Beginner's Guide

Edition 1.0

Last update: 2017.01.19

Contributors:

Klaus Christl
Erel Uziel

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

Updated for B4J version 4.70.

1 Getting Started

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.

1.1 Installing B4J

1.1.1 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"** in the platforms list (for 64 bit machines as well).
You should install the regular JDK for 64-bit computers as well.

- Download the file and install it.

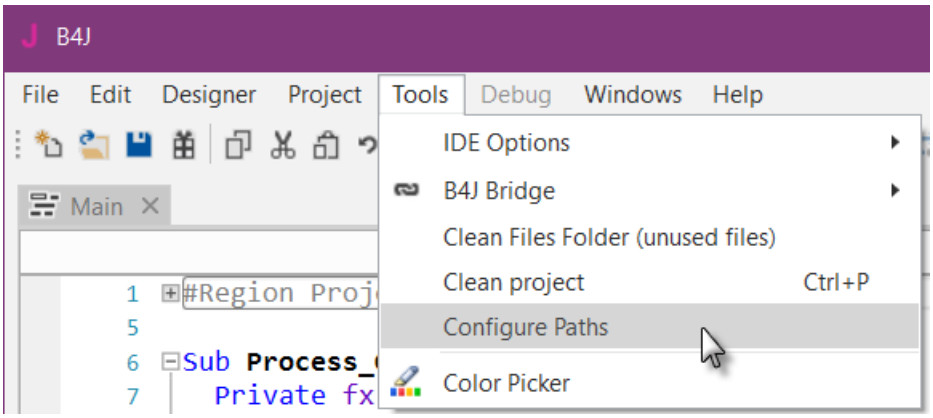
1.1.2 Installing B4J

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.

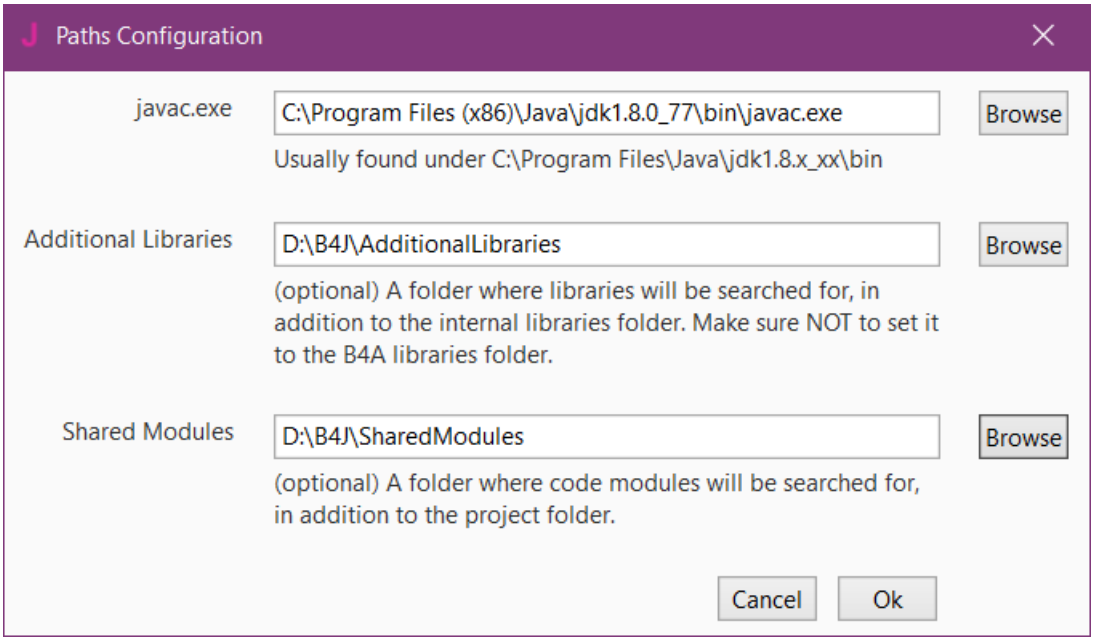
1.2 Configure Paths in the IDE

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.

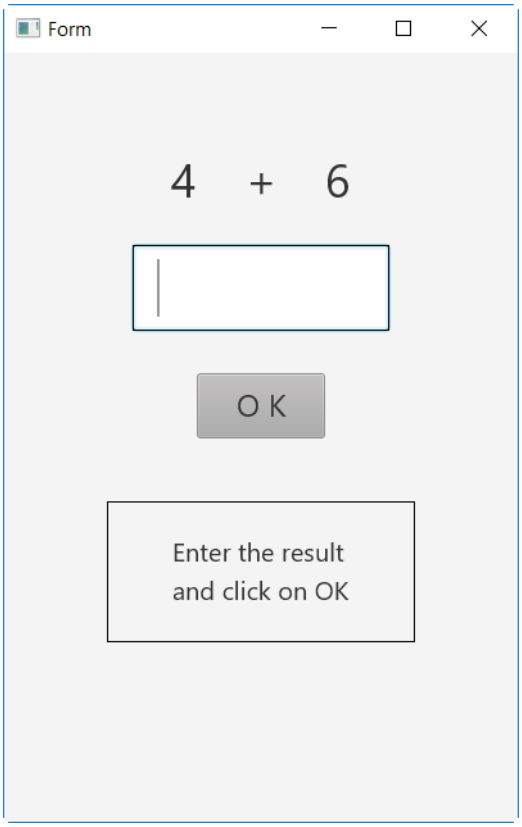
Additional libraries:
Create s specific folder for additional libraries, for example C:\B4J\AdditionalLibraries.

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

2 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 the Beginner's Guide:
SourceCode\MyFirstProgram\MyFirstProgram.b4j



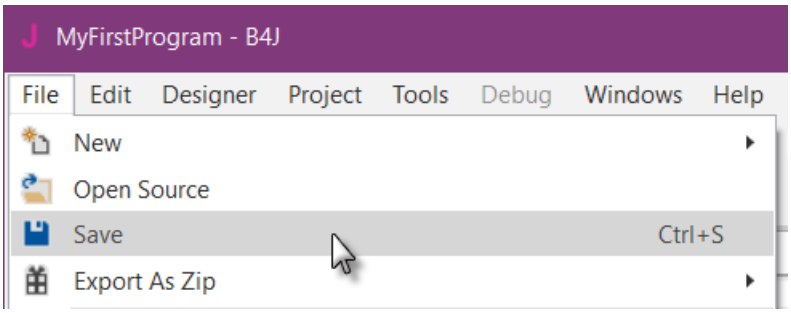
On the screen, we will have:
- 2 Labels displaying randomly generated numbers (between 1 and 9)
- 1 Label with the math sign (+)
- 1 TextField where the user must enter the result
- 1 Button, used to either confirm when the user has finished entering the result or generate a new calculation.
- 1 Label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

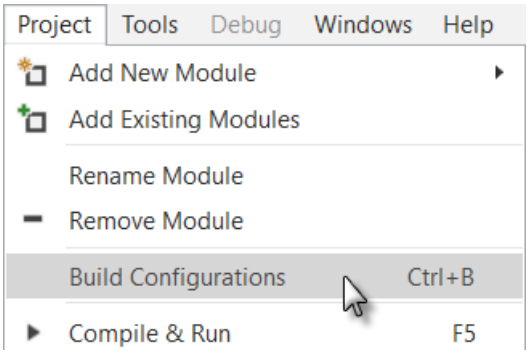


You must save the project before you can run the Designer.

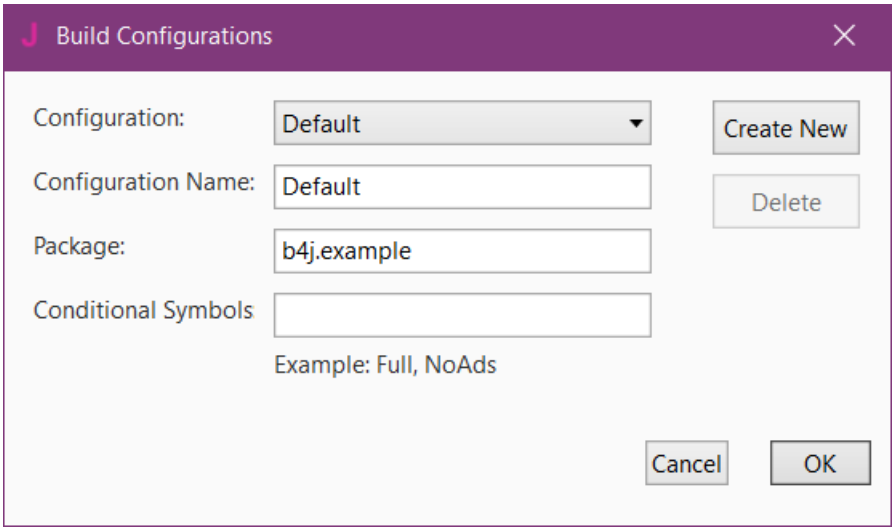
Create a new folder MyFirstProgram and save the project with the name MyFirstProgram.

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

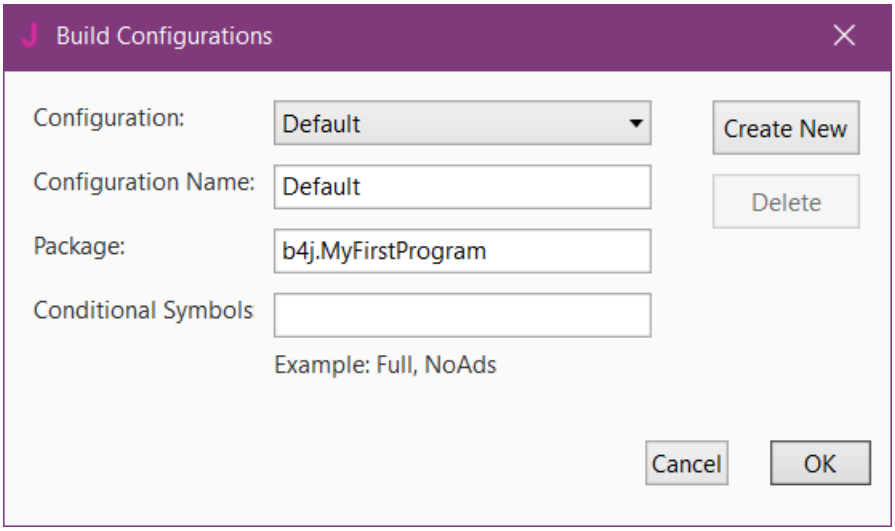
In the menu Project click on Build Configurations.



This window appears:



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

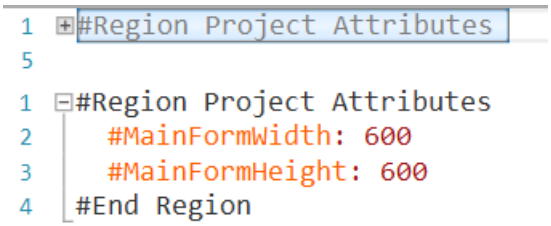


Set the Form size.

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.

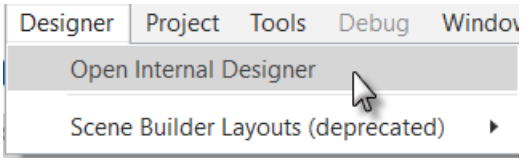


Here we can define the size of the project main window called Form.
The default values are Width = 600 and Height = 400.

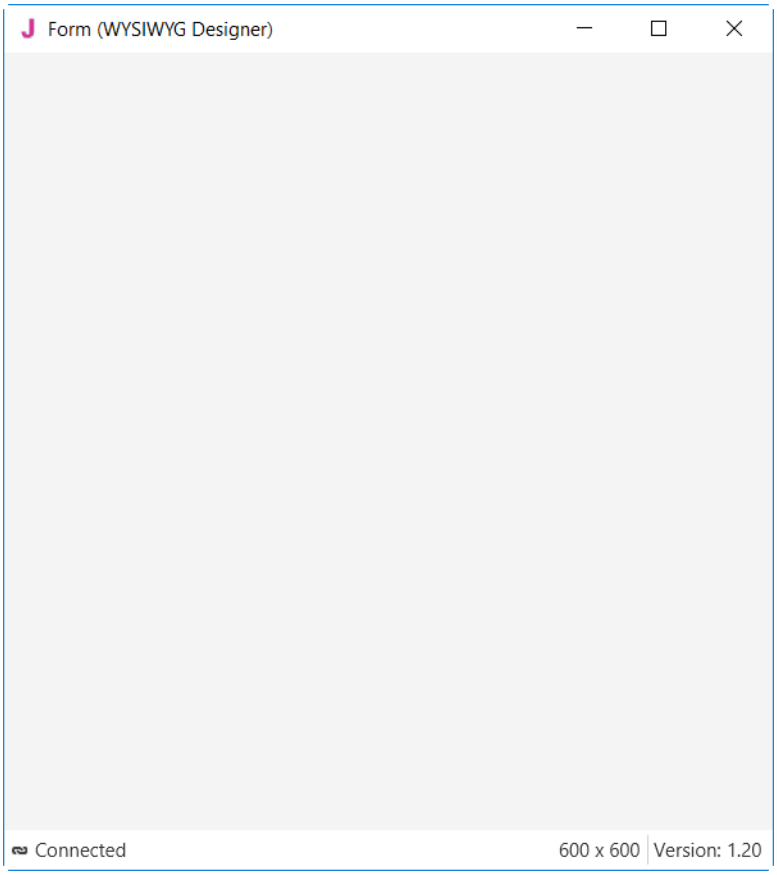
We change these to Width = 400 and Height = 600.

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

In the IDE open the Designer.

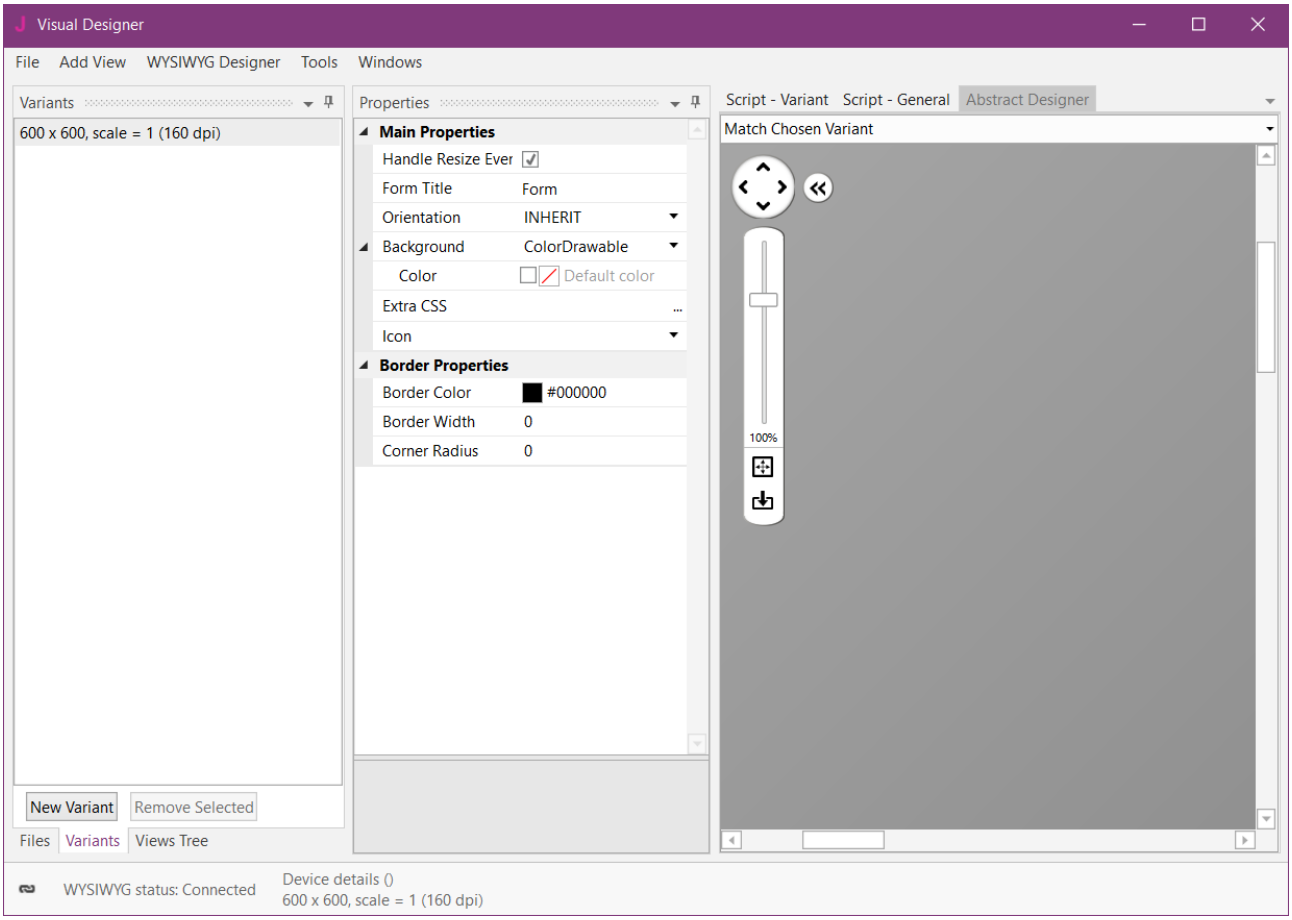


Wait until the Designer is ready.



We get a WYSIWYG Form.

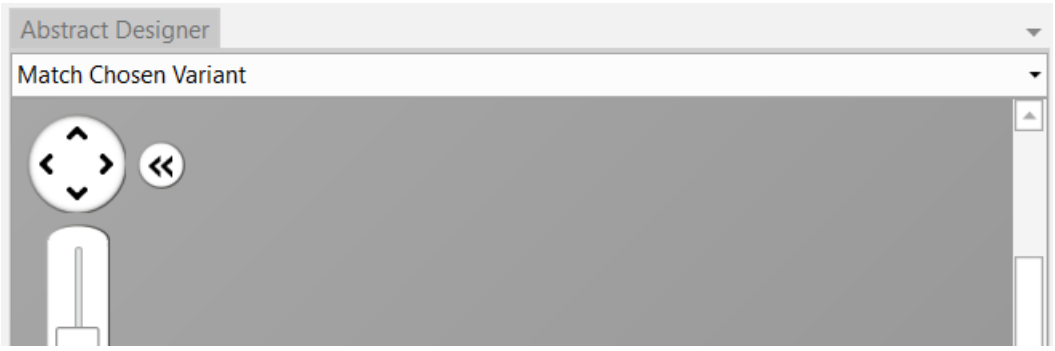
And the Designer looks like this.



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.

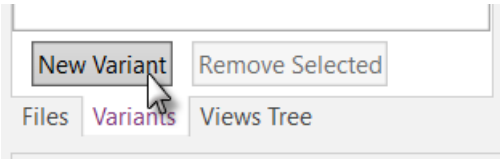


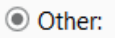
The dark gray area represents the screen area of the connected 'device' which is the WYSIWYG Form.

