Name, Tag, Text and Color properties as below.

btnBS <



btnAction O K





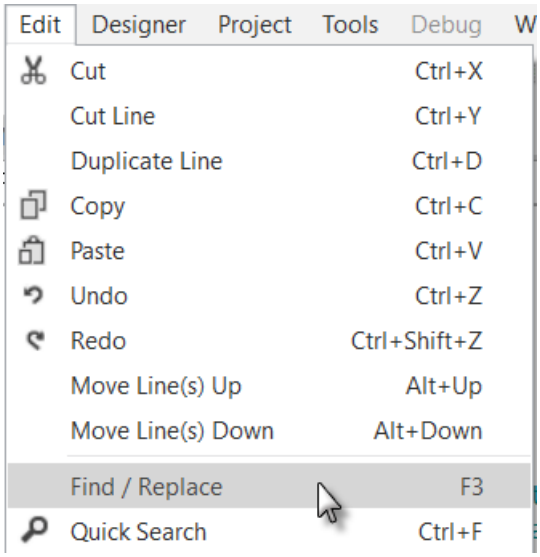
The finished new layout in the WYSIWYG form.

You could have followed all the evolutions of the layout in the WYSIWYG form.

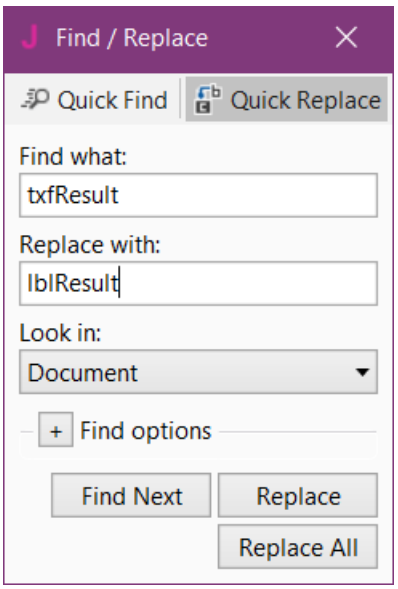
Now we will update the code.
First, we must replace the txfResult by lblResult because we replaced the TextField node by a Label.

```
10    Private lblComments As Label
11    Private lblMathSign As Label
12    Private lblNumber1 As Label
13    Private lblNumber2 As Label
14    Private txfResult As TextField
```

Double click on txfResult to select it.



Click on Find / Replace.



The Find / Replace window is displayed.

Click on Replace All and close the window.

We also need to change its node type form TextField to Label.

```
Private lblResult As Label
```

Now we write the routine that handles the Click events of the Buttons.
The Event Name for all buttons, except btnAction, is "btnEvent".
The routine name for the associated click event will be btnEvent_Click.
Enter the following code:

```
Private Sub btnEvent_MouseClicked

End Sub
```

We need to know what button raised the event. For this, we use the Sender object which is a special object that holds the object reference of the node that generated the event in the event routine.

```
Private Sub btnEvent_MouseClicked
   Private btnSender As Button
   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
   Case Else
   End Select
End Sub
```

To have access to the properties of the node that raised the event we declare a local variable `Private btnSender As Button`.
And set `btnSender = Sender`.

Then, to differentiate between the backspace button and the numeric buttons we use a Select / Case / End Select structure and use the Tag property of the buttons.
Remember, when we added the different buttons we set their Tag property to BS, 0, 1, 2 etc.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
```

Select sets the variable to test.
Checks if it is the button with the "BS" tag value.
Handles all the other buttons.

Now we add the code for the numeric buttons.
We want to add the value of the button to the text in the lblResult Label.

```
   Select btnSender.Tag
   Case "BS"
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

This is done in this line
```
   lblResult.Text = lblResult.Text & btnSender.Text
```

The "&" character means concatenation, so we just append to the already existing text the value of the Text property of the button that raised the event.

Now we add the code for the BackSpace button.
```
   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length > 0 Then
         lblResult.Text = lblResult.Text.SubString2(0, lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

When clicking on the BS button we must remove the last character from the existing text in lblResult. However, this is only valid if the length of the text is bigger than 0. This is checked with:
```
If lblResult.Text.Length > 0 Then
```

To remove the last character, we use the SubString2 function.
```
lblResult.Text = lblResult.Text.SubString2(0, lblResult.Text.Length - 1)
```

SubString2(BeginIndex, EndIndex) extracts a new string beginning at BeginIndex (inclusive) until EndIndex (exclusive).

Now the whole routine is finished.
```
Private Sub btnEvent_MouseClicked (EventData As MouseEvent)
   Private btnSender As Button

   btnSender = Sender
   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length >0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & btnSender.Text
   End Select
End Sub
```

In `Sub btnAction_MouseClicked` we add, at the end, `lblResult.Text = ""` to clear the text.
```
   Else
      New
      btnAction.Text = "O K"
      lblResult.Text = ""
   End If
End Sub
```

We can try to improve the user interface of the program by adding some colors to the lblComments Label.
Let us set:
- Yellow for a new problem
- Light Green for a GOOD answer
- Light Red for a WRONG answer.

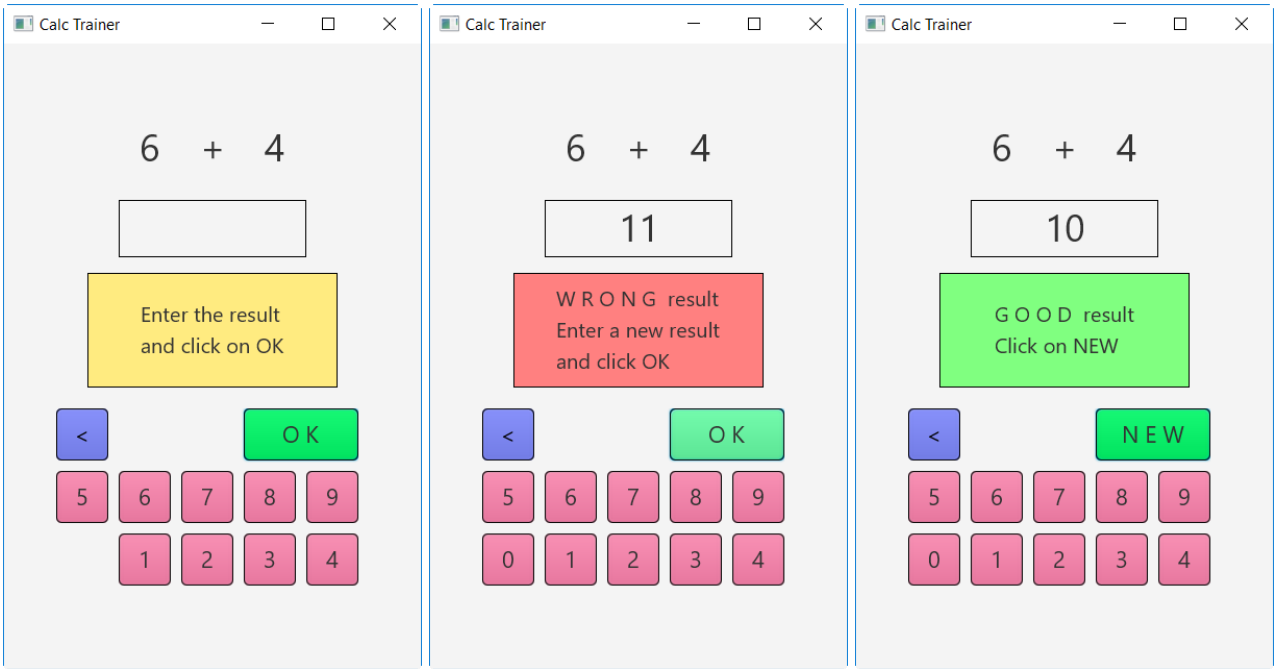
We first modify the New routine, where we add this line
`CSSUtils.SetBackgroundColor(lblComments, fx.Colors.RGB(255,235,128))`
We need to use the CSSUtils library for that! See next page how to add a library.

```
Private Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   CSSUtils.SetBackgroundColor(lblComments, fx.Colors.RGB(255,235,128))' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
End Sub
```

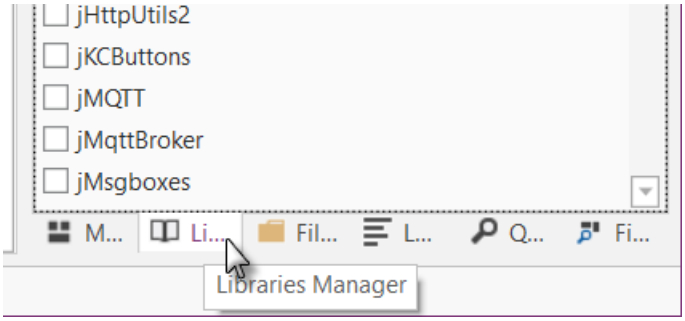
And in the CheckResult routine we add the two lines with `CSSUtils.SetBackgroundColor`...