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

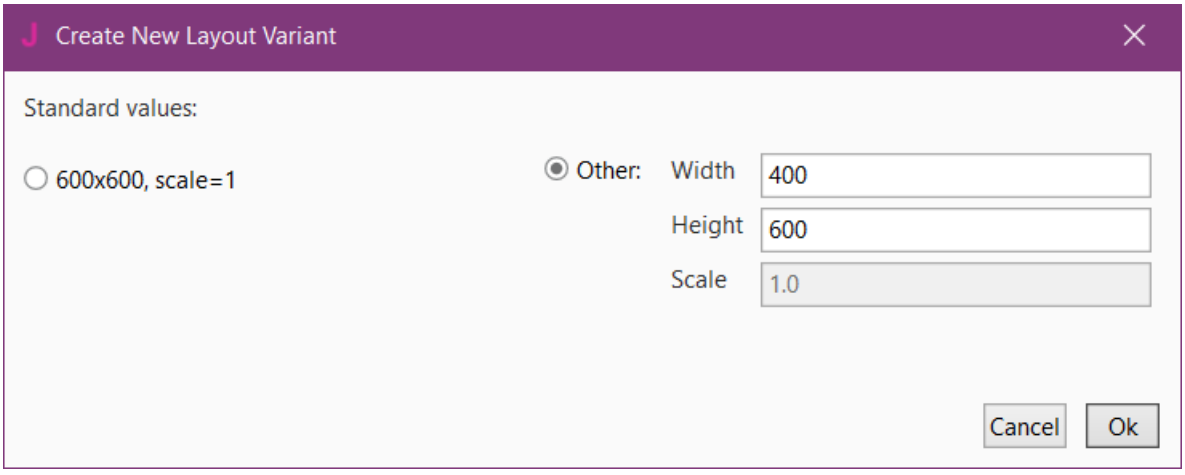
Click on 

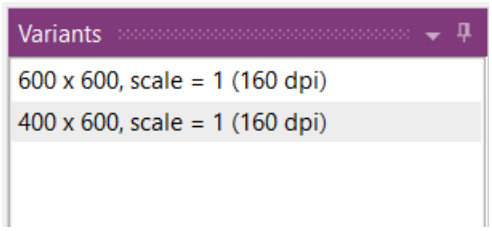
.



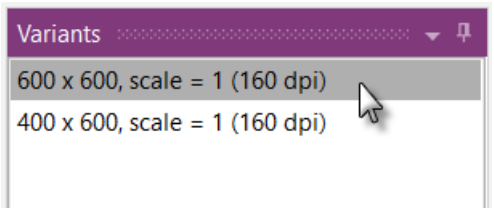
In this window click on 

and enter the two values.



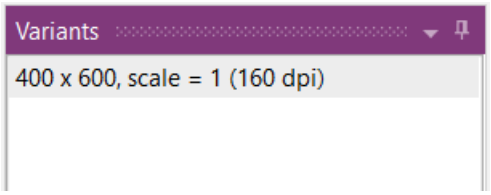


On top of the Variants window we see the new variant.

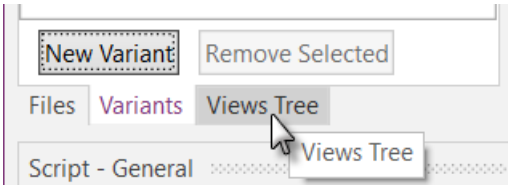


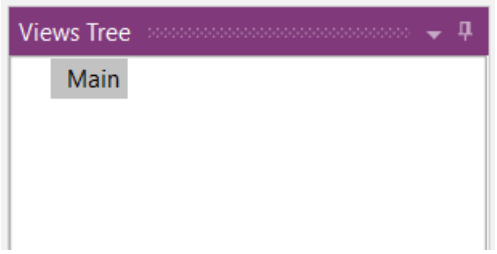
Click on 600 x 600, scale = 1 (160 dpi) to select the 600 * 600 variant.

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

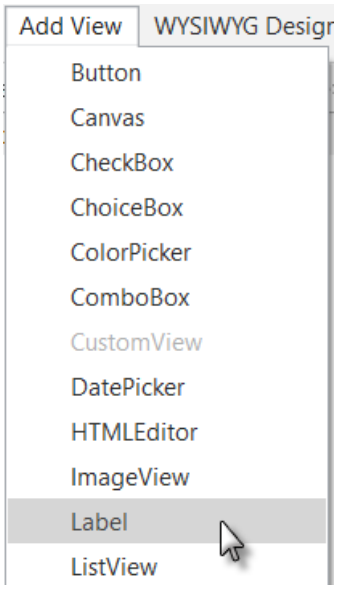


Click on Views Tree to show the Views Tree window.

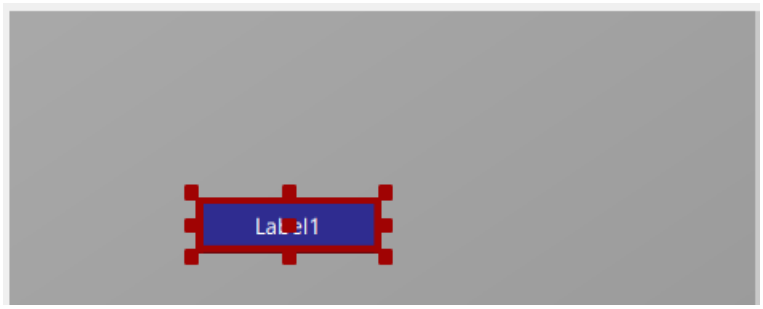




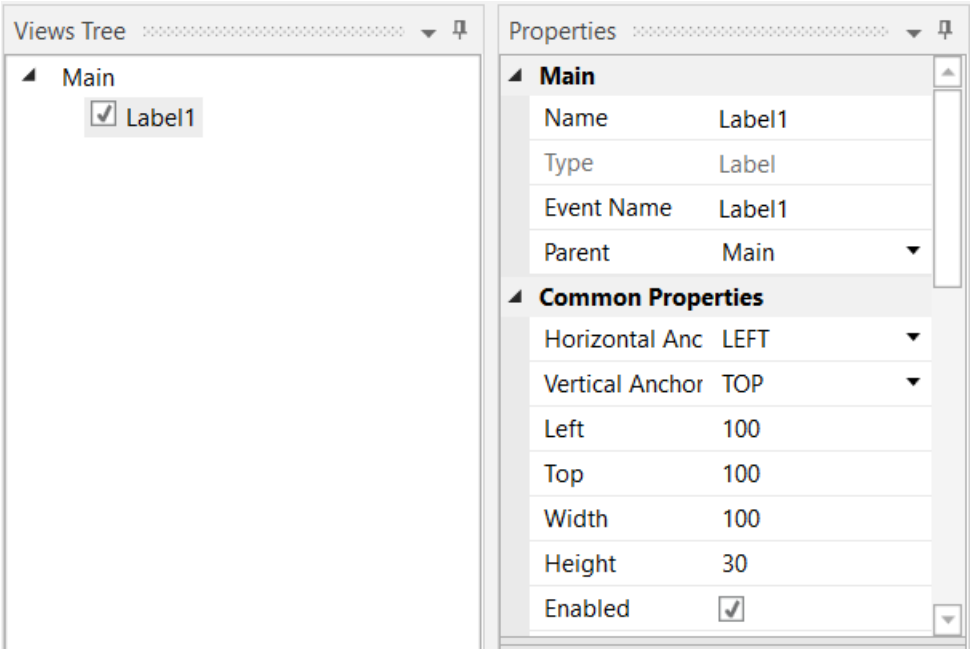
Now we will add the 2 Labels for the numbers.
In the Designer, add a Label.



In the IDE menu Add View click on Label.

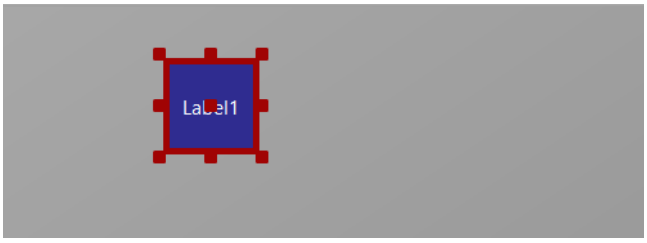


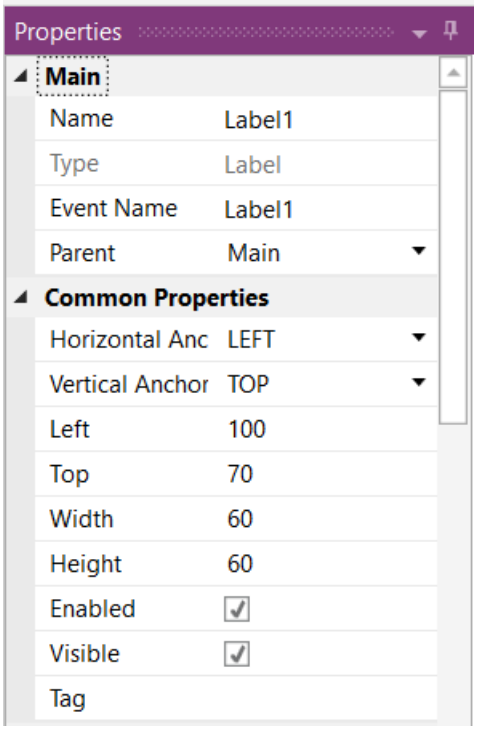
The Label appears in the Abstract Designer, in the Views Tree window and its default properties are listed in the Properties window.



And in the WYSIWYG Form.

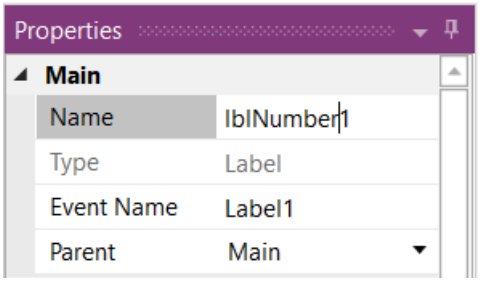
Resize and move the Label with the red squares like this.



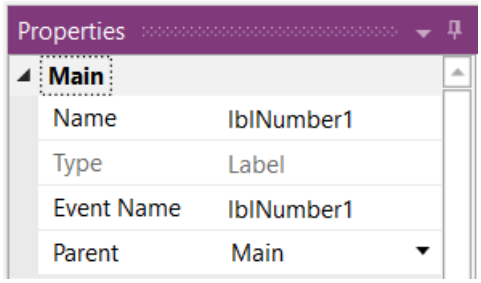


The new properties Left, Top, Width and Height are directly updated in the Properties window.
You can also modify the Left, Top, Width and Height properties directly in the Properties window.

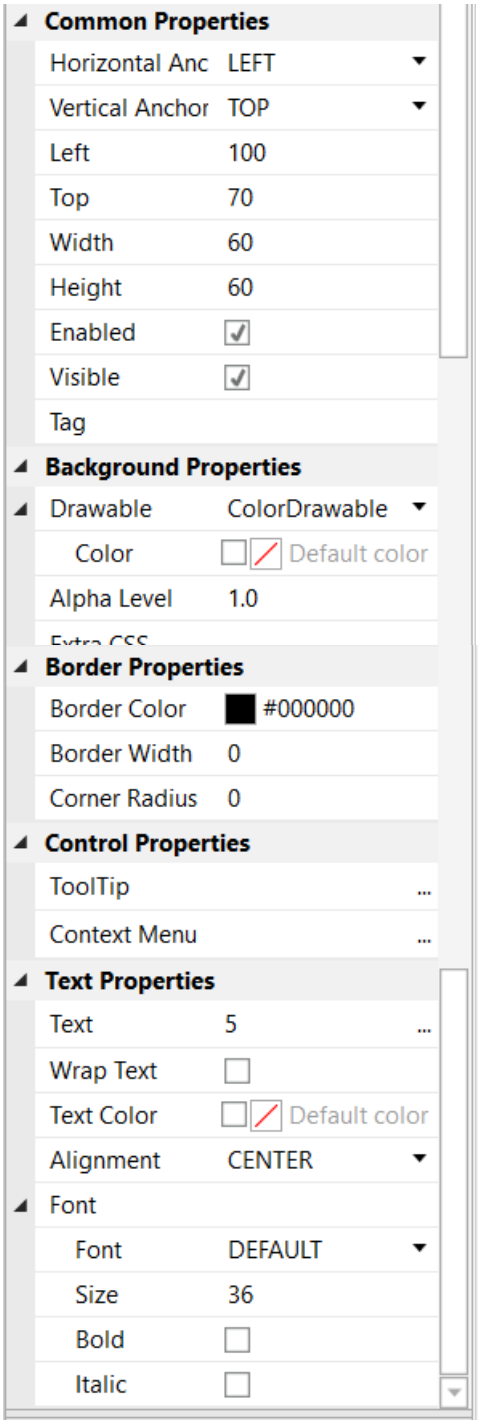
Let us change the properties of this first Label, per our requirements.
By default, the name is Label with a number, here Label1.
Let us change its name to lblNumber1.
The three letters 'lbl' at the beginning mean 'Label', and 'Number1' means the first number.
It is recommended to use significant names 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.

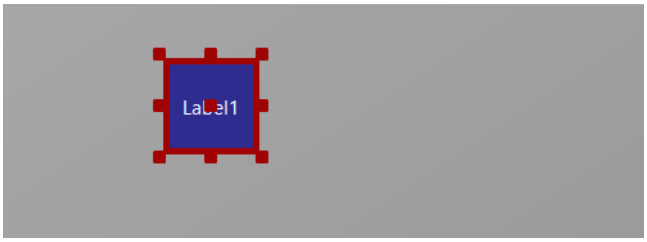
Visible is checked.

We leave the default colors.

Text set to 5

Set Text Alignment to Center.

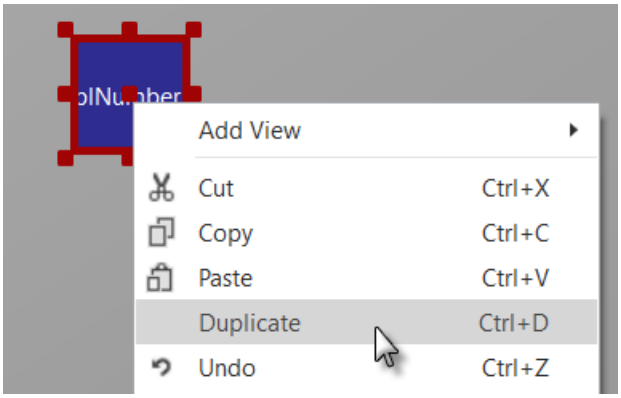
We leave the default Font.
Text Size, we set it to 36.



And the result in the Abstract Designer

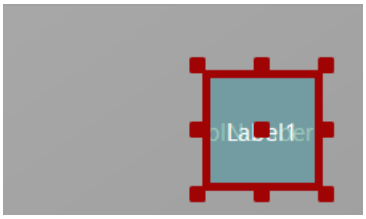
and on the WYSIWYG Form.

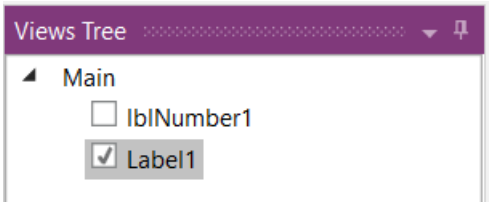
We need a second Label, like the first one. Instead of adding a new one, we copy the first one with the same properties. Only the Name and Left properties will change.



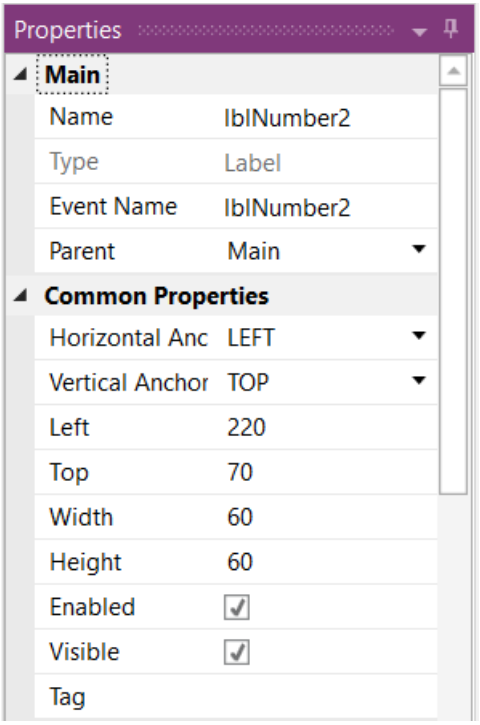
Right click on lblNumber1 and click on Duplicate in the popup menu.

The new label covers the previous one.





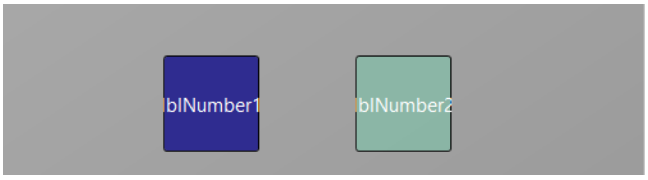
In the left part, in the Views Tree, you see the different nodes.
The new label Label1 is added.

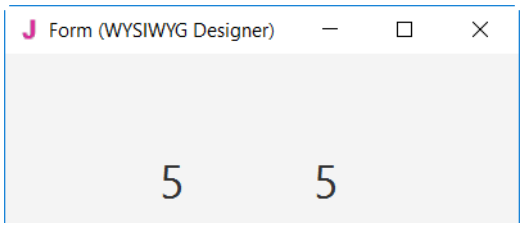


Let us position the new Label and change its name to lblNumber2.

Change the name to lblNumber2.

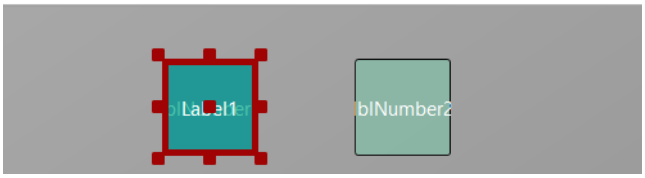
The Left property to 220.





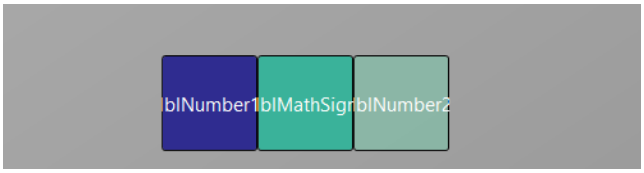
And the result in the Abstract Designer and on the WYSIWYG Form.

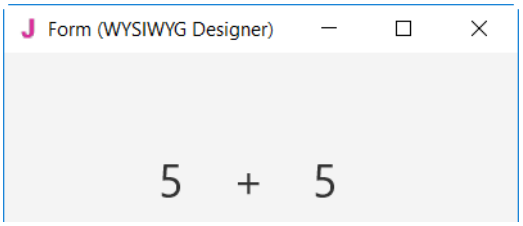
Let us now add a 3rd Label for the math sign. We copy once again lblNumber1.
Right click on lblNumber1 in the Abstract Designer and click on Duplicate in the popup menu.