```
Private Sub CheckResult
   If lblResult.Text = Number1 + Number2 Then
      CSSUtils.SetBackgroundColor(lblComments, fx.Colors.RGB(128,255,128))      ' light green color
      lblComments.Text = "G O O D  result" & CRLF & "Click on NEW"
      lblComments.Style = "-fx-background-color: palgreen;"     ' palegreen color
      btnAction.Text = "N E W"
   Else
      CSSUtils.SetBackgroundColor(lblComments, fx.Colors.RGB(255,128,128))      ' light red color
      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result" & CRLF & "and click OK"
   End If
End Sub
```

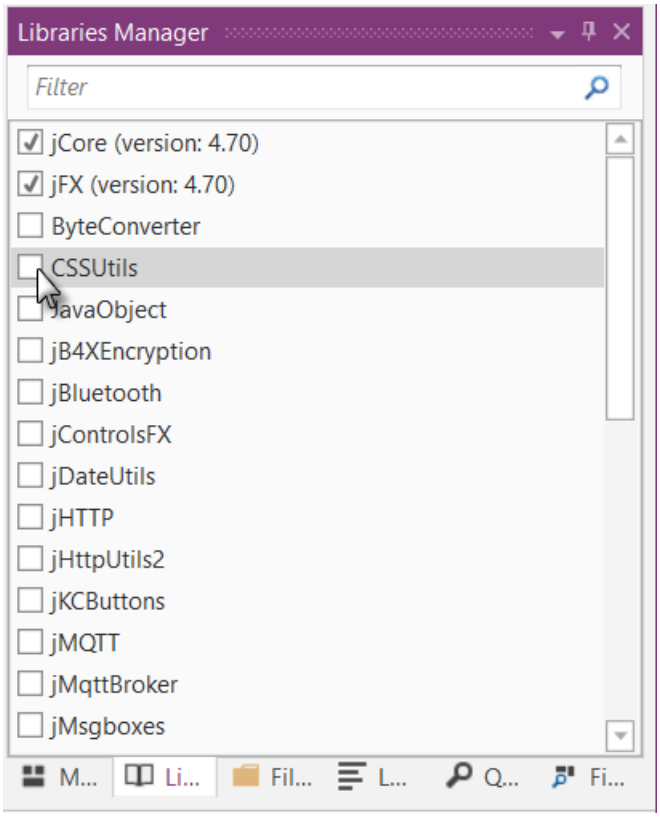
And we give the program a more meaningful title by adding `MainForm.Title = "Calc Trainer"` in AppStart just after `MainForm.Show`.



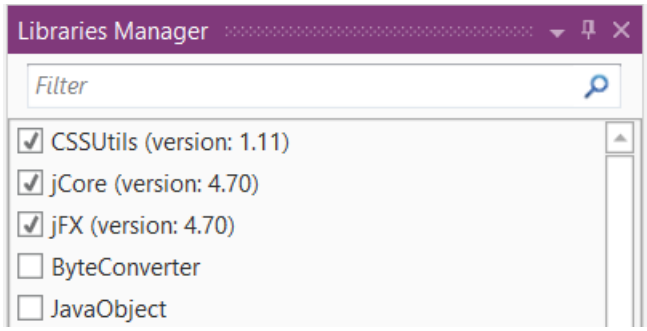
Add the CSSUtils library to the project.



In the lower right corner click on the Libraries Manager Tab.



In the libraries list check CSSUtils.



And the result.
The CSSUtils library is added to the project.

The libraries are ordered by alphabetic order.

The use of libraries is detailed in the Libraries chapter.

Another improvement would be to hide the '0' button to avoid entering a leading '0'.
For this, we hide the button in the New subroutine with line `btn0.Visible = False`.

```
Private Sub New
   Number1 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   Number2 = Rnd(1, 10)          ' Generates a random number between 1 and 9
   lblNumber1.Text = Number1     ' Displays Number1 in label lblNumber1
   lblNumber2.Text = Number2     ' Displays Number2 in label lblNumber2
   lblComments.Text = "Enter the result" & CRLF & "and click on OK"
   lblComments.Color = Colors.RGB(255,235,128)      ' yellow color
   lblResult.Text = ""           ' Sets lblResult.Text to empty
   btn0.Visible = False
End Sub
```

We see that `btn0` is in red, this means that this object is not recognized by the IDE.

```
   btn0.Visible = False
```

So we must declare it, by adding `btn0` into line 9:

```
   Private btnAction, btn0 As Button
```

Now `btn0` is no more in red.

```
btn0.Visible = False
```

In addition, in the btnEvent_MouseClicked subroutine, we hide the button if the length of the text in lblResult is equal to zero and show it if the length is greater than zero.

```
Private Sub btnEvent_MouseClicked (EventData As MouseEvent)
   Dim btnSender As Button

   btnSender = Sender

   Select btnSender.Tag
   Case "BS"
      If lblResult.Text.Length > 0 Then
         lblResult.Text = lblResult.Text.SubString2(0,lblResult.Text.Length - 1)
      End If
   Case Else
      lblResult.Text = lblResult.Text & Send.Tag
   End Select

   If lblResult.Text.Length = 0 Then
      btn0.Visible = False
   Else
      btn0.Visible = True
   End If
End Sub
```

5 Getting started B4R

B4R - The simplest way to develop native, powerful Arduino programs.

B4R follows the same concepts of the other B4X tools (B4A, B4i, B4J), providing a simple and powerful development tool.

Compiled apps run on Arduino compatible boards.

5.1 Installing Arduino IDE

B4R needs the installation of Arduino IDE version 1.6.7+.

Download Arduino IDE from this link: https://www.arduino.cc/en/Main/Software and install it.

You should not install Arduino 1.6.12 because there is a bug.

You should not install Arduino from this site: http://www.arduino.org/downloads
It wont work.

5.2 Install Microsoft .Net Framework

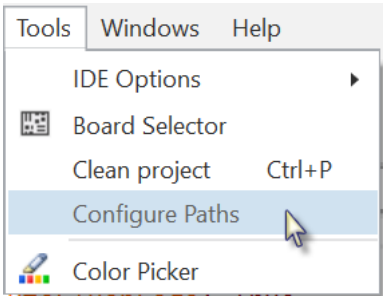
On most computers Microsoft .Net Framework is already preinstalled.

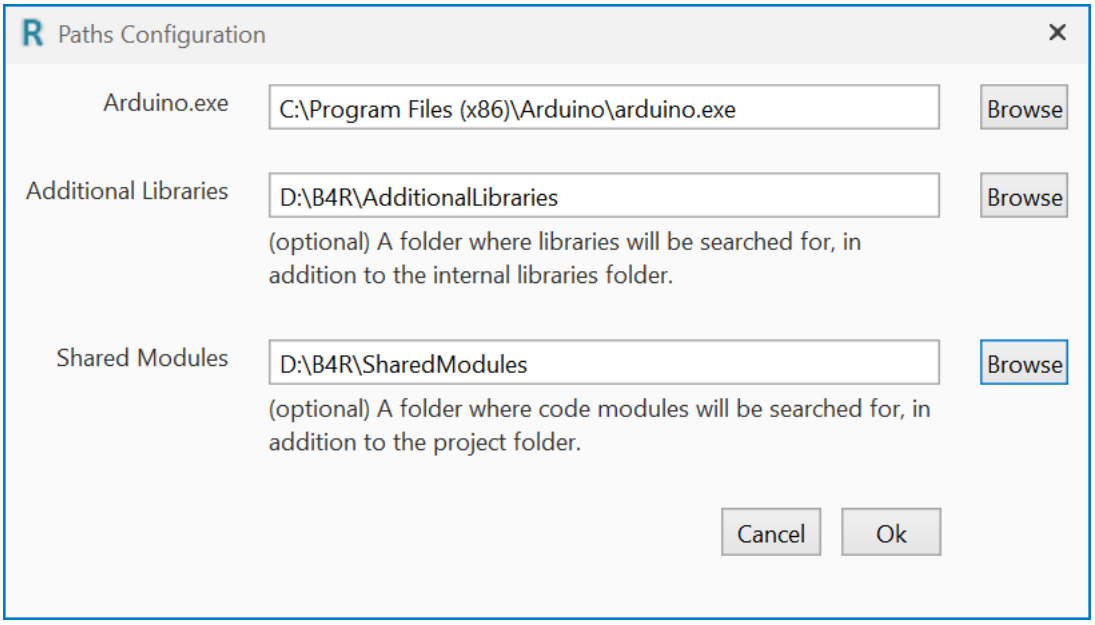
If it is not the case, you must install either:

- .Net Framework 4.5.2 for Windows Vista +
- .Net Framework 4.0 for Windows XP

5.3 Install and configure B4R

- Download and install B4R.
- Open B4R.
- Choose Tools menu - Configure Paths.





- Use the Browse button to locate "arduino.exe". The path will depend on where you installed the Arduino IDE.

It is recommended to create a specific folder for Additional Libraries.
B4R utilizes two types of libraries:

- Standard libraries, which come with B4R and are located in the Libraries folder of B4R.
  These libraries are automatically updated when you install a new version of B4R.
- Additional libraries, which are not part of B4R, should be saved in a specific folder different from the standard libraries folder.
  More details in Chapter Additional libraries folder.

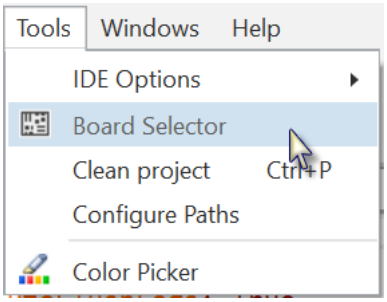
You may also define a special folder for Shared Modules.
A same Code Module can be shared between different projects without the need to load it in the project folder.

5.4 Connecting a board

When you connect a board to the PC with the USB cable Windows will load the driver and display the Serial port used.

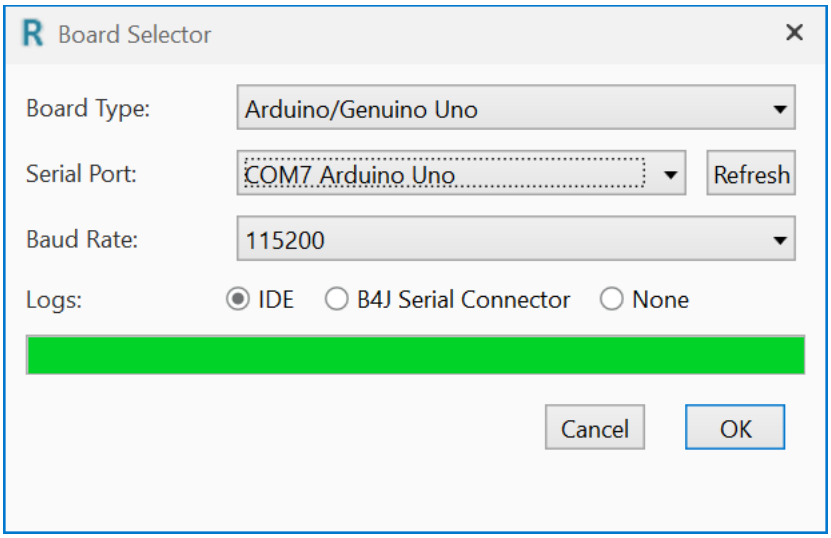
5.5 Select a Board

Run B4R.



Click on Board Selector.

The window below will be displayed.

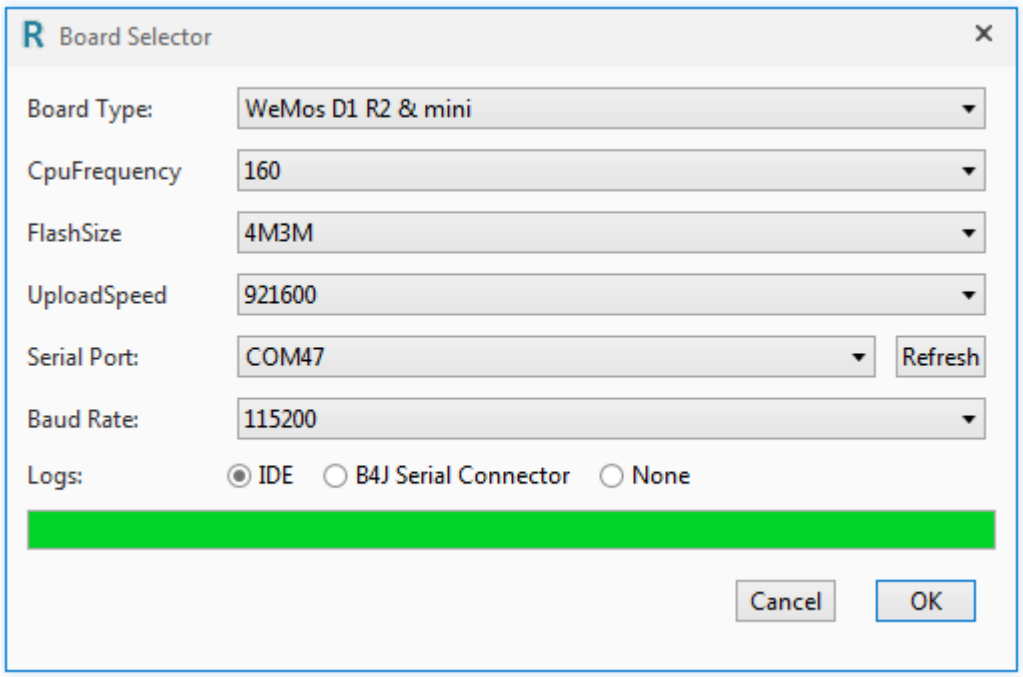


Select the Board Type in the drop down list.
Select the Serial Port and select the Baud Rate.

If only one board is connected it will automatically be recognized.

You will see only boards which are connected.

Depending on the board type several other properties can be set.



5.6 Arduino UNO board

In this chapter I will explain some basic functions of the Arduino UNO board which may be useful for beginners.
The Arduino UNO board is the basic board of the Arduino family.

There exist other more advanced boards.

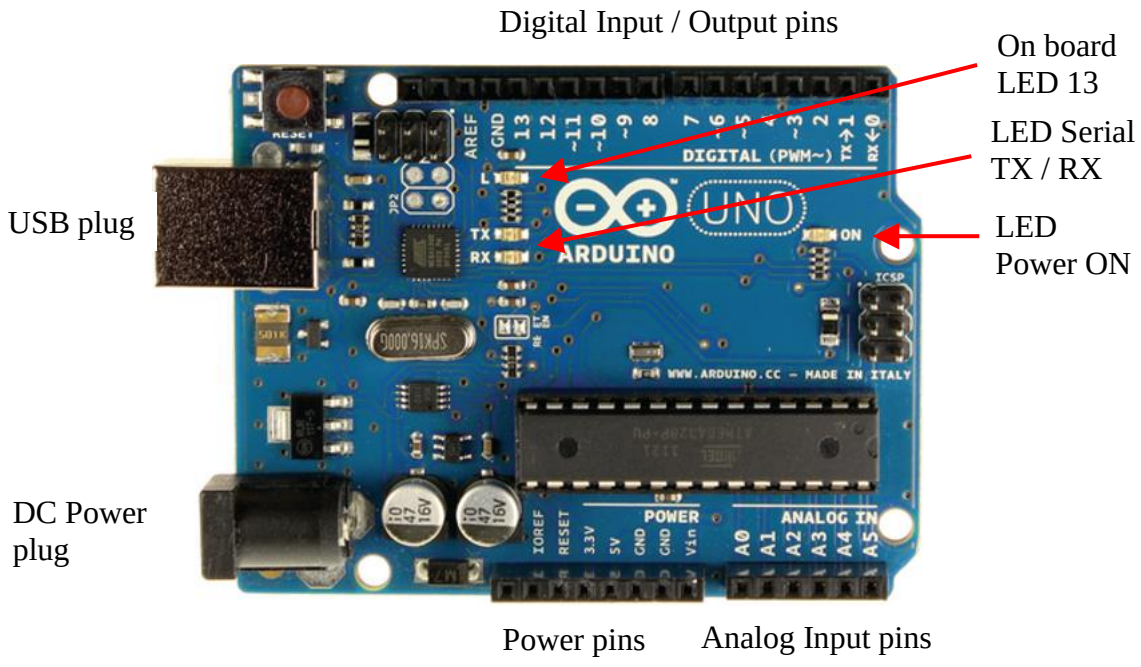
- Arduino DUE
- Arduino MEGA
- Arduino MICRO
- etc. see Compare board specs.

Additional boards called 'Shields' can be clipped onto the Arduino boards.

- Arduino Wi-Fi Shield 101
- Arduino Ethernet Shield
- etc.

The Arduino UNO:

The source of the information in this chapter is a summary from the Arduino site.



5.6.1 Power supply

The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V).
Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board (see the pins below). We don't advise it.

5.6.2 Pins

The Arduino UNO has 3 pin sockets:

- Power pins.
- Digital Input / Output pins.
- Analog Input pins.

5.6.3 Power pins

The Power pins are:

- **GND** Power ground, 2 pins.
- **VIN** Power supply input.
  The input voltage to the Uno board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
  Voltage 7 – 12 V.
- **5V** 5 Volt reference voltage. **Don’t provide the power to this pin!**
  This pin outputs a regulated 5V from the regulator on the board.
- **3.3V** 3.3 Volt reference voltage. **Don’t provide the power to this pin!**
  A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **RESET**
  Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.
- **IOREF**
  This pin on the Uno board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.

5.6.3.1 Digital Input / Output pins

Each of the 14 digital pins on the Uno can be used as an input or output, using the *pinMode* method, *DigitalRead* and *DigitalWrite* functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.

In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- PWM: ~3, ~5, ~6, ~9, ~10, and ~11. These numbers have this "~" prefix.
  They can provide 8-bit PWM (**P**ulse **W**idth **M**odulation) outputs with the *AnalogWrite* function allowing to modulate the brightness of a LED (**L**ight **E**mitting **D**iode) or run DC motors at different speeds.
  A value of 0 means always OFF and 255 means always ON.
  Usage:
  pinTest3.AnalogWrite(Value As UInt)
  ```
  pinTest3.AnalogWrite(196)
  ```
  After *AnalogWrite* the pin will generate a steady square wave of the specified duty cycle until the next call to *AnalogWrite* (or a call to *DigitalRead* or *DigitalWrite* on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.
- LED 13: There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

5.6.3.2 Analog input pins

The Arduino UNO has 6 Analog input pins A0 to A5 with 10 bit analog to digital converters (0 to 1023 resolution).
The reference voltage is 5 Volt allowing a resolution of 4.9 mV per unit.
While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/output (GPIO) pins (the same as digital pins 0 - 13).

5.6.4 Input modes INPUT / INPUT_PULLUP

If you have your pin configured as an INPUT, and are reading a switch, when the switch is in the open state the input pin will be "floating", resulting in unpredictable results. In order to assure a proper reading when the switch is open, a pull-up or pull-down resistor must be used. The purpose of this resistor is to pull the pin to a known state when the switch is open. A 10 K ohm resistor is usually chosen, as it is a low enough value to reliably prevent a floating input, and at the same time a high enough value to not draw too much current when the switch is closed.

The INPUT_PULLUP mode adds an internal pull up resistor no need to add one externally.

With a pull up resistor the pin returns False when the switch is closed because it sets the input to 0 Volt.

5.6.5 Basic Pin functions

5.6.5.1 Initialize

Initializes a pin.

Pin.Initialize(Pin As Byte, Mode As Byte)

Pin is the pin number.
- 0, 1, 2, 3 etc. for digital pins
- Pin.A0, Pin.A1 , Pin.A2 etc. for analog pins.

Mode is one of the three connection modes:
- MODE_INPUT
- MODE_INPUT_PULLUP adds an internal pull up resistor.
- MODE_OUTPUT

Example1: Initialize digital pin 3 as input.
```
Private pinTest1 As Pin
pinTest1.Initialize(3, pinTest1.MODE_INPUT)
```

Example2: Initialize digital pin 3 as input with pull up resistor.
```
Private pinTest2 As Pin
pinTest2.Initialize(3, pinTest2.MODE_INPUT_PULLUP)
```

Example3: Initialize digital pin 3 as output.
```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

Example4: Initialize analog pin 3 as input.
```
Private pinTest4 As Pin
pinTest4.Initialize(pinTest4.A4, pinTest4.MODE_INPUT)
```

The analog pins, on the Arduino UNO, can also be accessed with numbers, like:
```
pinTest4.Initialize(18, pinTest4.MODE_INPUT)
```

Pin.A0 = 14
Pin.A1 = 15
Pin.A2 = 16
Pin.A3 = 17
Pin.A4 = 18
Pin.A5 = 19

Initializing an analog pin as output works like a digital output pin.