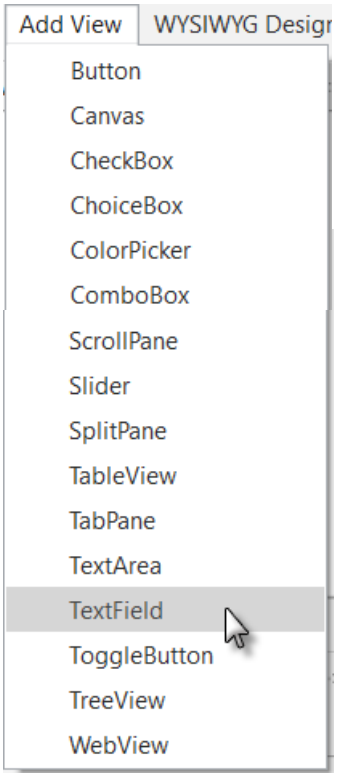
The new label covers lblNumber1.

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



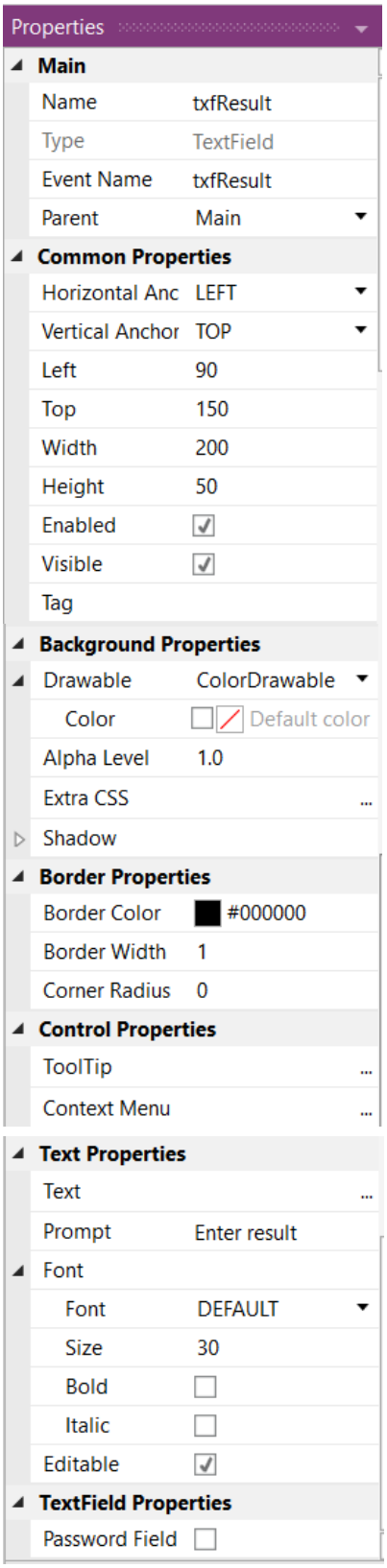


And the result in the Abstract Designer and on the WYSIWYG Form.



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.



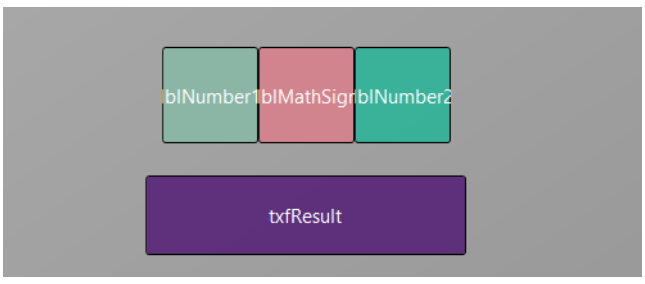
Change these properties.
Name to txfResult

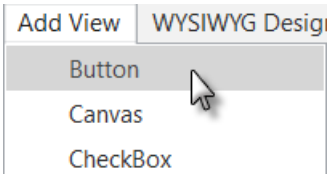
Left, Top, Width and Height.

Border Width to 1

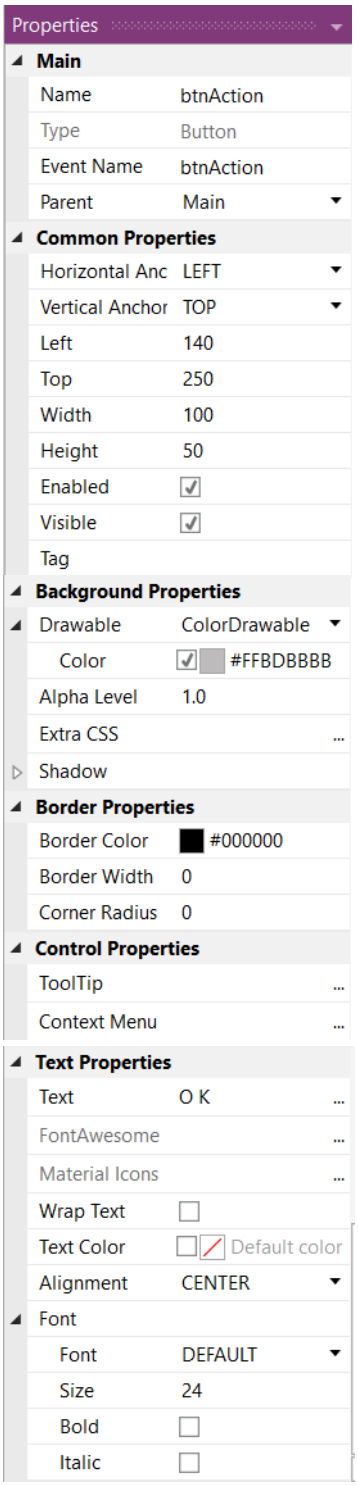
Prompt to Enter result
Prompt represents the text shown in the TextField node if no text is entered.
Size to 30

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

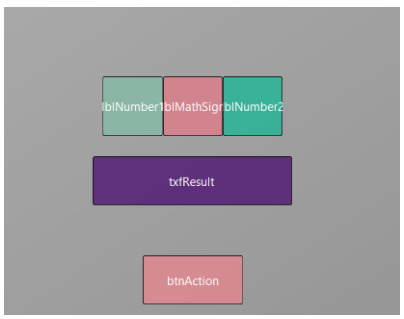
Left, Top, Width and Height.

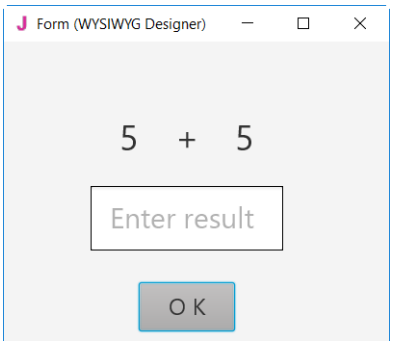
Background Color to #FFBDBBBB

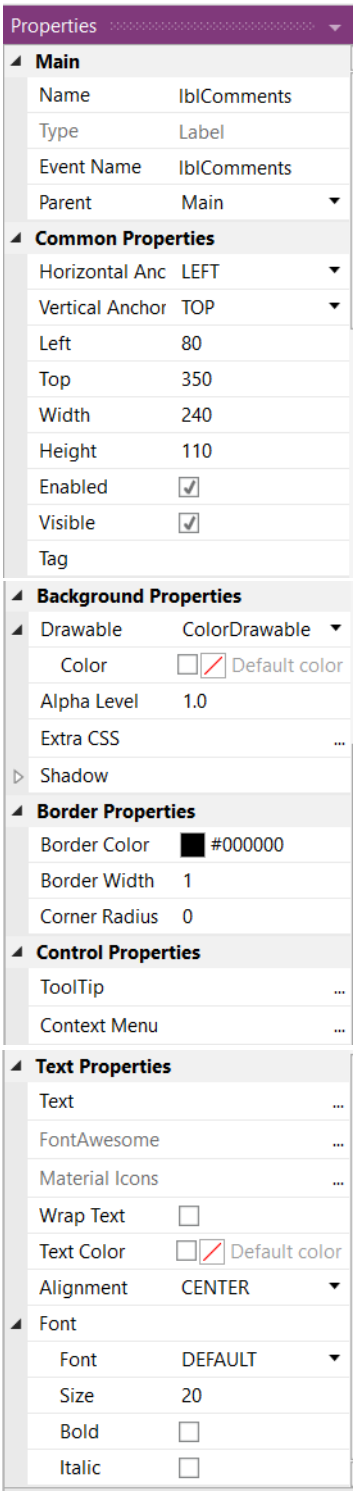
Text to O K (with a space between O and K)

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

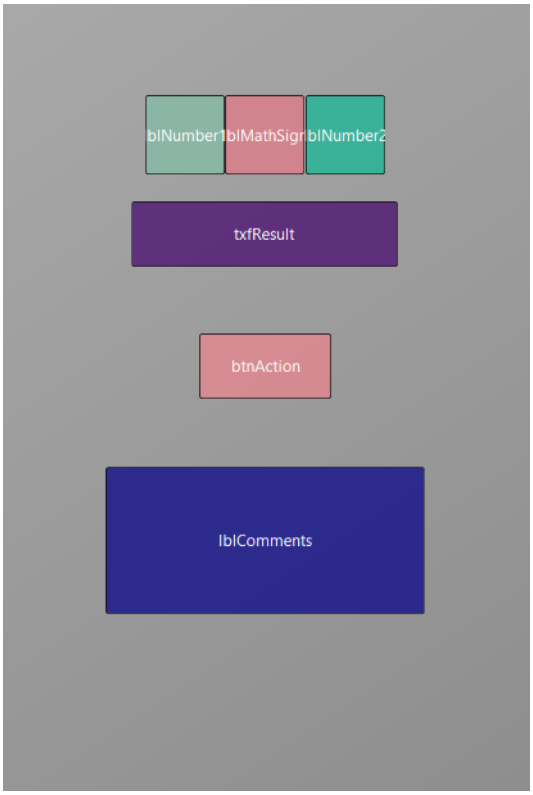
Left, Top, Width and Height.

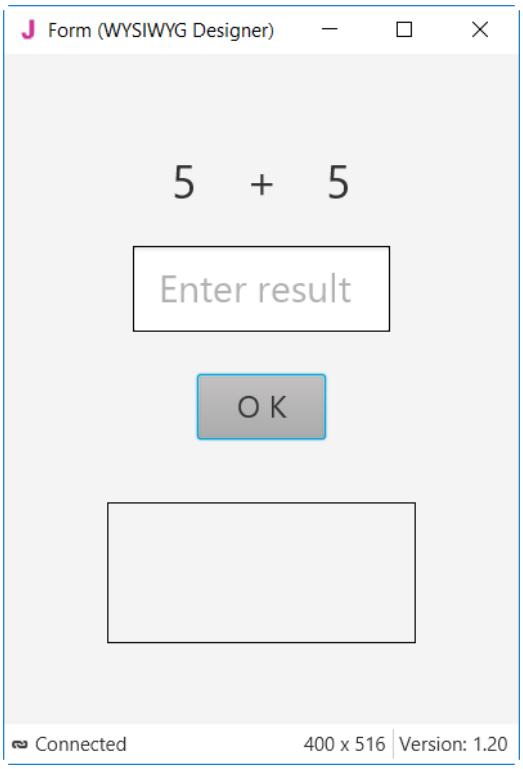
Border Width to 1

Text empty

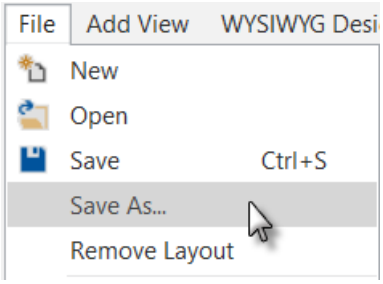
Font Size to 20

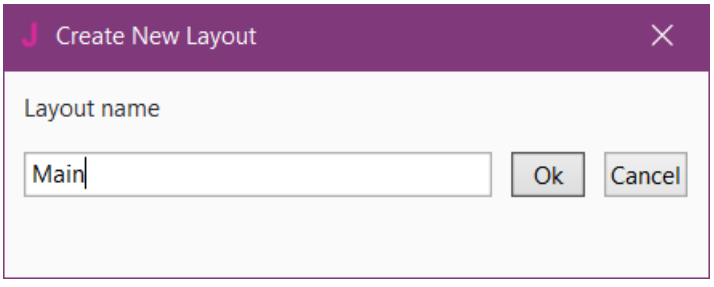
And the result.





Now we save the layout in a file.

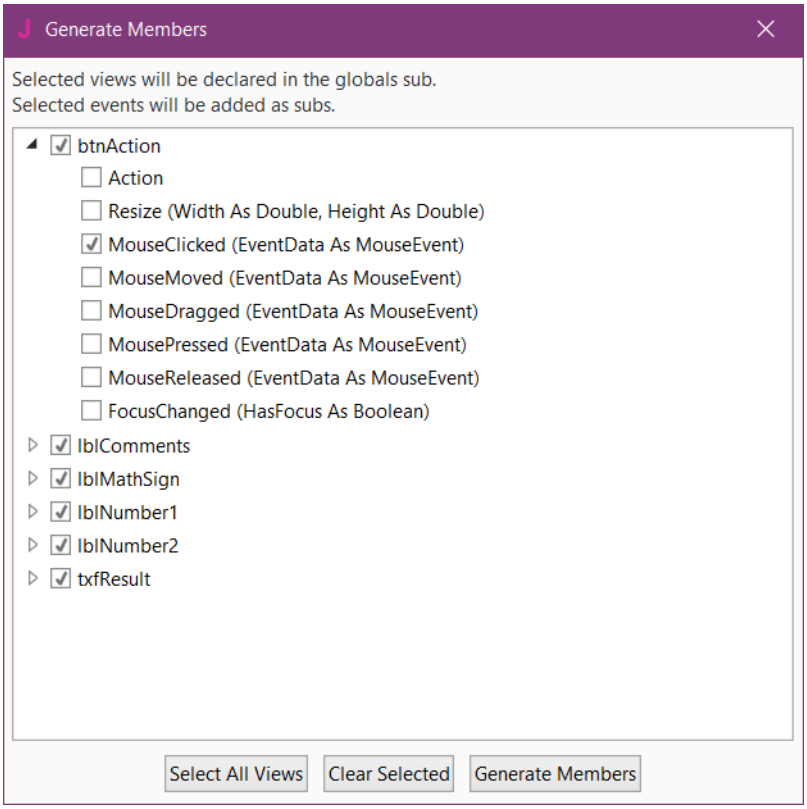
Click on  and save it with the name 'Main'.



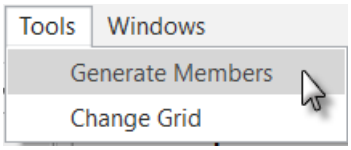
Click on  .

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.

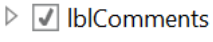
The *Generate Members* tool automatically generates references and subroutine frames.



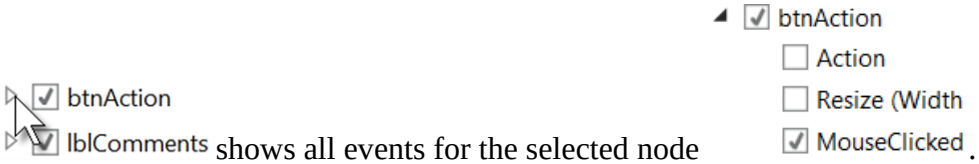
Click on



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.

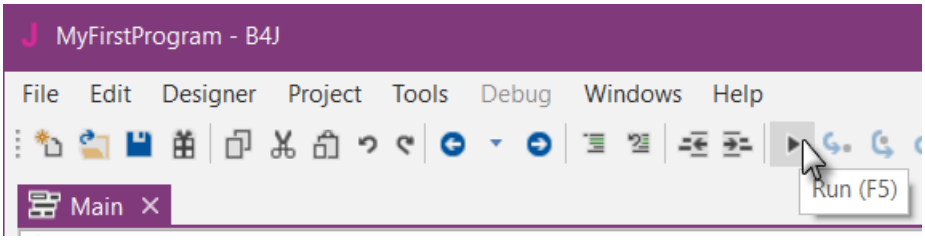
The last routine checks the result.

```
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
```

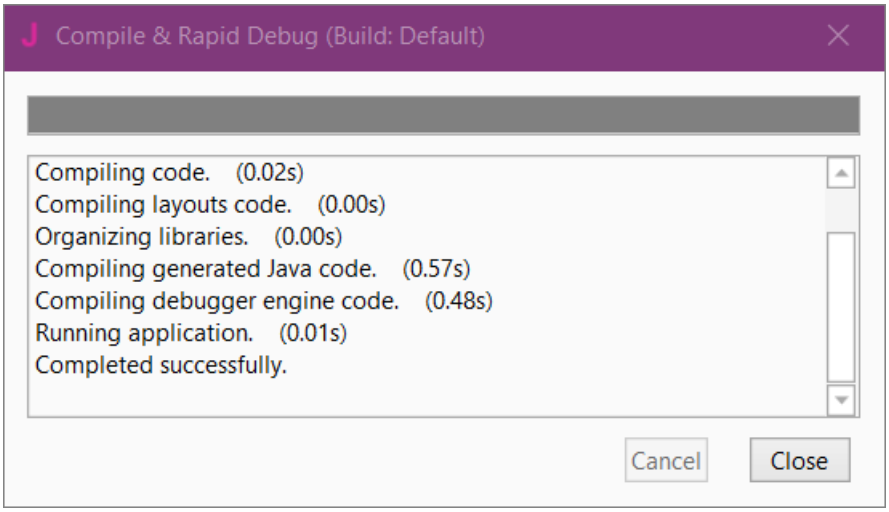
With `If txfResult.Text = Number1 + Number2 Then` we check if the entered result is correct.

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

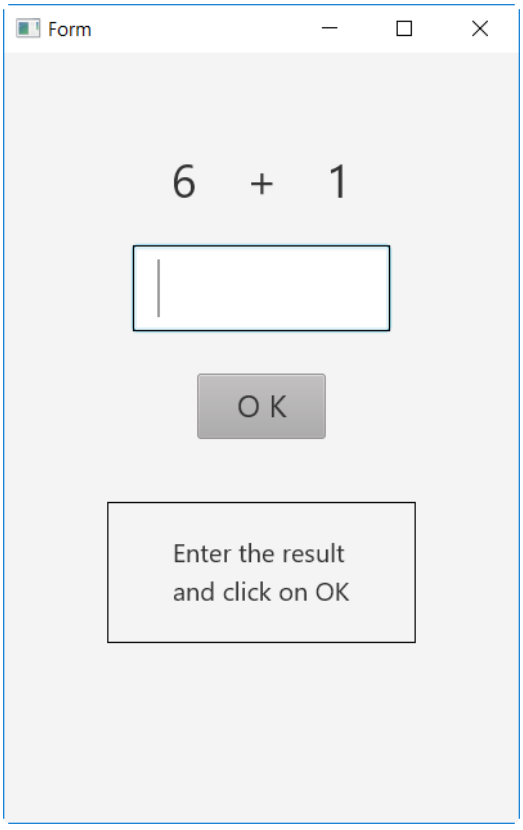
Let us now compile and run the program.
In the IDE on top click on  :



The program is going to be compiled.