5.6.5.2 DigitalRead

Reads the current digital value of a pin.
The return value is True or False.

Pin.DigitalRead returns a Boolean.

There are two input modes depending on the input signal.

- Pin.MODE_INPUT
- Pin. MODE_INPUT_PULLUP adds an internal pullup resistor for use with a switch.

Example:

```
Private pinTest1 As Pin
pinTest1.Initialize(3, pinTest.MODE_INPUT)

Private Value As Boolean
Value = pinTest1.DigitalRead
```

The Arduino uses internally 0 and 1 for a boolean variable.
`Log("State: ", Value)`
Will write either 0 for False or 1 for True in the Logs. In the code you can use False and True.

5.6.5.3 DigitalWrite

Writes a Boolean value to the given pin.
It can be used for all digital pins and also all analog pins.

Pin.DigitalWrite (Value As Boolean)

Example:

```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)

pinTest3.DigitalWrite(True)
```
directly with the value.
`pinTest3.DigitalWrite(Value)` with a variable.

5.6.5.4 AnalogRead

AnalogRead reads the current value of an analog pin.
The return value is an UInt with values between 0 and 1023 (10 bits).
The reference voltage is 5V.

Example:

```
Private pinPot As Pin
pinPot.Initialize(pinPot.A4, pinPot.MODE_INPUT)

Private Value As UInt
Value = pinPot.AnalogRead
```

5.6.5.5 AnalogWrite

AnalogWrite writes a Byte value to the given pin.
AnalogWrite has nothing to do with the analog pins nor with AnalogRead.

AnalogWrite can only be used on the digital pins ~3, ~5, ~6, ~9, ~10, and ~11 on the Arduino UNO, the pins with the ~ prefix.

Pin.AnalogWrite (Value As UInt)

Example: we use digital pin ~3 which allows PWM.

```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

`pinTest3.AnalogWrite(145)` directly with the value.
`pinTest3.AnalogWrite(Value)` with a variable, `Value` must be a UInt variable.

5.7 First example programs

All the projects were realized with the Arduino Starter Kit.

The diagrams for the projects were realized with the Fritzing software.

When we run the IDE we get the default code like below.
The `#Region Project Attributes` is normally collapsed; these attributes are explained in the Code header Project Attributes chapter.

```
#Region Project Attributes
   #AutoFlushLogs: True
   #CheckArrayBounds: True
   #StackBufferSize: 300
#End Region

Sub Process_Globals
   'These global variables will be declared once when the application starts.
   'Public variables can be accessed from all modules.
   Public Serial1 As Serial
End Sub

Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")
End Sub
```

`Public Serial1 As Serial`	Defines the serial interface with the computer.
`Serial1.Initialize(115200)`	Initializes the serial port with a Baud rate of 115200 Hertz.
`Log("AppStart")`	Shows `AppStart` in the Logs when the program starts.

All the projects were realized with the Arduino Starter Kit.

The sketches for the projects were realized with the Fritzing software.

5.7.1 Button.b4r

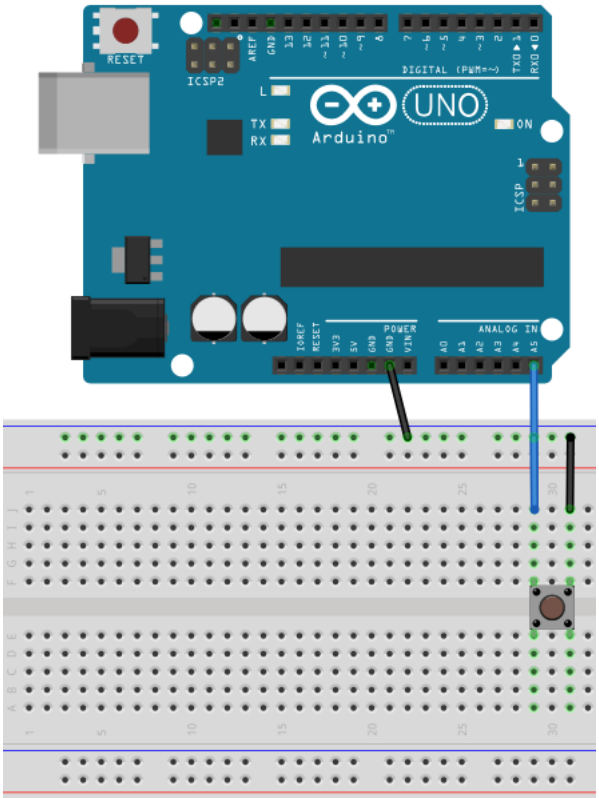
Let us write our first program.

It's similar to Erels Button example from the forum. It uses a pushbutton switch and the Led 13 on the Arduino UNO board.

The project Button.b4r is available in the SourceCode folder.

- Open B4R.
- Save the project as Button in a folder with the name Button.
- Build the board with the pushbutton and the wires.
- Connect the Arduino to the PC.
- Write the code.
- Run the program.

5.7.1.1 Sketch



Material:

- 1 pushbutton switch

Connect one Arduino **GND** (ground) pin to the ground **GND** line of the breadboard.
Then connect one pin of the pushbutton switch to the ground line.
And connect the other pin of the pushbutton to pin **A5** of the Arduino analog pins.

We could have connected the first pin of the pushbutton directly to the **GDN** pin of the Arduino, but the connection in the image is ready for the next examples.

We could also have used one of the digital pins instead of the analog pin.

5.7.1.2 Code

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinButton As Pin      'pin for the button
   Private pinLED13 As Pin       'pin for LED 13 on the Arduino
End Sub
```

We declare the pins for the pushbutton and the on board Led 13.

```
Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")

   pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
   'Using the internal pull up resistor to prevent the pin from floating.
   pinButton.AddListener("pinButton_StateChanged")

   pinLED13.Initialize(13, pinLED13.MODE_OUTPUT)
End Sub
```

We initialize `pinButton`, as the analog pin **A5**, with `pinButton.A5` and set the input mode to `pinButton.MODE_INPUT_PULLUP`. We need a pullup resistor to prevent the pin from floating, `MODE_INPUT_PULLUP` connects an internal pull up resistor.

We add `pinButton.AddListener("pinButton_StateChanged")`, to generate a StateChanged event when the state of pin `pinButton` changes which means that the pushbutton is pressed or released.

We initialize `pinLED13`, as the onboard Led as digital pin 13 and set the output mode to `pinLED13.MODE_OUTPUT`.

```
Sub pinButton_StateChanged (State As Boolean)
   Log("State: ", State)
   'state will be False when the button is clicked because of the PULLUP mode.
   pinLED13.DigitalWrite(Not(State))
End Sub
```

We add a Log, `Log("State: ", State)`, to display the state in the Logs.

We write the State to the digital output of the on board led, `pinLED13.DigitalWrite(Not(State))`.

We write `Not(State)` because State will be False when the pushbutton is pressed because of the PULLUP mode.

Click on ▶ or press F5 to run the code.

When you press the pushbutton, led 13 on the Arduino UNO will be ON and when you release the pushbutton led 13 will be OFF.

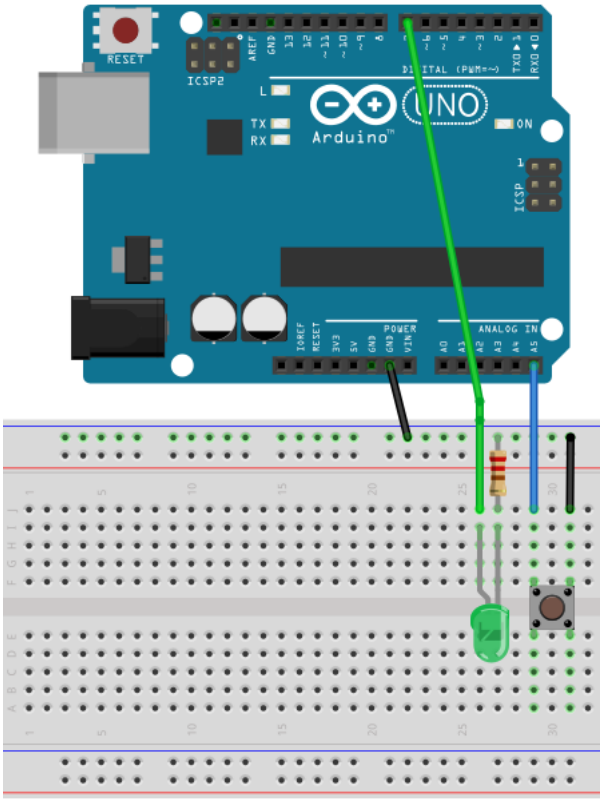
5.7.2 LedGreen.b4r

For this project we use a copy of the Button project.
Create a new LedGreen folder, copy the files of the Button project and rename the Button.xxx files to LedGreen.xxx.

We add a green Led which can be switched on and off with the button from the first example.

The project LedGreen.b4r is available in the SourceCode folder.

5.7.2.1 Sketch



Material:

- 1 pushbutton switch
- 1 green LED
- 1 220 Ω resistor

We keep the mounting of the pushbutton switch from the first example.
One pin on the **GND** line of the breadboard.
The other pin to digital pin **7** on the Arduino.

And we
- Add a green Led on the breadboard.
- Connect the cathode (-) via a 220 Ω resistor to the ground **GND** line of the breadboard.
- Connect the anode (+) to digital pin **7**.

5.7.2.2 Code

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinButton As Pin            'pin for the button
   Private pinLEDGreen As Pin          'pin for the green Led
   Private LightOn = False As Boolean
End Sub
```

We keep the definition of `pinButton`.

We change the definition
`Private pinLED13 As Pin`
to
`Private pinLEDGreen As Pin`
for the green Led

We add a global boolean variable `LightOn` which is True when the light is ON.

```
Private Sub AppStart
   Serial1.Initialize(115200)

   pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
   'Using the internal pull up resistor to prevent the pin from floating.
   pinButton.AddListener("pinButton_StateChanged")

   pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
End Sub
```

We leave the code for `pinButton`.
We initialize `pinLEDGreen` as digital pin **7** and set the output mode to `pinLEDGreen.MODE_OUTPUT`.

```
Private Sub pinButton_StateChanged (State As Boolean)
   If State = False Then  'remember, False means button pressed.
      LightOn = Not(LightOn)
      pinLEDGreen.DigitalWrite(LightOn)
   End If
End Sub
```

Every time State is False, pushbutton pressed, we change the variable `LightOn` and write it to `pinLEDGreen`.

5.7.3 LedGreenNoSwitchBounce.b4r

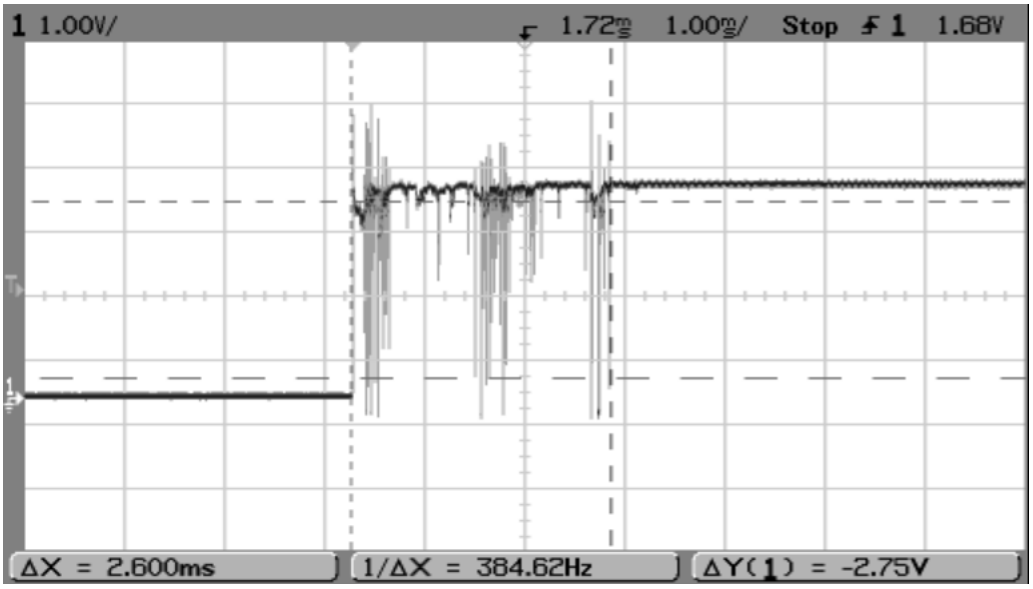
For this proejct we use exactly the same circuit as LedGreen.b4r.

The only difference is in the code.

The project LedGreenNoSwitchBounce.b4r is available in the SourceCode folder.

The pushbutton switch we use in our projects has a problem called bouncing.
The signal of a mecanical switch is not clean, the switch has several bounces which are interpreted by the the digital inputs as several state changes. If we have an even number of state changes it is similar to having done nothing.
But we want only one state change per button press.

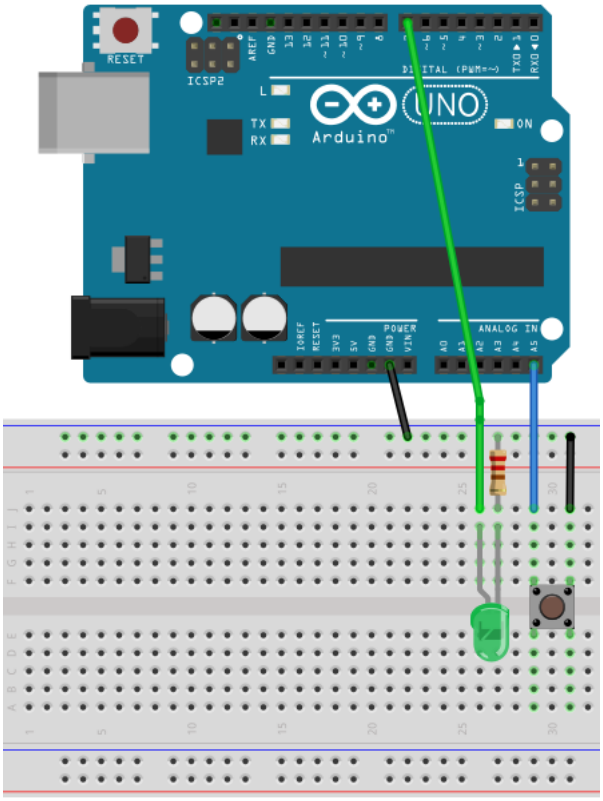
Image of switch bouncing (source Wikipedia):



The switch bounces between on and off several times before settling.

To solve this problem we don't react in the `pinButton_StateChanged` routine on state changes within a given time.

5.7.3.1 Sketch



Material:

- 1 pushbutton switch
- 1 green LED
- 1 220 Ω resistor

We keep the mounting of the pushbutton switch from the first example.
One pin on the **GND** line of the breadboard.
The other pin to digital pin **7** on the Arduino.

And we
- Add a green Led on the breadboard.
- Connect the cathode (-) via a 220 Ω resistor to the ground **GND** line of the breadboard.
- Connect the anode (+) to digital pin **7**.

5.7.3.2 Code

The code is almost the same as LedGreen.b4r.

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinButton As Pin            'pin for the button
   Private pinLEDGreen As Pin          'pin for the green Led
   Private LightOn = False As Boolean
   Private BounceTime As ULong
   Private BounceDelay = 10 As ULong
End Sub
```

We add two new variables: `BounceTime` and `BounceDelay`.

```
Private Sub AppStart
   Serial1.Initialize(115200)

   'Using the internal pull up resistor to prevent the pin from floating.
   pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
   pinButton.AddListener("pinButton_StateChanged")

   pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
End Sub
```

Same as LedGreen.b4r

New `pinButton_StateChanged` routine:

```
Private Sub pinButton_StateChanged (State As Boolean)
   If State = False Then  'remember, False means button pressed.
      If Millis - BounceTime < BounceDelay Then
         Return            'Return, bouncing
      Else
         LightOn = Not(LightOn)
         pinLEDGreen.DigitalWrite(LightOn)
         BounceTime = Millis   'reset BounceTime to current time
      End If
   End If
End Sub
```

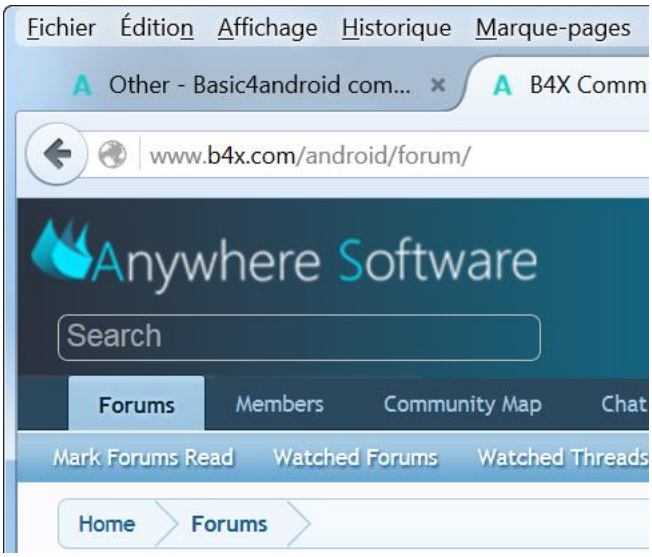
Every time State is False, pushbutton pressed:

- We check if the time between the current state change and the first state change.
  If the time is shorter than `BounceDelay`, which means a bounce, we do nothing.
  If the time is longer than `BounceDelay`, which means a real state change, we execute the code.
- We change the variable `LightOn` and write it to `pinLEDGreen`.
- Set the new `BounceTime`.

6 Help tools

To find answers to many questions about B4A the following tools are very useful.

6.1 Search function in the forum

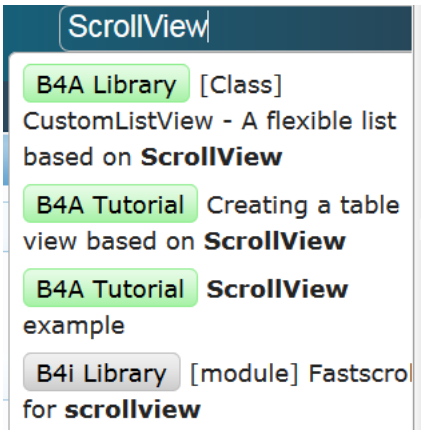


In the upper left corner you find the searchbox for the forum. Depending on the window size it can be on the top right side.

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

The function shows you the posts that match your request.