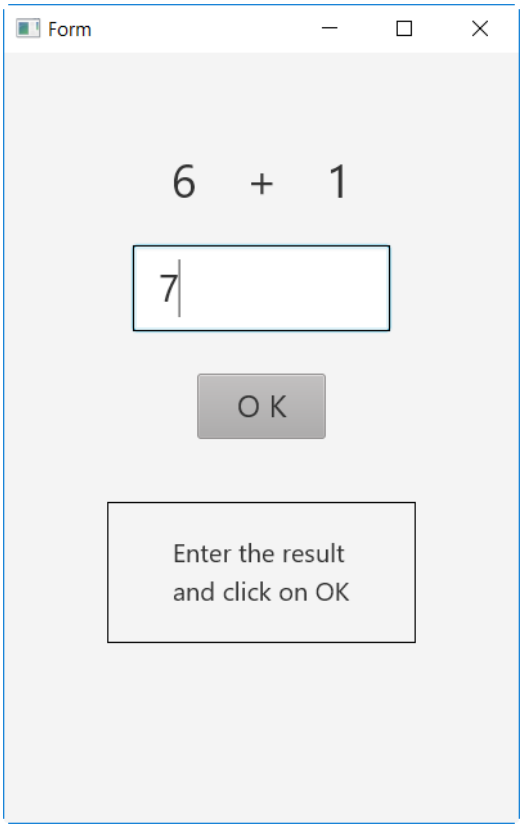


When you see `'Completed successfully.'` as in the message box, the compiling and transfer is finished.



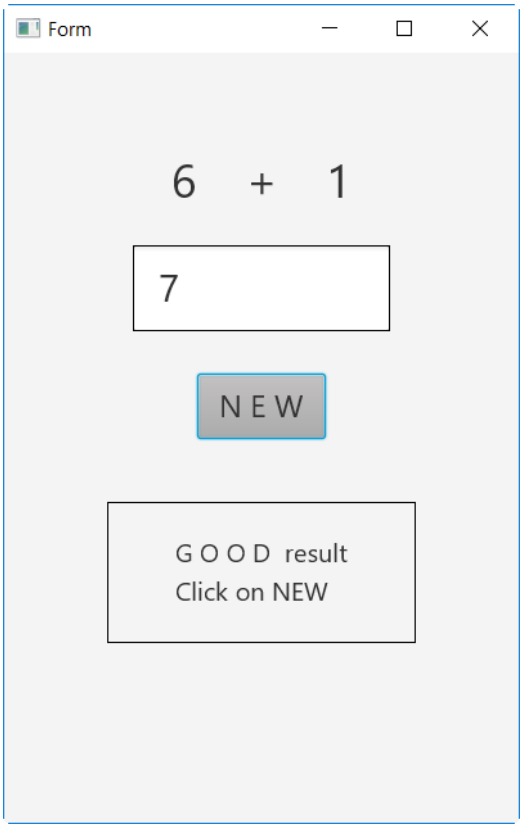
Then you should see something like the image on the left, with different numbers.

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



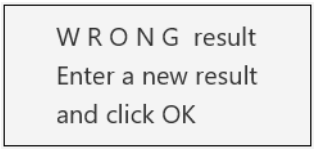
Enter 7

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:



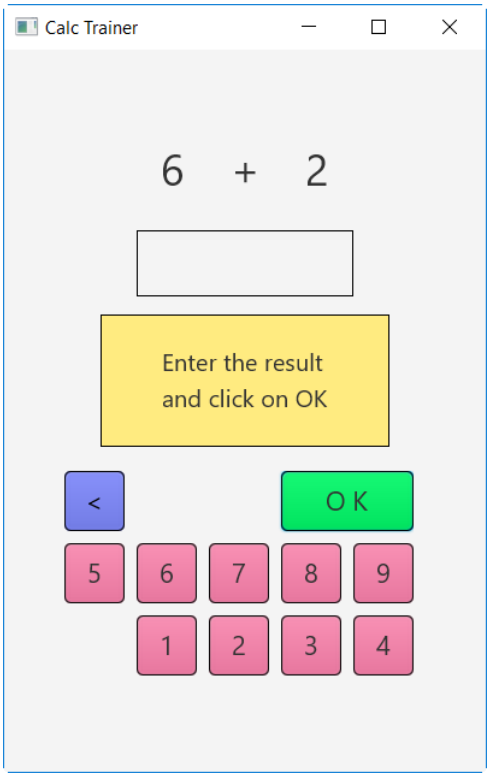
Click on  to define a new problem.

3 Second program

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.

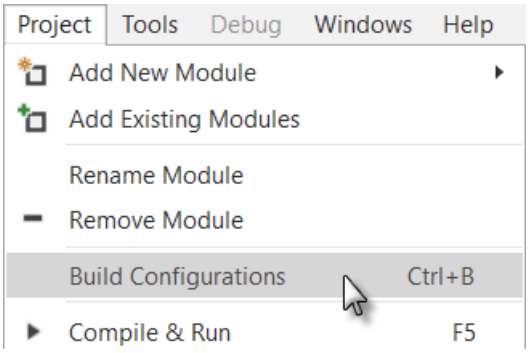


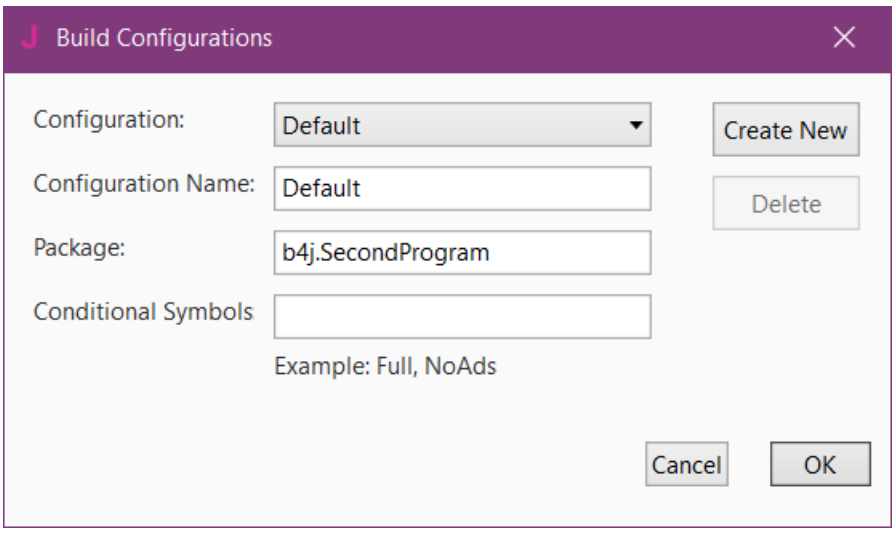
Load this new program in the IDE.

Run the Designer.

We need to change the Package Name.

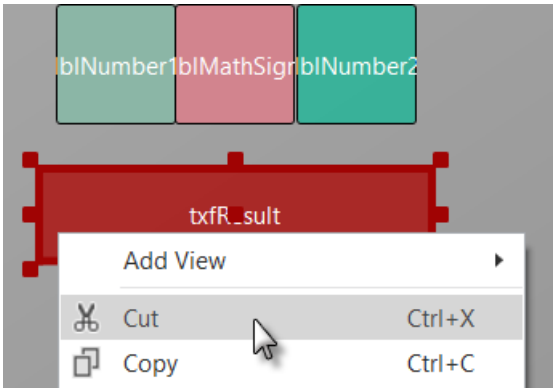
In the IDE Project menu.
Click on Build Configurations.



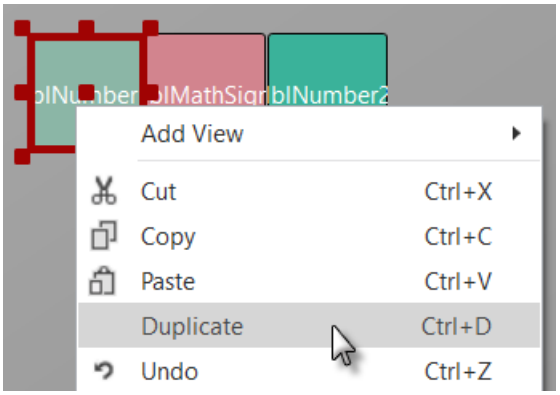


Change the Package name to b4j.SecondProgram and click on OK.

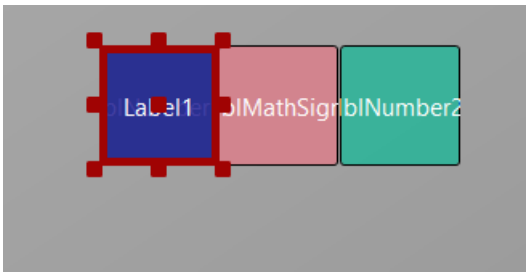
We want to replace the txfResult TextField node by a new Label.
In the Abstract Designer, click on the txfResult node.



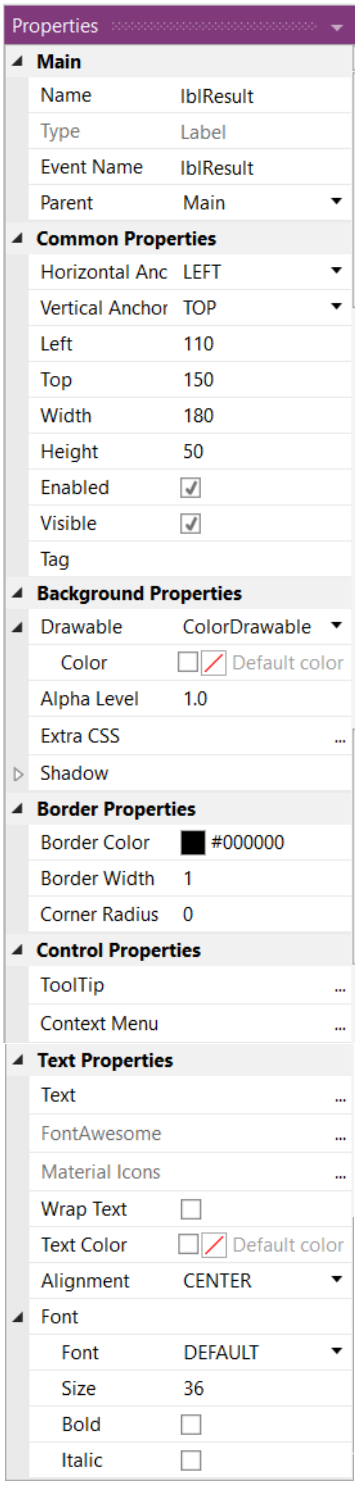
Right click on txfResult and click on Cut.



Right click on lblNumber1 and click on Duplicate.



The new label covers lblNumber1.



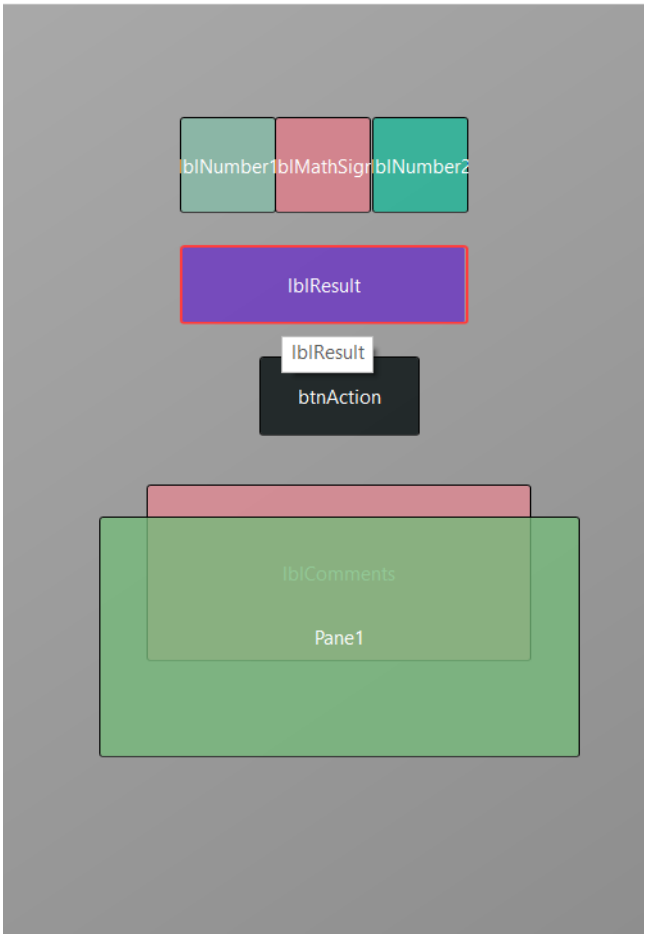
Modify the following properties:

Name to lblResult

Left, Top, Width, Height

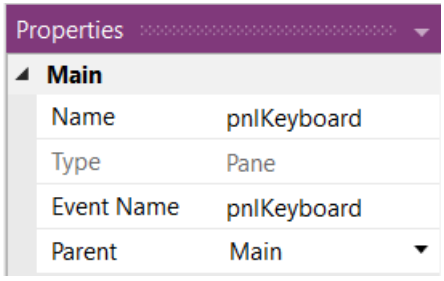
Border Width to 1

Text to "" no character

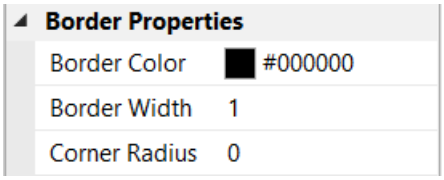


Let us add a Pane for the keyboard buttons.

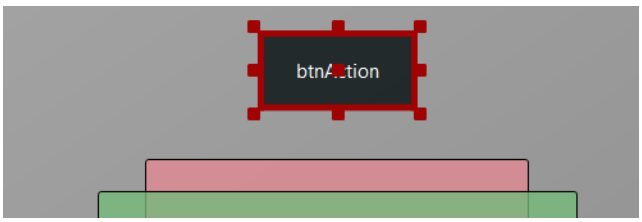
Position and resize it as in the image.



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

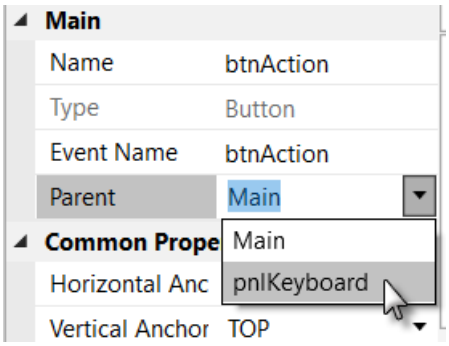


Change
Corner radius to 0

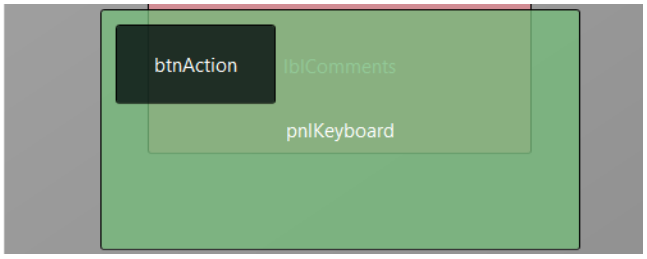


We will move btnAction from the Activity to the pnlKeyboard Panel.

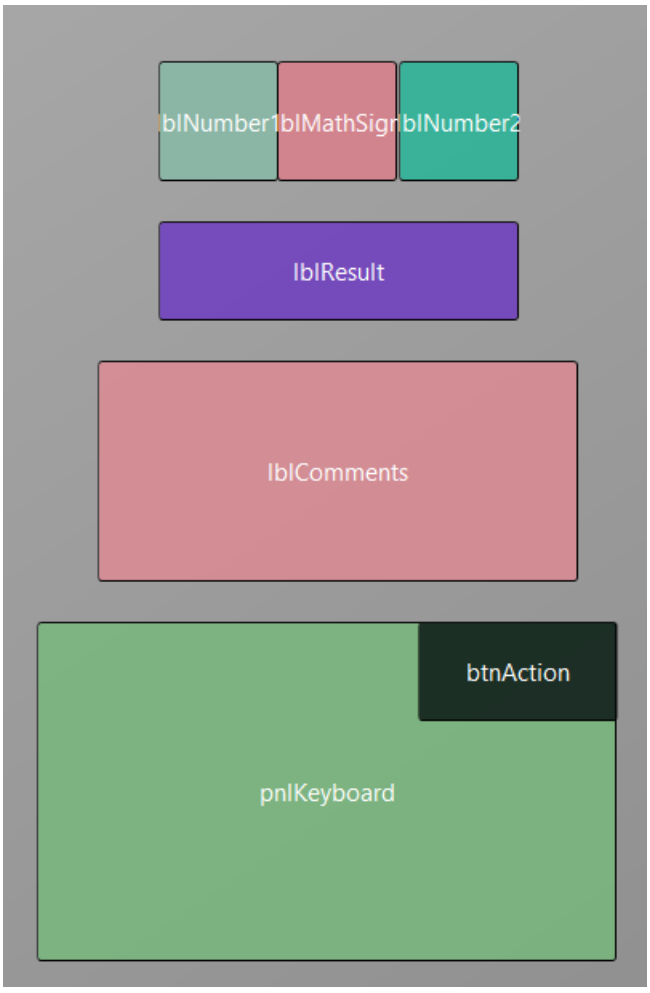
Click on btnAction.



In the Parent list click on pnlKeyboard.



The button now belongs to the Pane.

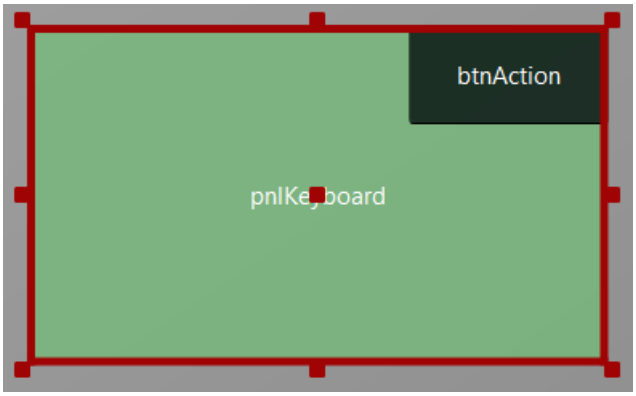


Now we rearrange the nodes to get some more space for the keyboard.

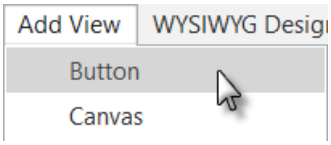
Set the properties below:

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.

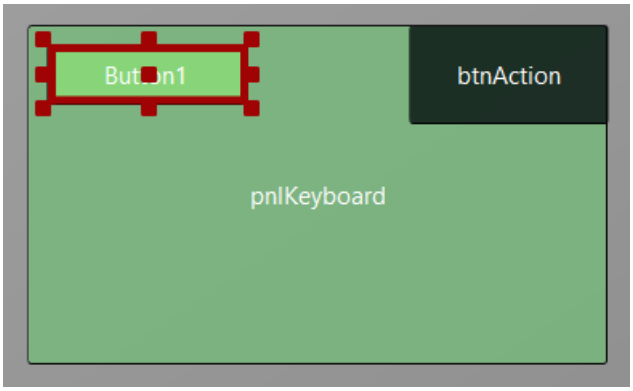


Click on the pnlKeyboard panel to select it.