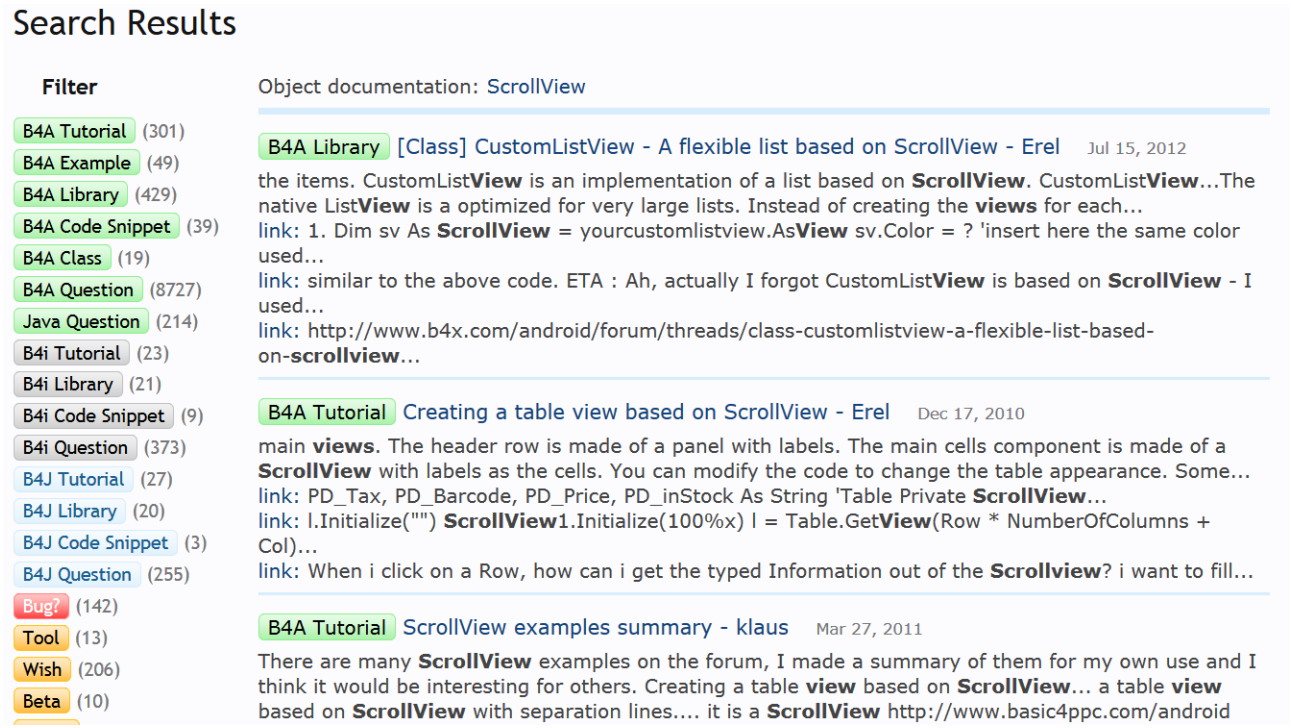
Example: Enter the keyword ScrollView : 



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

Click on an item to show the thread.

And the result:



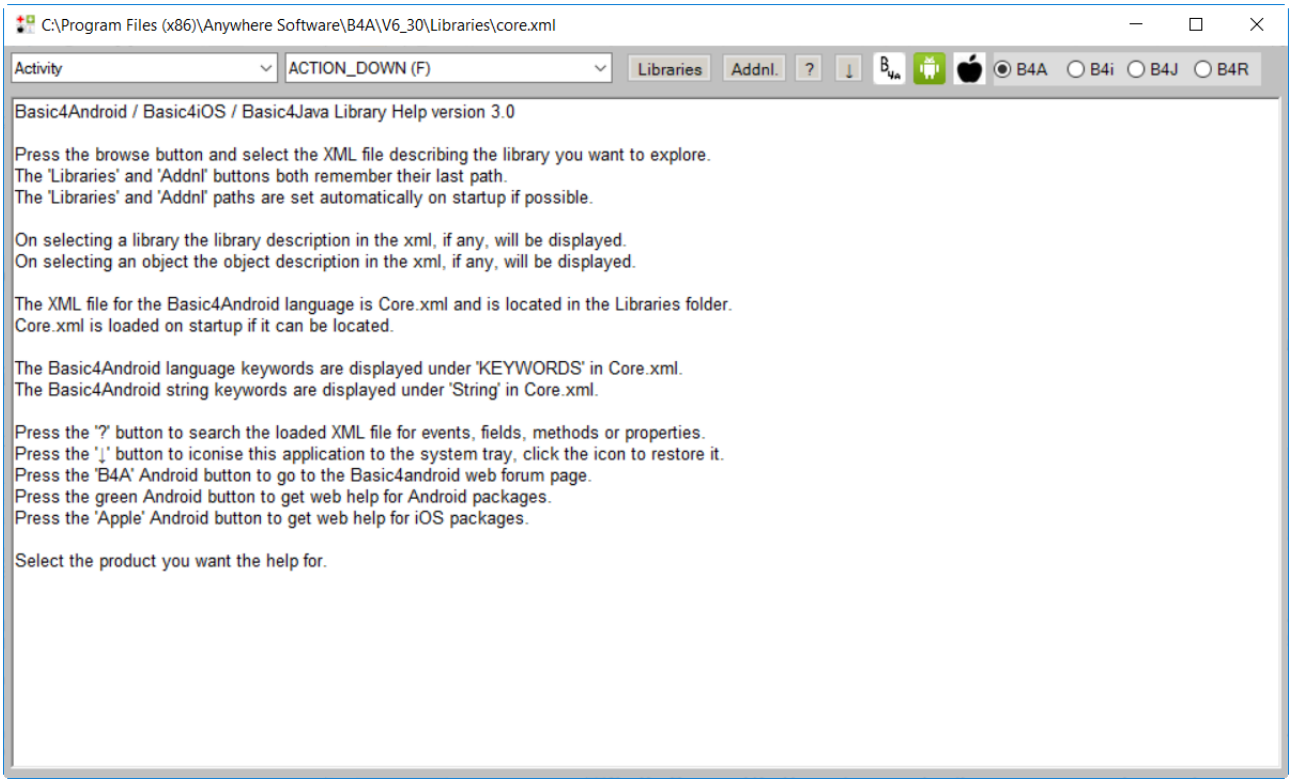
On the left you have a list of forums which you can filter.

Click on the title to show the selected post.

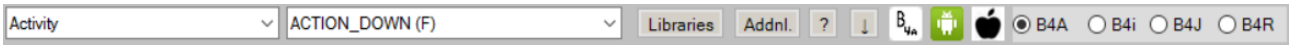
6.2 B4x Help Viewer

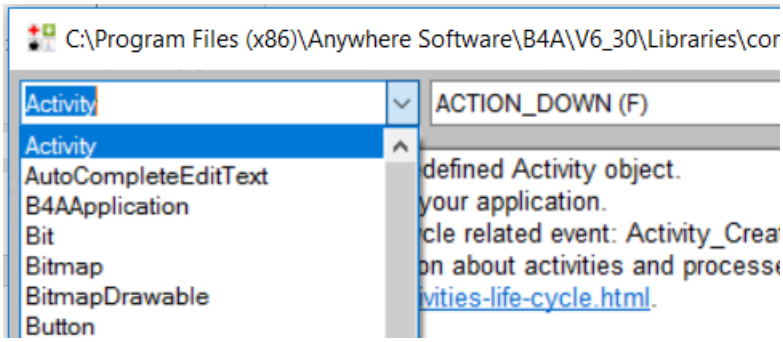
This program shows xml help files. It was originally written by Andrew Graham (agrham) for B4A. I modified it, with Andrews' agreement, to show B4A, B4J, B4i and B4R xml help files.

The program can be downloaded from the forum.

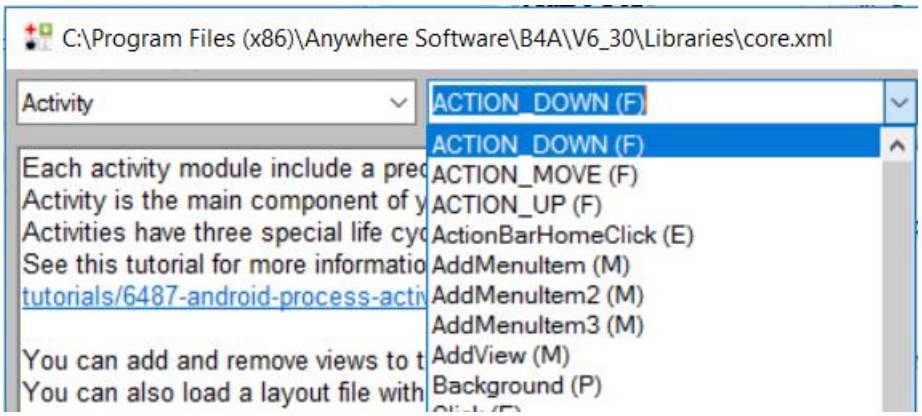


On top we find:





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

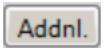


Besides the objects list you find another drop down list with the

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

for the selected object.

Select the standard library to display.



Select the additional library to display.

Search engine to find the object for a given keyword.

Closes B4AHelp

Launches the forum 'Online Community'.

Launches the Android Developers site.

Launches the iOS developer's site.

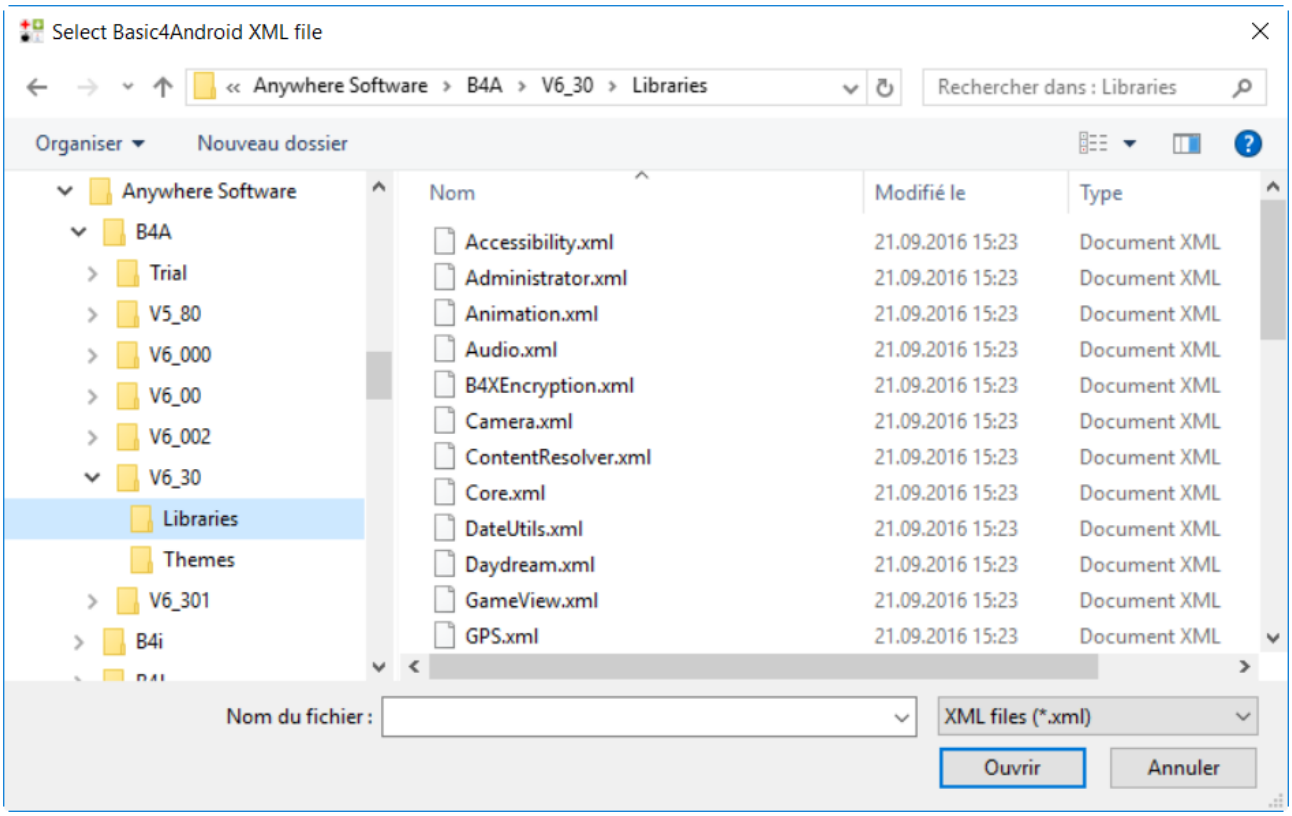
B4A help files.