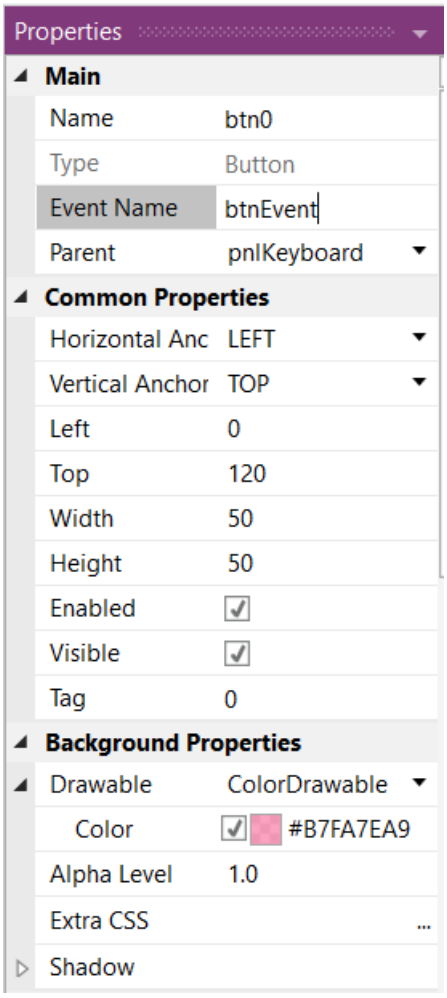


Click on

to add a new button.



The new button is added.



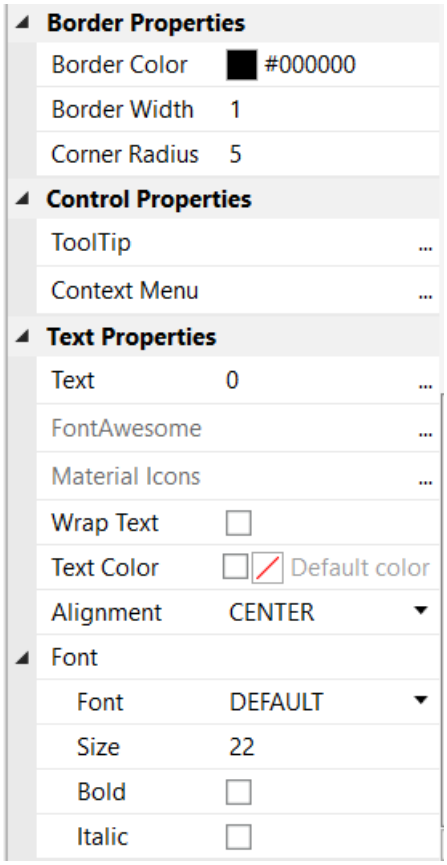
Change following properties:

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

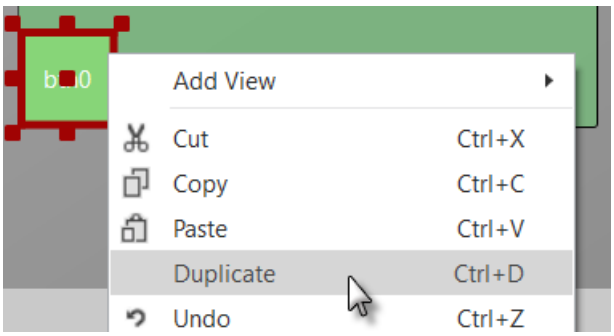


Border Width to 1
Corner Radius to 5

Text to 0

Size to 22

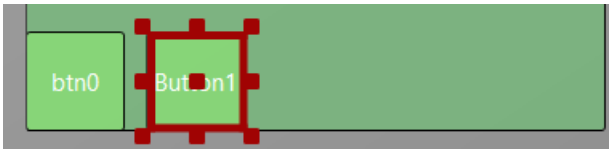
The button looks now like this.



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

Select the Button btn0.

Right click on btn0
and click on Duplicate.



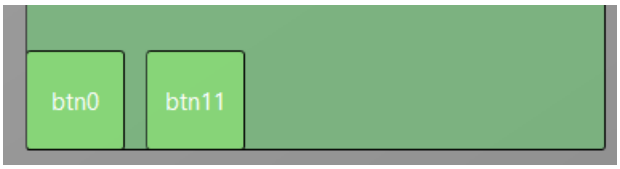
Move the new Button next to the previous one with a space.

Main	
Name	btn11
Tag	1
Text	1 ...

Change the following properties:
Name to btn1

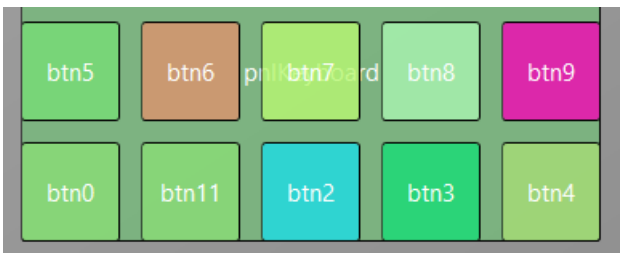
Tag to 1

Text to 1



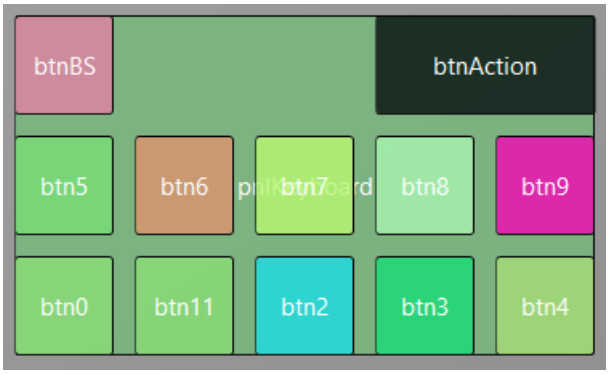
And the result.

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.

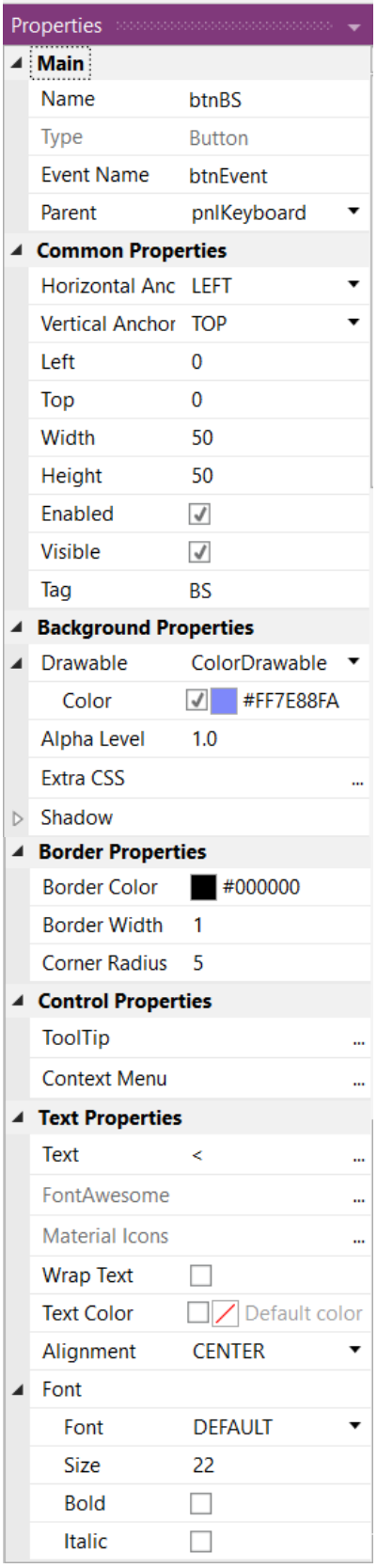


To create the BackSpace button, duplicate one of the number buttons, and position it in the top left corner.

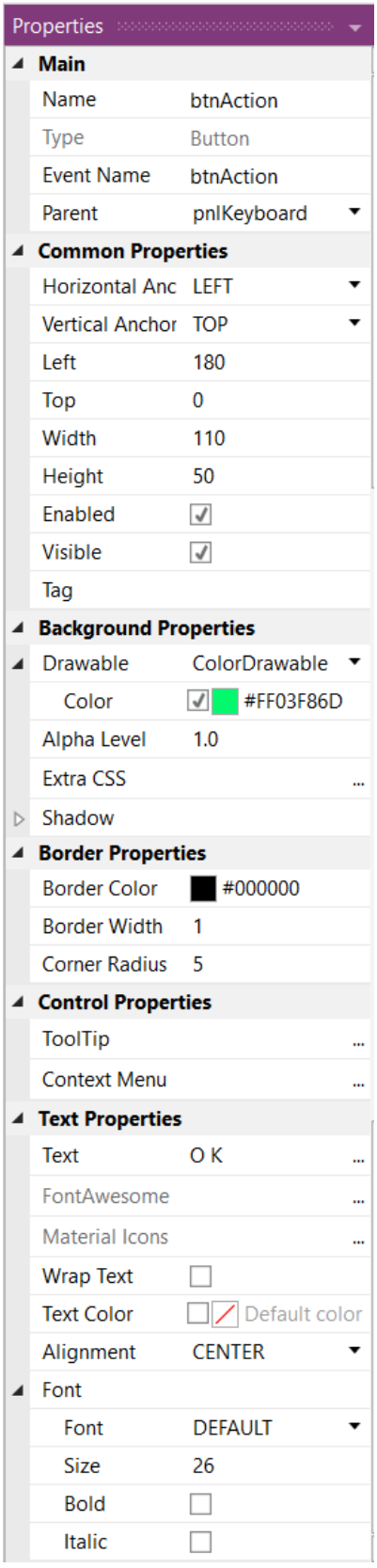
Resize and position btnAction.

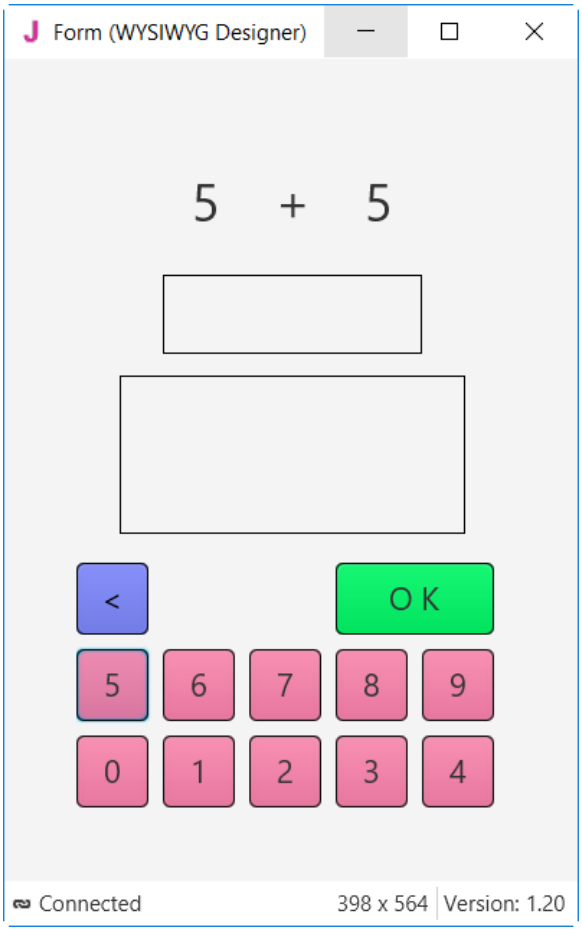
Change their Name, Tag, Text and Color properties as below.

btnBS <



btnAction O K





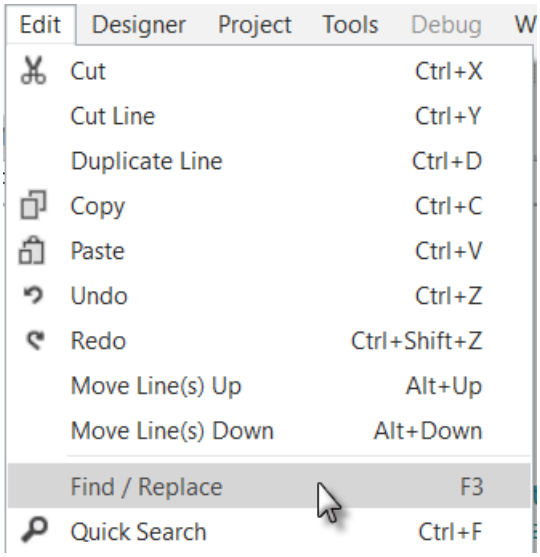
The finished new layout in the WYSIWYG form.

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

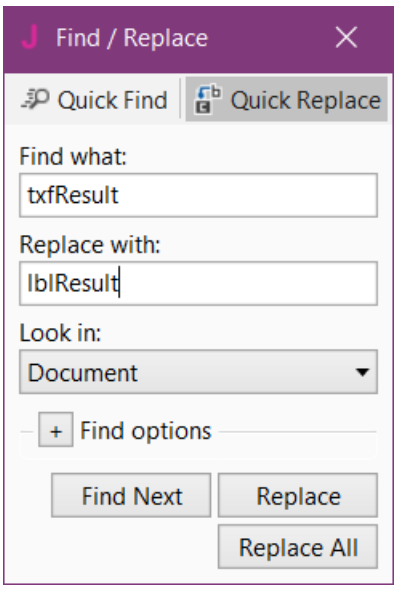
Now we will update the code.
First, we must replace the txfResult by lblResult because we replaced the TextField node by a Label.

```
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.

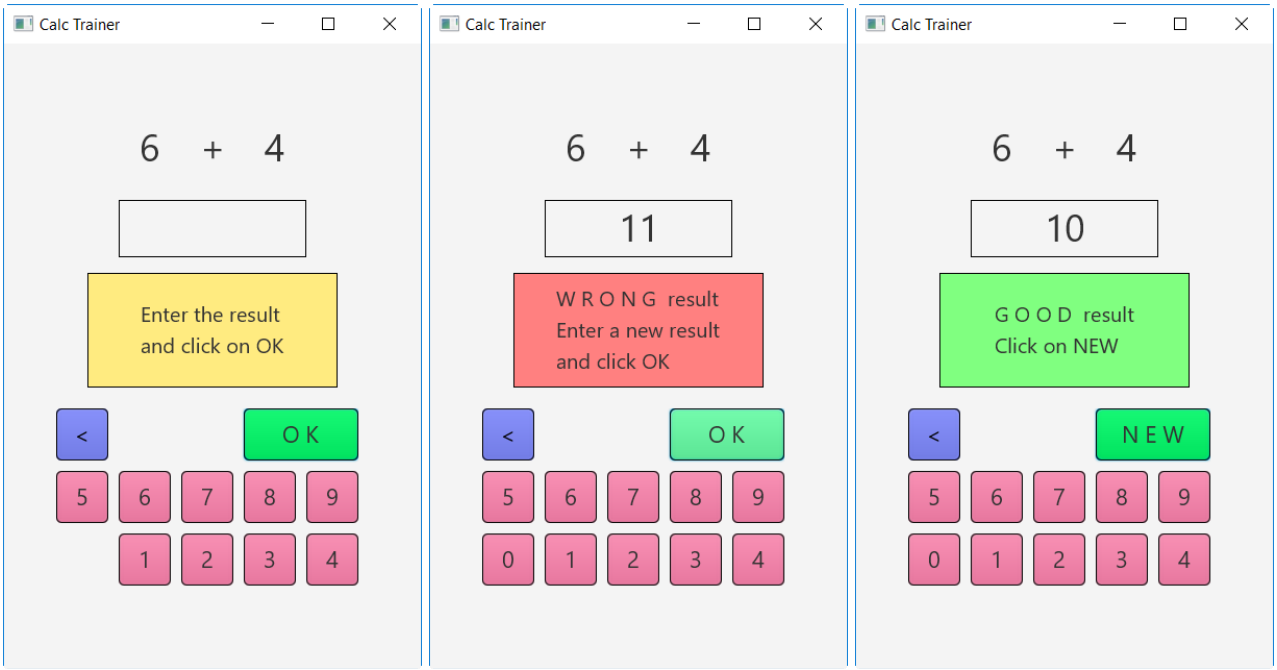
We first modify the New routine, where we add this line
`CSSUtils.SetBackgroundColor(lblComments, fx.Colors.RGB(255,235,128))`
We need to use the CSSUtils library for that! See next page how to add a library.

```
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
```

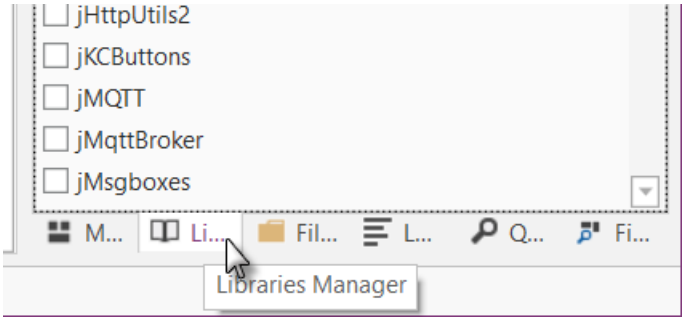
And in the CheckResult routine we add the two lines with `CSSUtils.SetBackgroundColor`...

```
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
```

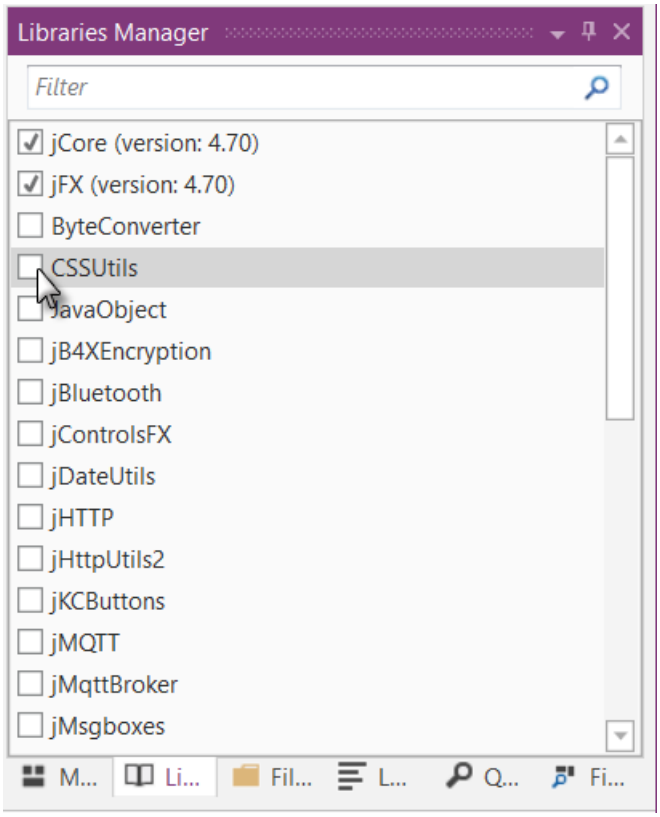
And we give the program a more meaningful title by adding `MainForm.Title = "Calc Trainer"` in AppStart just after `MainForm.Show`.



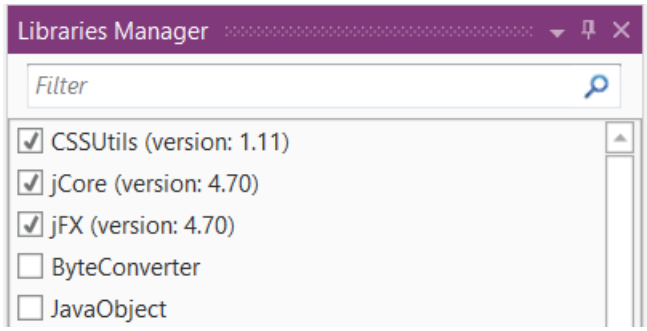
Add the CSSUtils library to the project.



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.

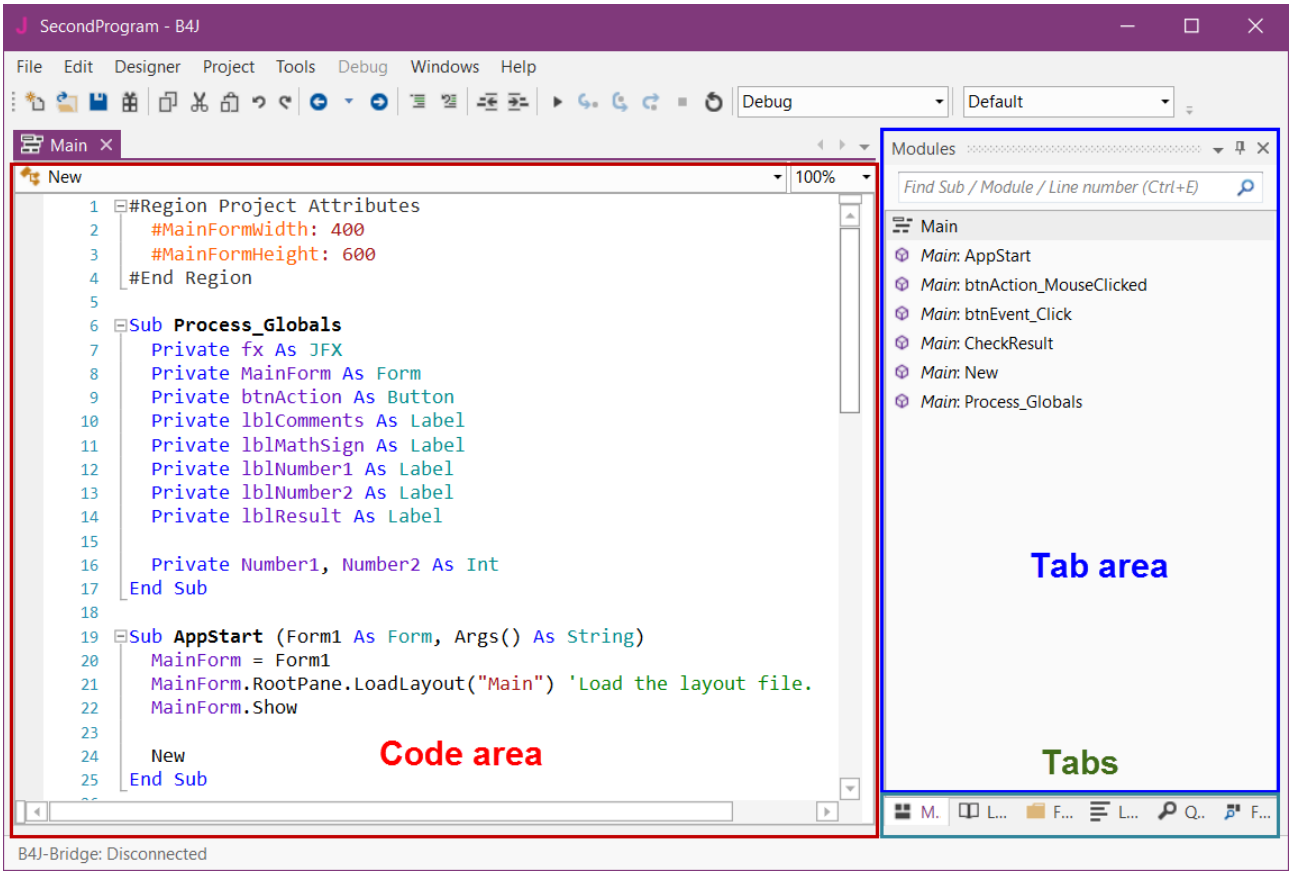
```
btn0.Visible = False
```

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
```

4 The IDE

The **I**ntegrated **D**evelopment **E**nvironment.
When you run the IDE you will get a form like the image below:



You see 3 main areas:

- Code area The code editor
- Tabs area Window showing different data depending on the selected Tab.
- Tabs Tabs for different settings.

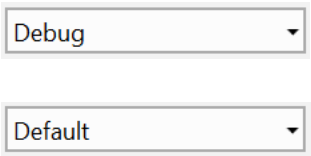
4.1 Menu and Toolbar

4.1.1 Toolbar

Generates a new empty project.
Loads a project.
Saves the current project.
Export as zip, exports the whole project in a zip file.
Copies the selected text to the clipboard.
Cuts the selected text and copies it to the clipboard.
Pastes the text in the clipboard at the cursor position.
Undoes the last operation.
Redoes the previous operation.
Navigate Backwards.
Navigation History.
Navigate Forward.
Sets the selected lines as comments.
Uncomments the selected lines.
Decrease the indentation of the selected lines.
Increase the indentation of the selected lines.
Runs the compiler.

The 5 functions below are active only when the debugger is active.
Details in Debugging.

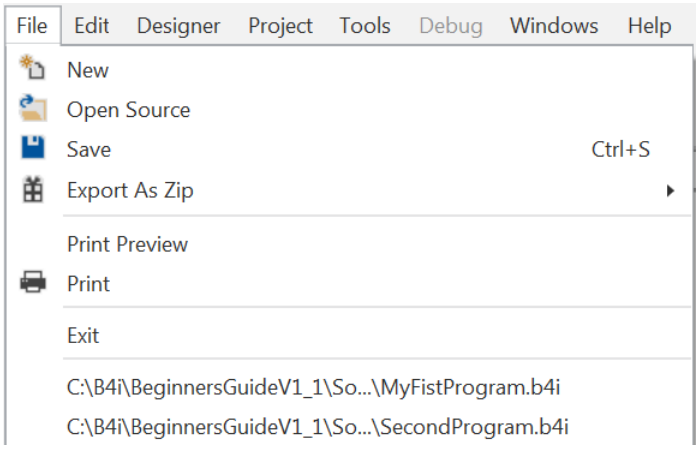
Step In [F8].
Step Over [F9].
Step Out [F10].
Stop.
Restart [F11].



Compiler options list, currently only Debug.

Build Configuration.

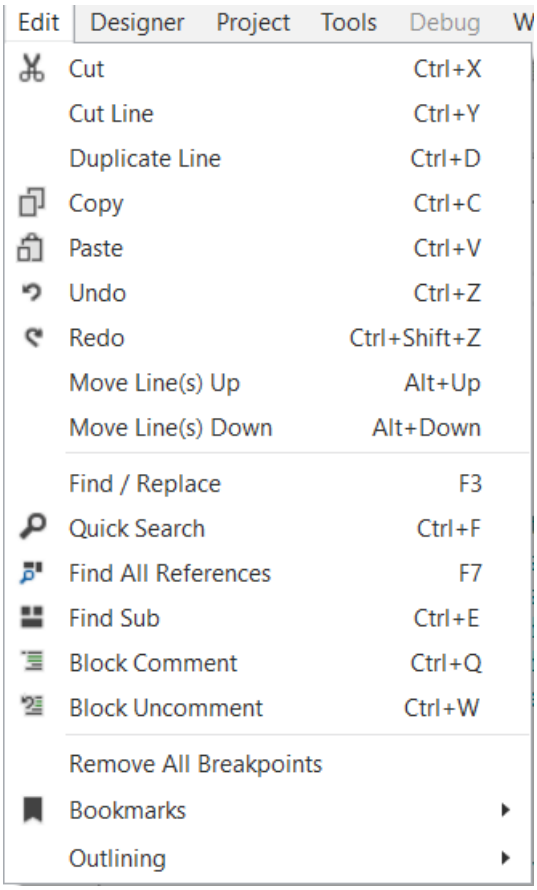
4.1.2 File menu



New Generates a new empty project.
Open Source Loads a project.
Save Saves the current project.
Export As Zip
Exports the whole project in a zip file.
Print Preview Shows a print preview.
Print Prints the code.
Exit Leaves the IDE.

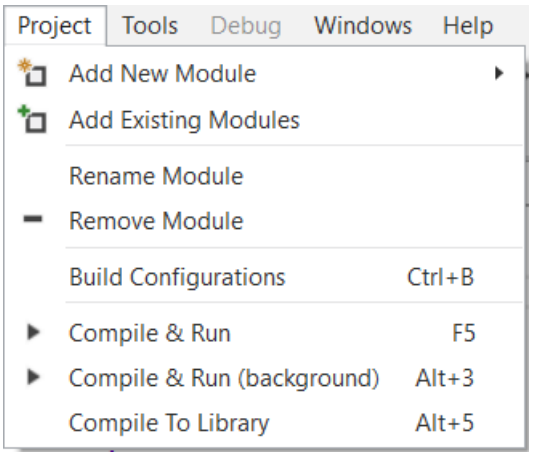
List of last loaded programs.

4.1.3 Edit menu



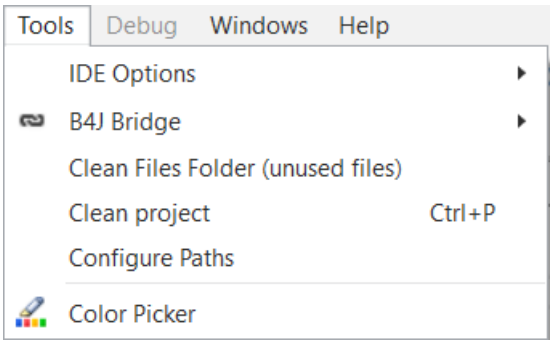
Cut Cuts the selected text and copies it to the clipboard.
Cut Line Cuts the line at the cursor position.
Duplicate Line Duplicates the selected line
Copy Copies the selected text to the clipboard.
Paste Pastes the text in the clipboard at the cursor position.
Undo Undoes the last operation.
Redo Redoes the previous operation.
Move Line(s) Up Moves the selected lines up.
Move Line(s) Down Moves the selected lines down.
Find / Replace Activates the Find and Replace function.
Quick Search Quick search function
Find All References Finds all References of a selected item
Find Sub Finds the selected Sub
Block Comment Sets the selected lines as comments.
Block Uncomment Uncomments the selected lines.
Remove All Breakpoints Breakpoints.
Bookmarks Bookmarks
Outlining Collapse the whole code.

4.1.4 Project menu



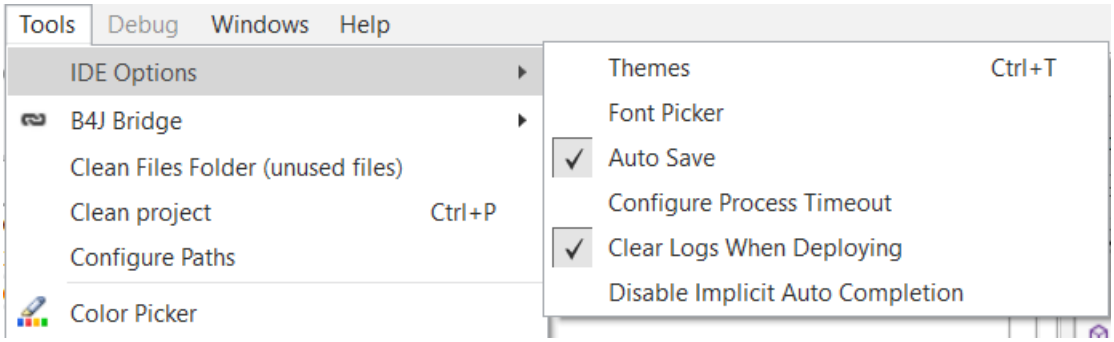
Adds a new module
Adds an existing module
Changes the module name
Removes the current module
Build Configurations Changes the package name.
Compiles and runs the program.
Compile & Run in the background.
Compiles to a library.

4.1.5 Tools menu



IDE Options
B4J Bridge, sets the IP address for connection with Wifi
Clean Files Folder (unused files)
Clean Project
Configure Paths
Color Picker

4.1.5.1 IDE Options



Themes

Font Picker

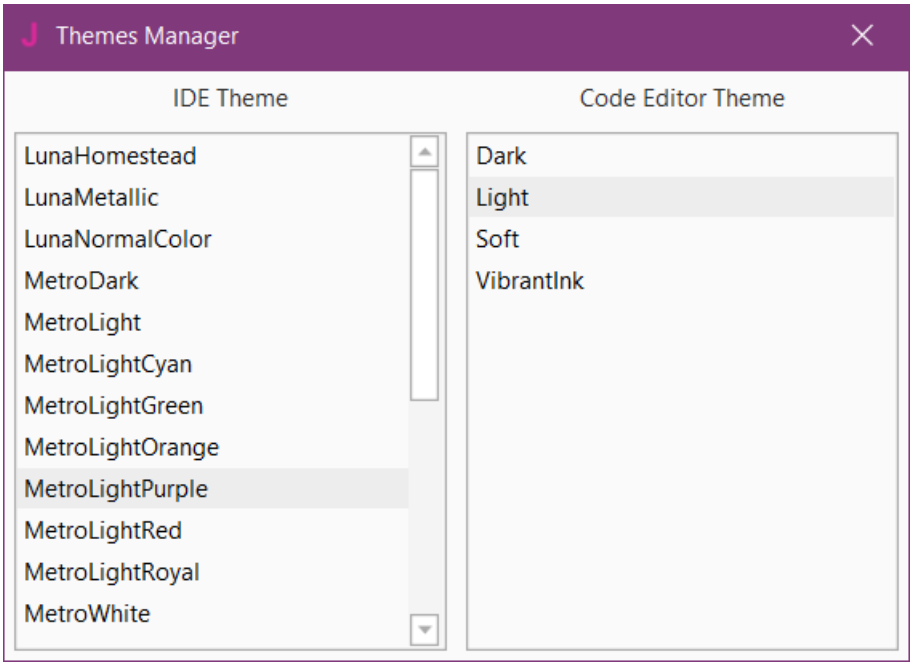
Auto Save — Saves the program every time you run it.

Configure Process Timeout

Clear Logs When Deploying — Removes all Log statements when compiled in Release mode.

Disable Implicit Auto Completion

4.1.5.1.1 Themes

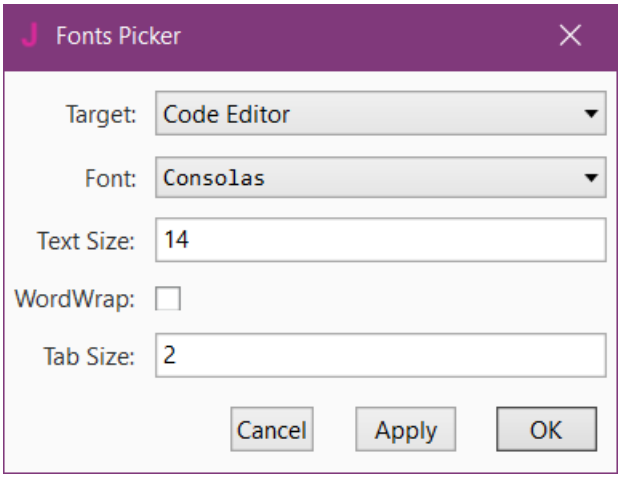


You can select different themes for the IDE.

The default theme is MetroLightPurple.

When you select one you see directly the new colors.

4.1.5.1.2 Font Picker



You can select a different font and text size.

Code Editor or for the Logs.

Select the font.

Enter the text size.

Select WordWrap

Enter the Tab size.

4.1.5.1.2.1 Word wrap

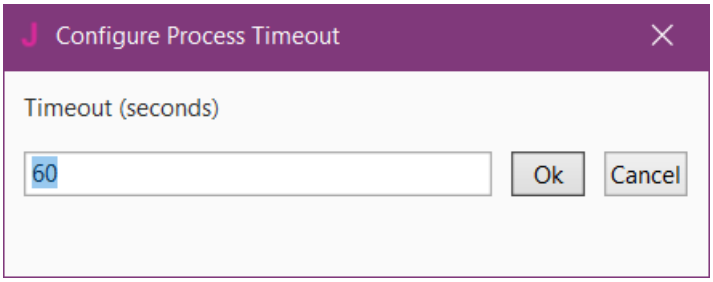
Without word wrap. The end of the line is hidden.

```
66      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new result"
```

With word wrap. The end of the line is wrapped to the next line.

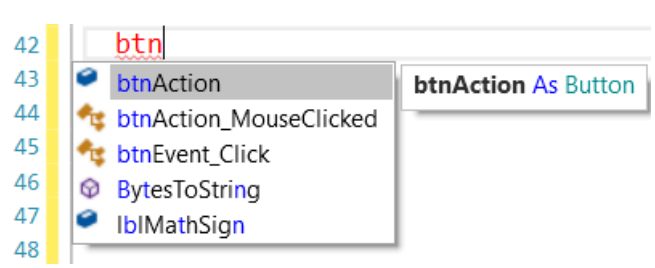
```
66      lblComments.Text = "W R O N G  result" & CRLF & "Enter a new
result" & CRLF & "and click OK"
```

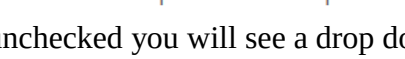
4.1.5.1.2.2 Configure Process Timeout



Sometimes the compilation needs more time.
If you get a message 'Process timeout' you can increase the time.

4.1.5.1.2.3 Disable Implicit Auto Completion



If  is unchecked you will see a drop down list with possible words during typing.

If checked  you won't see the auto completion list.

4.1.5.2 Clean Files Folder (unused files)

Deletes files that are located under the Files folder but are not used by the project (it will not delete any file referenced by any of the project layouts).
A list of unused files will be displayed before deletion (and you may cancel the operation).

4.1.5.3 Clean Project

Deletes all files that are generated during compilation.

4.2 Code area

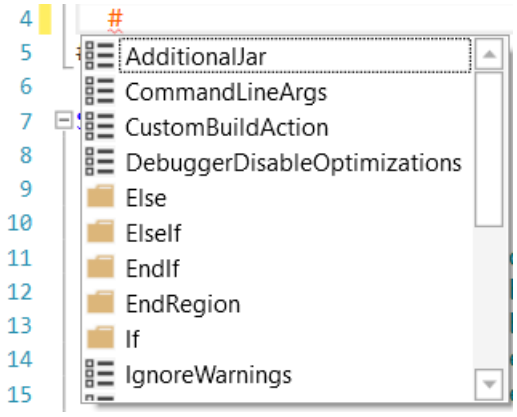
The code of the selected module is displayed in this area and can be edited.
The examples below are based on the code of the SecondProgram.

4.2.1 Code header Project Attributes

On top of the code you find the Project Attributes.

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

When you want to add a new Attribute you can just write # and the inline help shows all possibilities.



4.2.1.1 #MainFormWidth

This project attribute sets the Form Width.

4.2.1.2 #MainFormHeight

This project attribute sets the Form Height.

4.2.1.3 #If / #End If

It is possible to add #If / #End If structures in the code for different compiler options.
Example in the Build Configurations tutorial in the forum. The example is for B4A but the principle is the same.

4.2.1.4 #Region / #End Region

You can define regions in your code and collapse them. Details in Collapse a Region.

4.2.1.5 #IgnoreWarnings

The compiler adds warnings in the Log Tab, you can ignore warnings.
Details in Test Compile / Warnings.

4.2.2 Undo – Redo

In the IDE, it is possible to undo the previous operations and redo undone operations.
Click on to undo and on to redo.