B4i help files.

B4J help files.

B4R help files.

Libraries Standard libraries



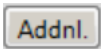
Select the library to display and click on 

(Open).

Here 

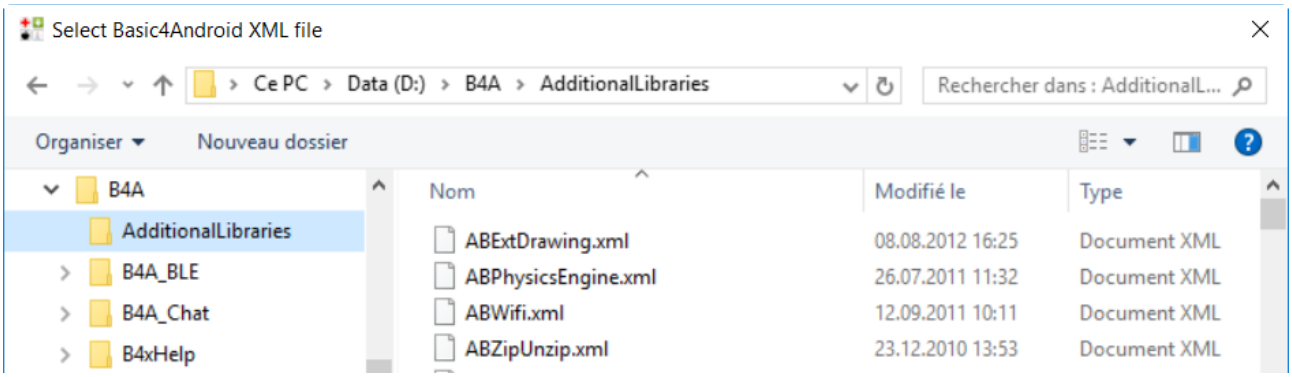
you can select the directory where the standard libraries are saved.

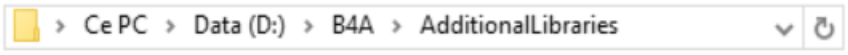
Once selected the directory is saved for the next start of the program.



Additional libraries.

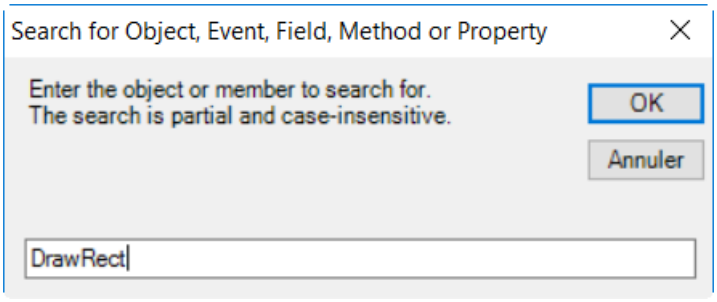
The same also for the additional libraries.



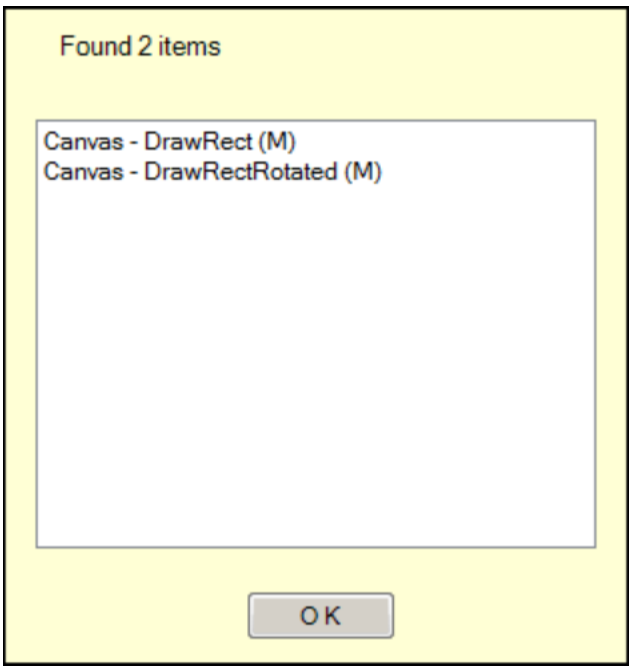
Here 

you can select the directory for the additional libraries.

Search engine for the selected library.

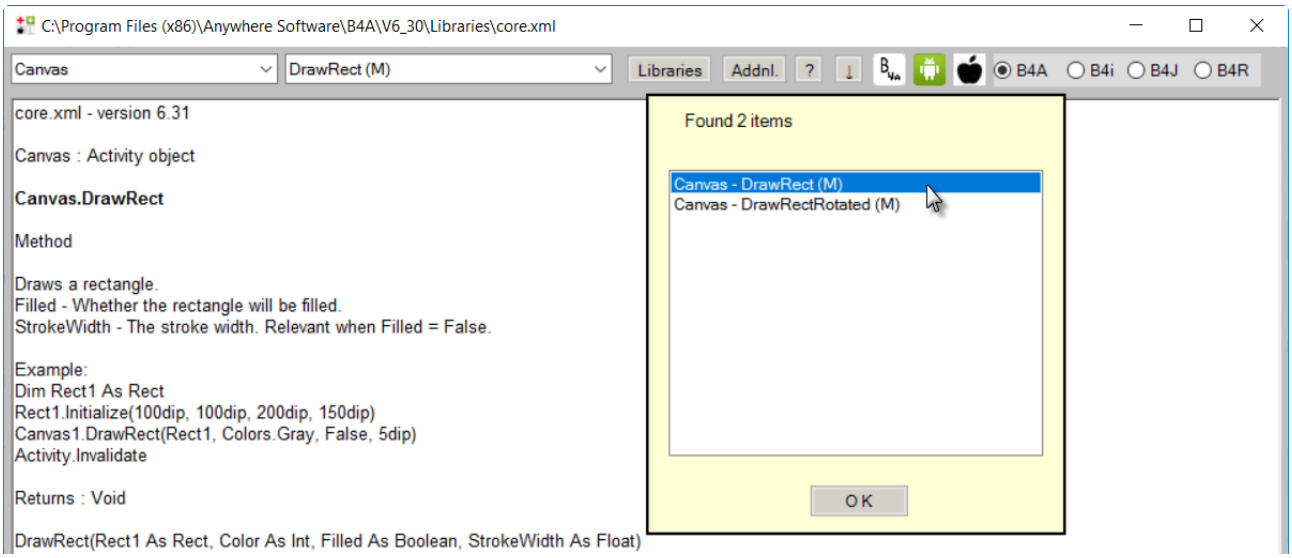


Example:
Selected library: Core
Enter *DrawRect*

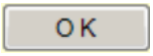


And the result.

We get the object Canvas and two methods.



Click on an item in the list to show its help.