4.2.3 Collapse a subroutine

In the IDE, a subroutine can be collapsed to minimize the number of lines displayed.

```
55 Sub btnAction_Click
56   If btnAction.Text = "O K" Then
57     If txfResult.Text="" Then
58       Msgbox("No result entered","E R R O R")
59     Else
60       CheckResult
61     End If
62   Else
63     New
64     btnAction.Text = "O K"
65   End If
66 End Sub
```

The btnAction_Click routine expanded.

Click on to collapse the subroutine.

The btnAction_Click routine collapsed.

```
54
55 Sub btnAction_Click
67
```

Hovering with the mouse over the collapsed routine name shows its content.

```
54
55 Sub btnAction_Click
67   Sub btnAction_Click
68     If btnAction.Text = "O K" Then
69       If txfResult.Text="" Then
70         Msgbox("No result entered","E R R O R")
71       Else
72         CheckResult
73       End If
74     Else
75       New
76       btnAction.Text = "O K"
77     End If
78   End Sub
```

4.2.4 Collapse a Region

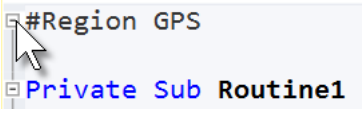
You can define 'Regions' in the code, which can be collapsed.

Example:

```
#Region GPS
#End Region
#Region GPS
Private Sub Routine1
End Sub
Private Sub Routine2
End Sub
Private Sub Routine3
End Sub
#End Region
```

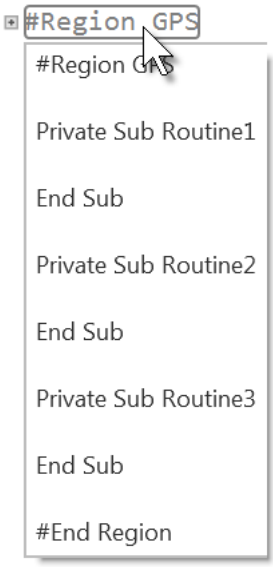
`#Region GPS` sets the beginning of a region and `#End Region` the end

Then you can add the subroutines between the two limits:



Then clicking on ⊟

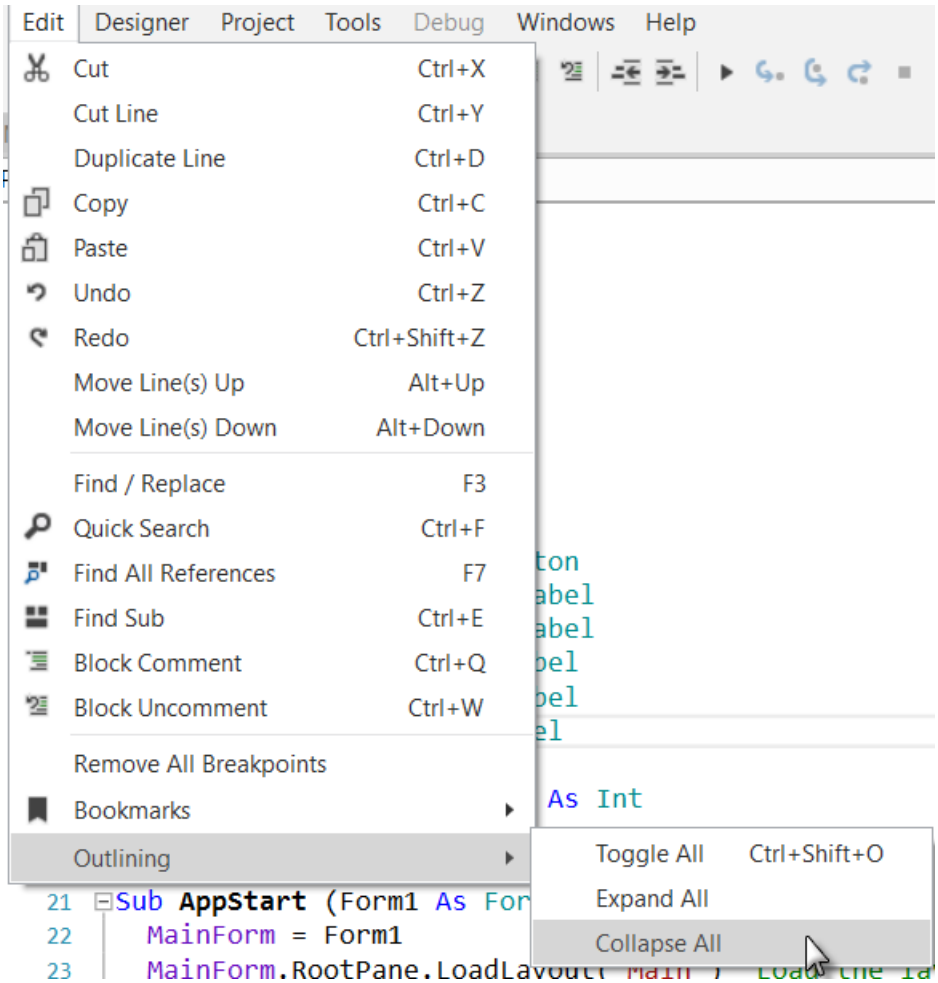
collapses the whole region.



Hovering over GPS

shows the beginning of the code, not all the routines in the region.

4.2.5 Collapse the whole code



In the Edit menu there are three functions:

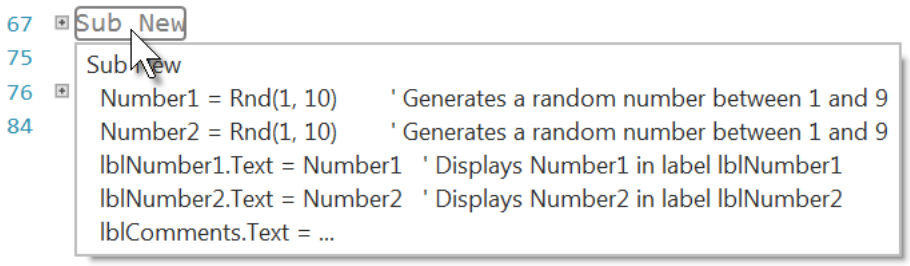
- **Toggle All**
Expands the collapsed routines and collapses the extended routines and regions.

- **Expands All**
Expands the whole code

- **Collapse All**
Collapses the whole code.
Clicking on Collapse All.

The whole code collapsed.

```
1   'Code module
2   #Region  Project Attributes
9
10  Sub Process_Globals
26
27  Private Sub Application_Start (Nav As NavigationController)
37
38  Private Sub Page1_Resize(Width As Int, Height As Int)
41
42  Private Sub Application_Active
45
46  Private Sub Application_Inactive
49
50  Private Sub Application_Background
53
54  Sub btnAction_Click
66
67  Sub New
75
76  Sub CheckResult
84
```



Hovering with the mouse over a subroutine shows the beginning of its content.

4.2.6 Copy a selected bloc of text

It is possible to copy a selected bloc of text to the clipboard.

To select the bloc press Alt and move the mouse cursor.

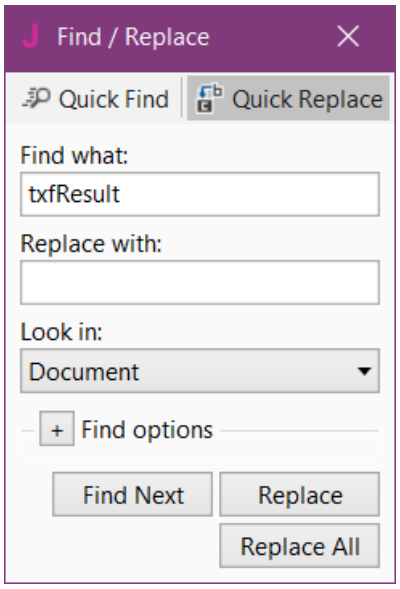
```
15    Private btnAction As Button
16    Private lblMathSign As Label
17    Private lblComments As Label
18    Private lblNumber1 As Label
19    Private lblNumber2 As Label
20    Private txfResult As TextField
```

4.2.7 Find / Replace

```
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
```

Example:
Click on `txfResult` to select it.

Press F3, or click on Find / Replace in the Edit menu.



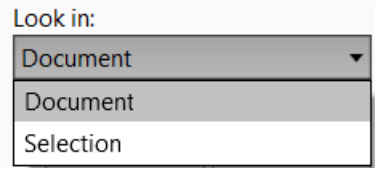
This window will be displayed.

Enter `lblResult` in the 'Replace with' field.

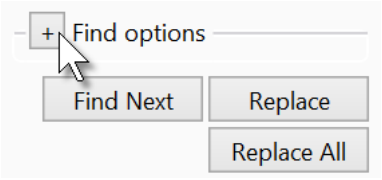
Now, you can either:

- Find Next — Find the next occurrence.
- Replace — Replace the current occurrence and find the next one.
- Replace All — Replace all occurrences.

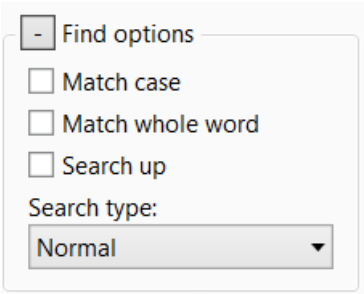
Look in:



You can search either in a Selection or in the Document, which means in the selected module not the whole project.



You can select Find options, click on +.



These options are self-explanatory.

4.2.8 Commenting and uncommenting code

A selected part of the code can be set to comment lines or set to normal.

```
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
```

Original code

```
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
```

Select the code.

Click on  .

```
17  ' Private btnAction, btn0 As Button
18  ' Private lblMathSign As Label
19  ' Private lblComments As Label
20  ' Private lblNumber1 As Label
21  ' Private lblNumber2 As Label
22  ' Private lblResult As Label
```

All lines are set as comments.

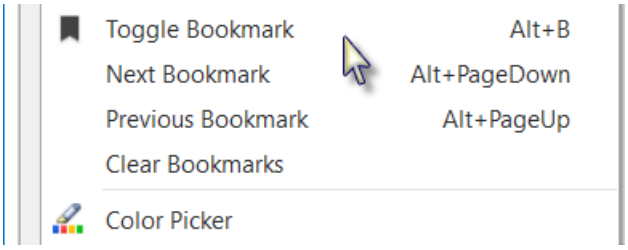
To set the lines to normal, select the lines and click on  .

4.2.9 Bookmarks

You can set 'bookmarks' anywhere in the code and jump forward and backwards between these bookmarks.

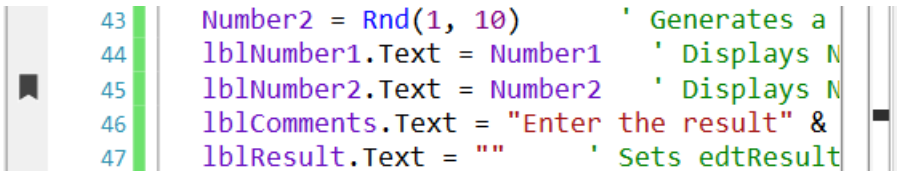
To set or clear a bookmark, select the line and press Alt + B.

Or right click on the line where you want to set a bookmark.



You will get a pop up menu, click on
Toggle Bookmark
to activate or deactivate a bookmark.

You will see this mark on the left of the line and a small black line in the right slider:



To jump to the next bookmark press Alt + PageDown
or right click and click on Next Bookmark Alt+PageDown

To jump to the previous bookmark press on Alt + PageUp
or right click and click on Previous Bookmark Alt+PageUp

To clear all bookmarks right click and click on Clear Bookmarks

4.2.10 Indentation

A good practice is to use the indentation of code parts.
For example, for subroutines, loops, structures etc.

```
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
```

This code is difficult to read because the structure of the code is not obvious.

```
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
```

This code is much easier to read, the structure of the code is in evidence.

A tabulation value of 2 for the indentation is a good value.

```
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
```

Example with an indentation of 4

Personally,
I prefer a value of 2.

Whole blocks of code can be indented forth and back at once.

```
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
```

Original code

```
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
```

Select the code block.

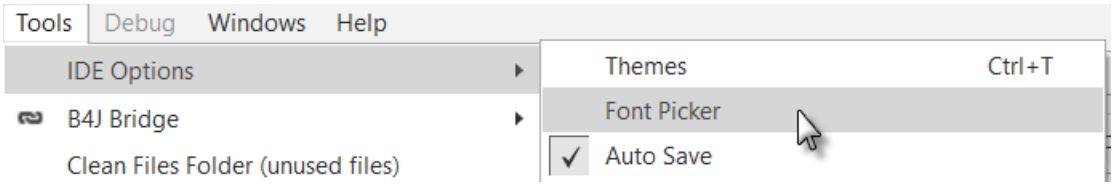
Click on  .

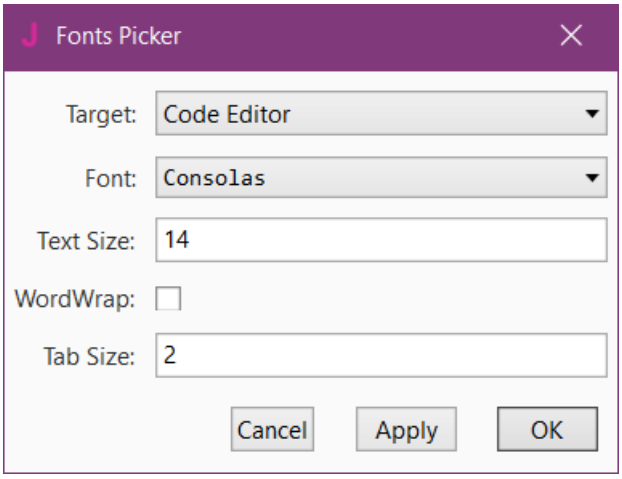
```
17        Private btnAction As Button
18        Private lblMathSign As Label
19        Private lblComments As Label
20        Private lblNumber1 As Label
21        Private lblNumber2 As Label
22        Private txfResult As TextField
```

The whole block has moved one tabulation to the right.

To move a block to the left.
Select the code and click on  .

The indentation value can be changed in the Tools menu in the IDE Options Font Picker.





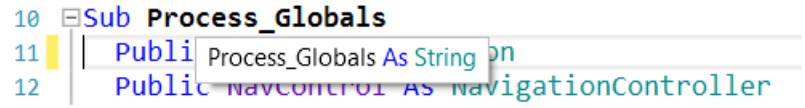
Enter the value and click on OK.

4.2.11 Documentation tool tips when hovering over code elements

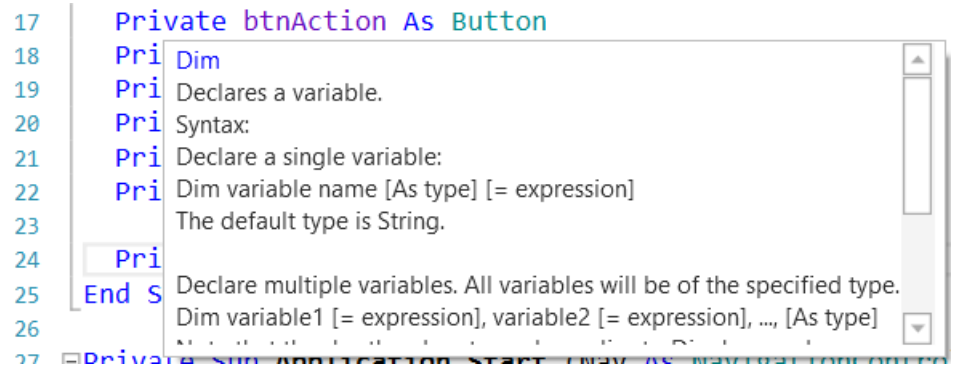
When you hover over code elements the on-line help is displayed.

Examples:

Hovering over Globals:



Hovering over Private:



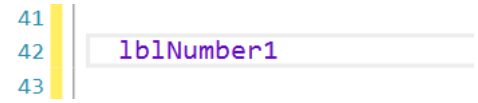
4.2.12 Auto Completion

A very useful tool is the autocomplete function.

Example:

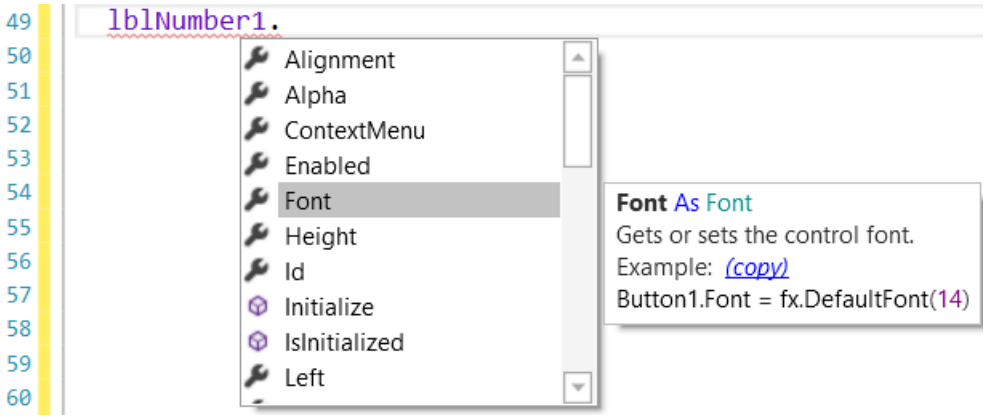
Let us write lblN
All variables, nodes and property names containing the letters already written are shown in a popup menu with the online help for the highlighted variable, node or property name.

To choose lblNumber1 press Return.



The selected name is completed.

To choose lblNumber2 press the down arrow and press Return.



When selecting an item, the internal help is displayed.

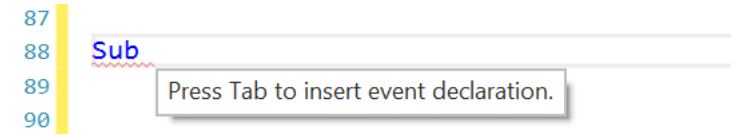
Pressing on the up / down arrows selects the previous or next item with its help.

Pressing a character updates the list and shows the parameter beginning with that character.

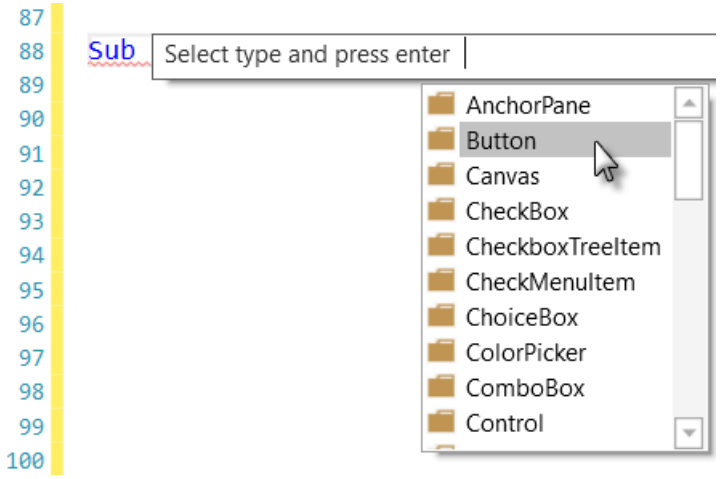
The best way to learn it is to 'play' with it.