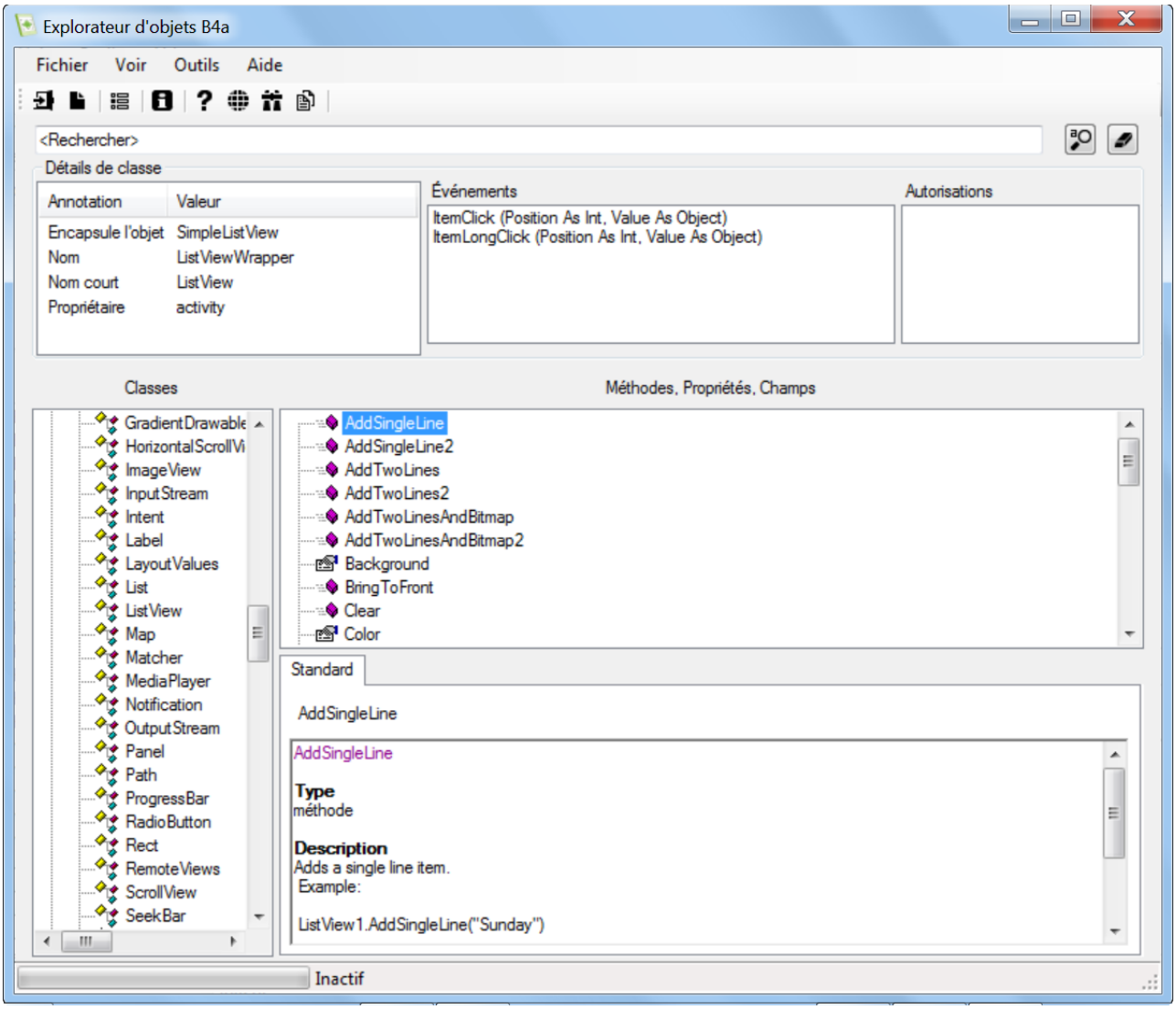
Click on  to leave the search result list.

6.3 Help documentation - B4A Object Browser

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

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

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



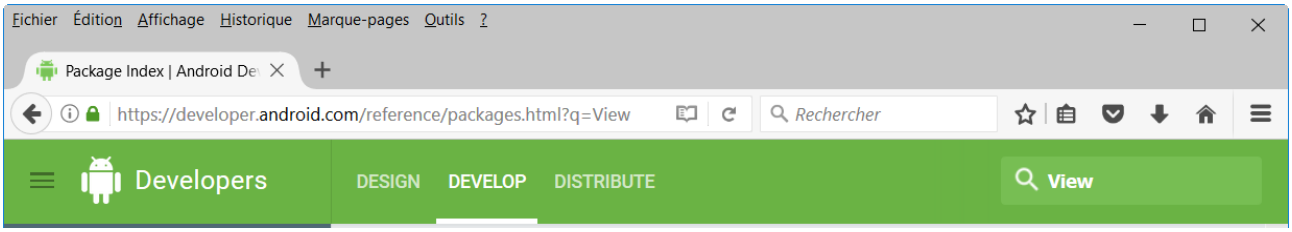
6.4 Useful links

6.4.1 B4A

A useful link for layout graphics. [Android cheat sheet for graphic designers]

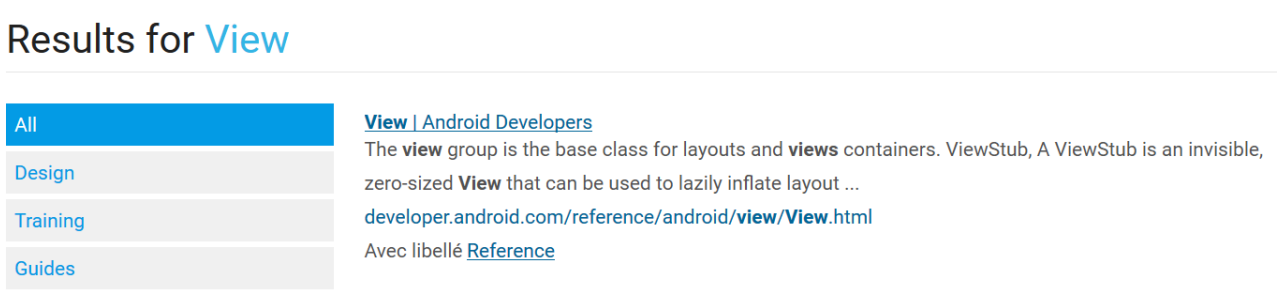
Android Developers. [Design] [Develop] [Distribute]

Android Developers searching for any request.



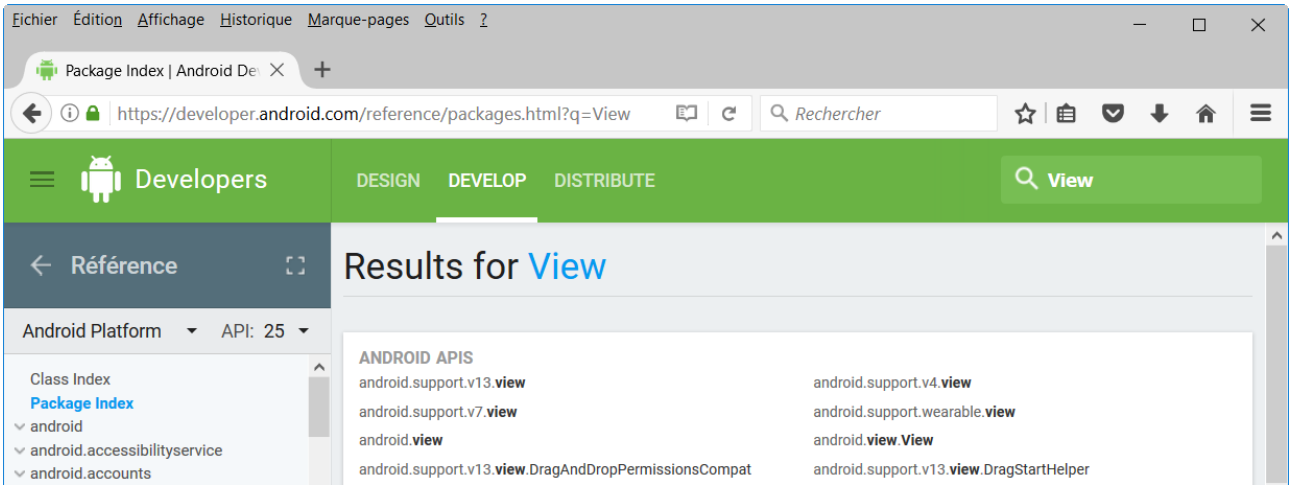
In the upper right corner you find the search field.

Enter *View* in the field: 



Click on the link View | Android Developers.

And you get all the information about Views.



6.4.2 B4i

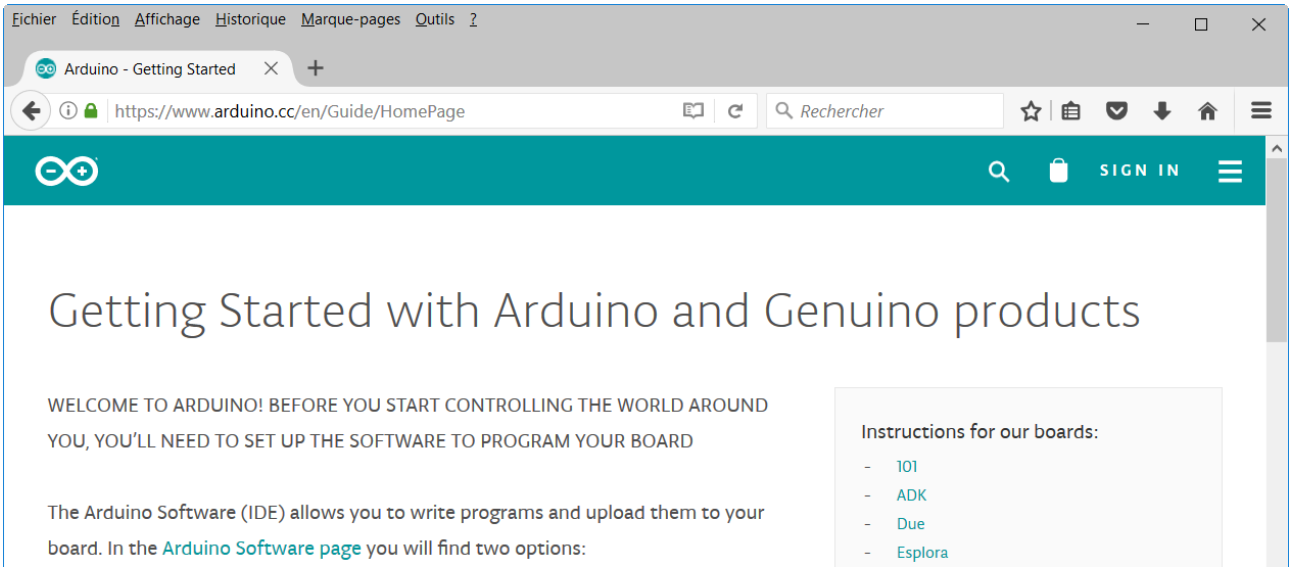
iOS developers:

6.4.3 B4J

6.4.4 B4R

Arduino HomePages:

https://www.arduino.cc/en/Guide/HomePage



6.5 Books

B4A book

Written by Philip Brown under the pseudo Wyken Seagrave.

http://pennypress.co.uk/b4a-book/

MagBook Build your own Android App.

Written by Nigel Whitfield.

http://www.magbooks.com/product/build-your-own-android-app/