A second Autocomplete function allows you to create event subroutines.

Enter the Sub word plus a blank character.



Press Tab.



Select the type, Button in our example.

Select the type, MouseClicked in our example.

The subroutine frame is generated.

```
Sub EventName_MouseClicked (EventData As MouseEvent)

End Sub
```

Modify 'EventName' to the EventName of the button, 'btnOK' in our case and press Return.

```
Sub btnOk_MouseClicked (EventData As MouseEvent)

End Sub
```

The routine is ready.

```
Sub btnOk_MouseClicked (EventData As MouseEvent)

End Sub
```

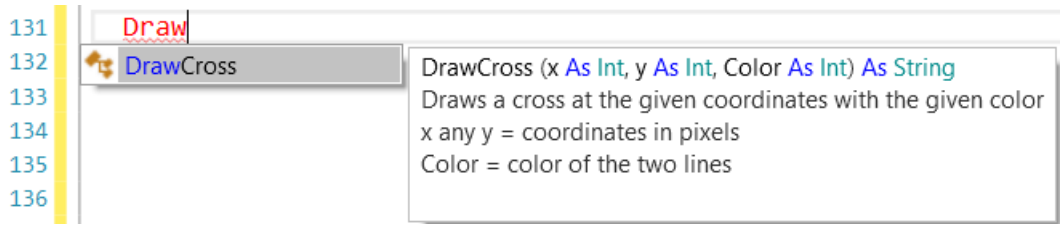
4.2.13 Built in documentation

Another useful function is the built-in documentation.

Comments above subs, such as:

```
'Draws a cross at the given coordinates with the given color
'x any y = coordinates in pixels
'Color = color of the two lines
Sub DrawCross(x As Int, y As Int, Color As Int)
	Private d = 3dip As Int

	cvsLayer(2).DrawLine(x - d, y, x + d, y, Color, 1)
	cvsLayer(2).DrawLine(x, y - d, x, y + d, Color, 1)
End Sub
```

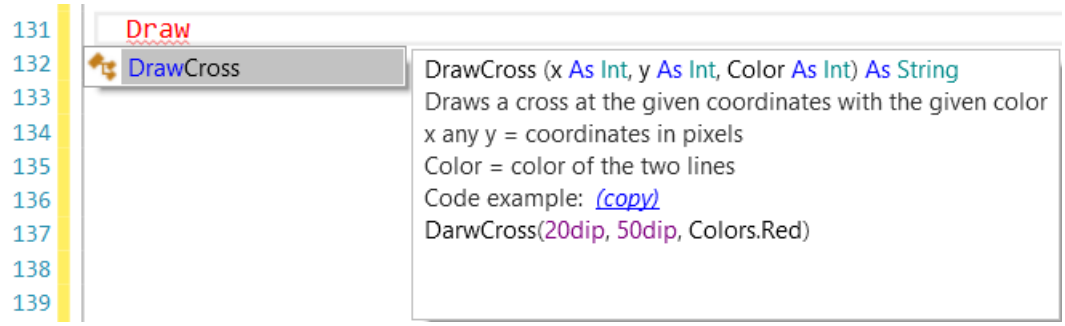
Will automatically appear in the auto complete pop-up window:



If you want to add a code example you can use <code> </code> tags:

```
'Draws a cross at the given coordinates with the given color
'x any y = coordinates in pixels
'Color = color of the two lines
'Code example: <code>
'DarwCross(20dip, 50dip, Colors.Red)
'</code>
Sub DrawCross(x As Int, y As Int, Color As Int)
   Private d = 3dip As Int

   cvsLayer(2).DrawLine(x - d, y, x + d, y, Color, 1)
   cvsLayer(2).DrawLine(x, y - d, x, y + d, Color, 1)
End Sub
```

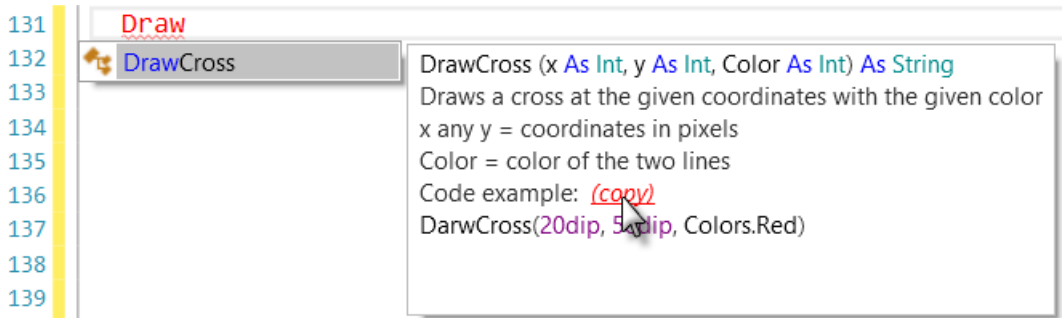
The code will be syntax highlighted:



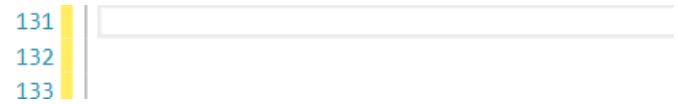
4.2.13.1 Copy code examples

You can copy the code example in your code.

When hovering over (copy) you can copy the code example to the clipboard.



Remove `Draw`



And copy.

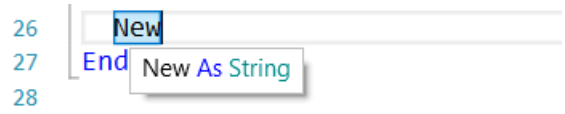
```
131    DrawCross(20dip, 50dip, Colors.Red)
132
133
```

4.2.14 Jump to an identifier

Sometimes it is useful to jump from a subroutine call to the subroutine definition or to go from a node or a variable to its definition.

The easiest way is to press Ctrl and click on the desired text.

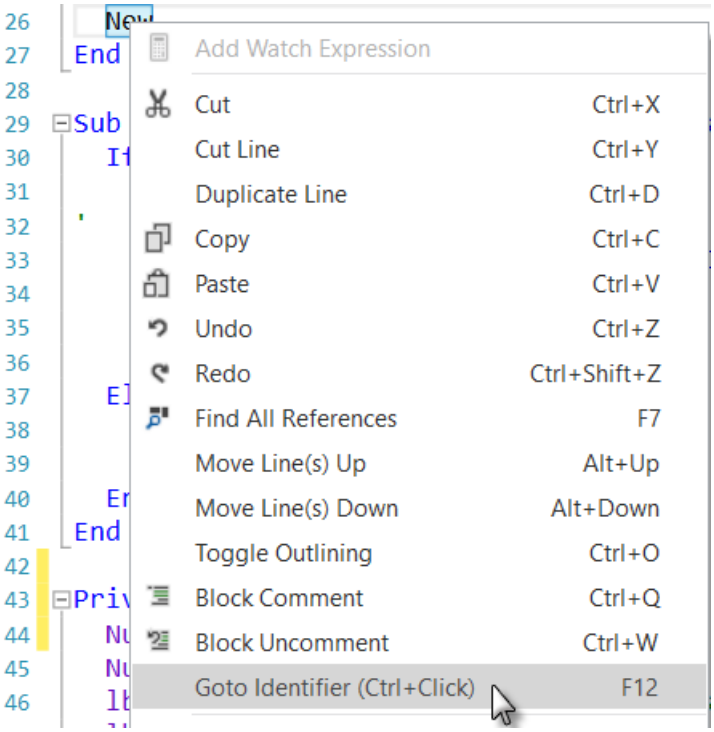
The cursor changes when you hover over the text.



Another way:
Double click on the text of the subroutine call, the variable or the node to select it.

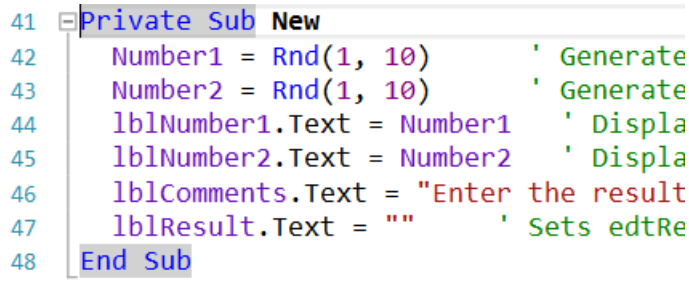
It is highlighted in dark blue and all the other occurrences are highlighted in light blue.

Press F12, or like below.



Right click on the selected text.

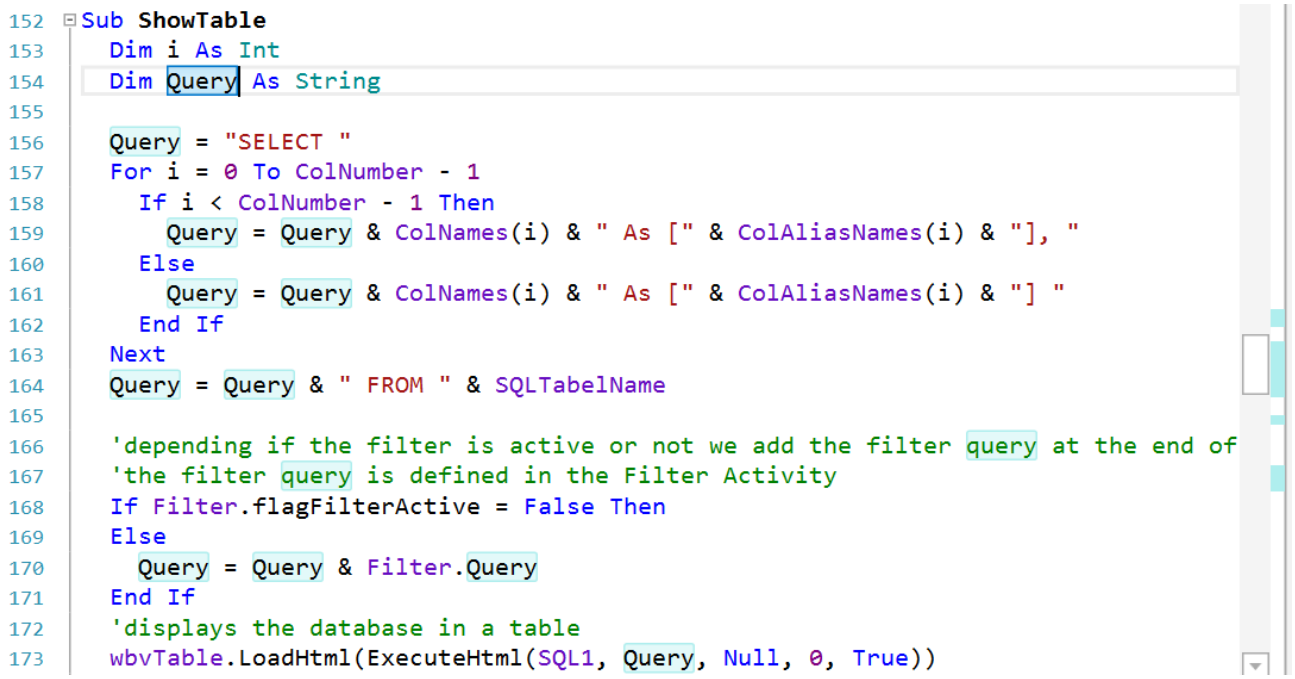
Click on Goto Identifier (Ctrl+Click).



And you are there.

4.2.15 Highlighting occurrences of words

When you select a single word, it is highlighted in dark blue and all the other occurrences in the code are highlighted in light blue and in the scroll node on the right side.
With the slider, you can move up or down the code to go to the other occurrences.



```
152 Sub ShowTable
153   Dim i As Int
154   Dim Query As String
155
156   Query = "SELECT "
157   For i = 0 To ColNumber - 1
158     If i < ColNumber - 1 Then
159       Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & "], "
160     Else
161       Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & "] "
162     End If
163   Next
164   Query = Query & " FROM " & SQLTabelName
165
166   'depending if the filter is active or not we add the filter query at the end of
167   'the filter query is defined in the Filter Activity
168   If Filter.flagFilterActive = False Then
169   Else
170     Query = Query & Filter.Query
171   End If
172   'displays the database in a table
173   wbvTable.LoadHtml(ExecuteHtml(SQL1, Query, Null, 0, True))
```

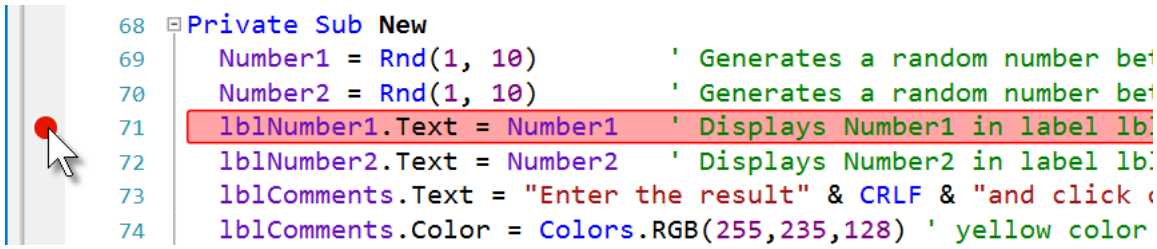
4.2.16 Debug

The Debug mode is activated by default on top of the IDE.

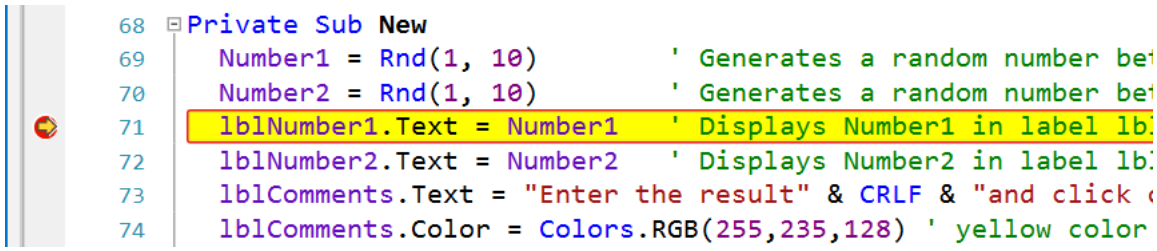
Look at chapter Debugging for debug features.

4.2.17 Breakpoints

Clicking on the left side in a line sets a breakpoint.



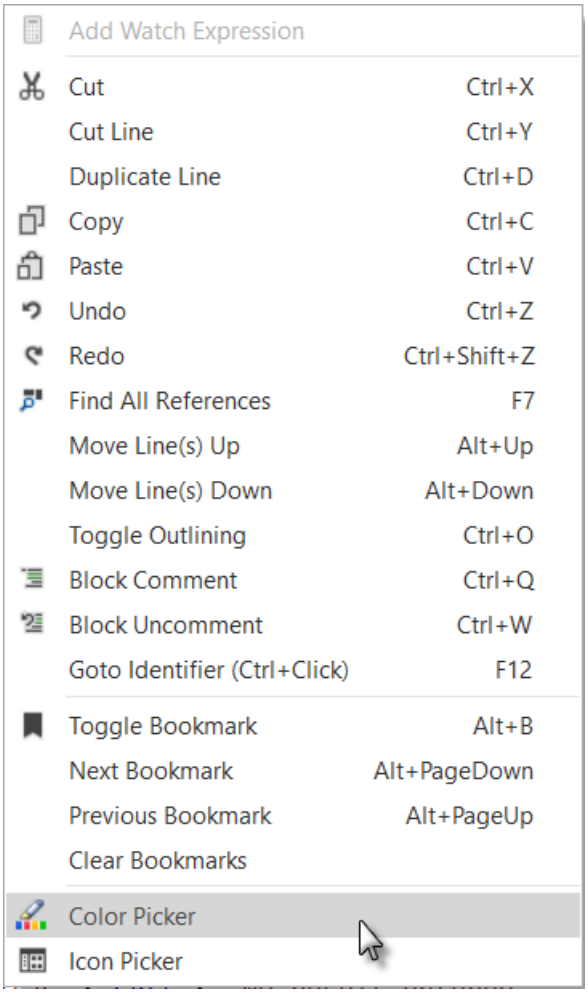
When the program runs it stops at the first encountered breakpoint.



You can remove all breakpoints in the Edit menu with Remove All Breakpoints.

The use of breakpoints is explained in detail in the Debugging chapter.

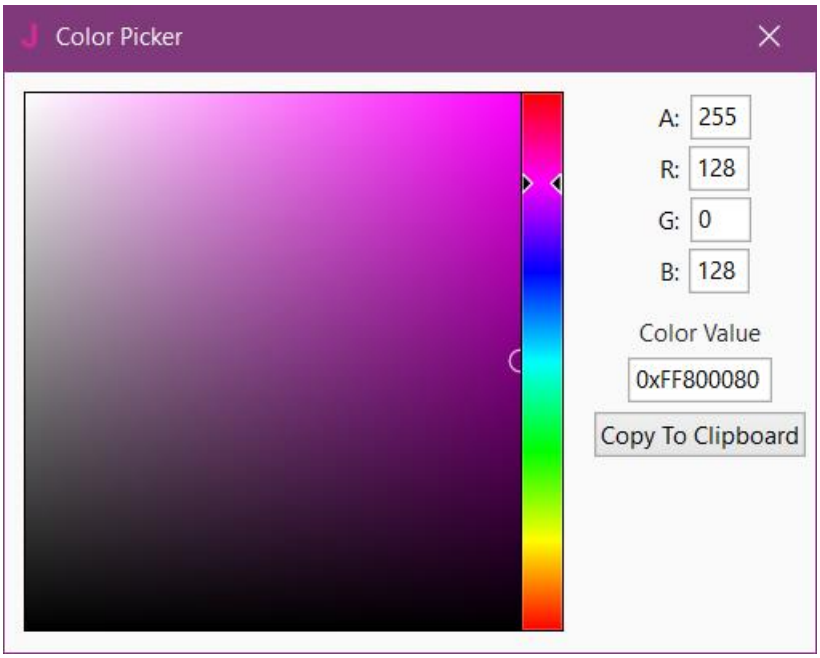
4.2.18 Color Picker



In the Tools menu click on Color Picker.

Or, in the code, right click to show the pop up menu below and click on menu Color Picker.

The Color Picker will be displayed.

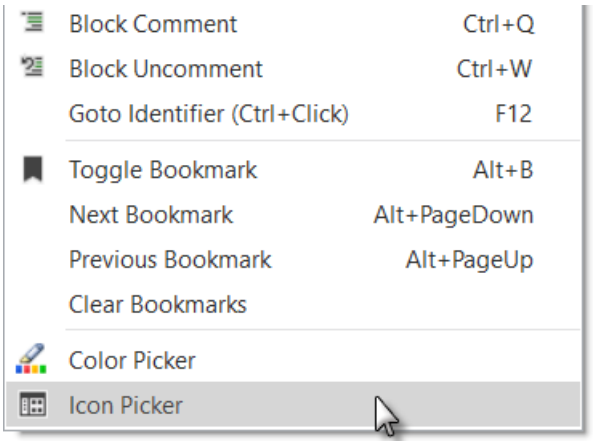


To select a color you can:
-Move the small circle.
-Move the vertical cursor.
-Enter the ARGB values.

Click on Copy To Clipboard to copy the value to the Clipboard.

You may then paste the value into your code.

4.2.19 Icon Picker



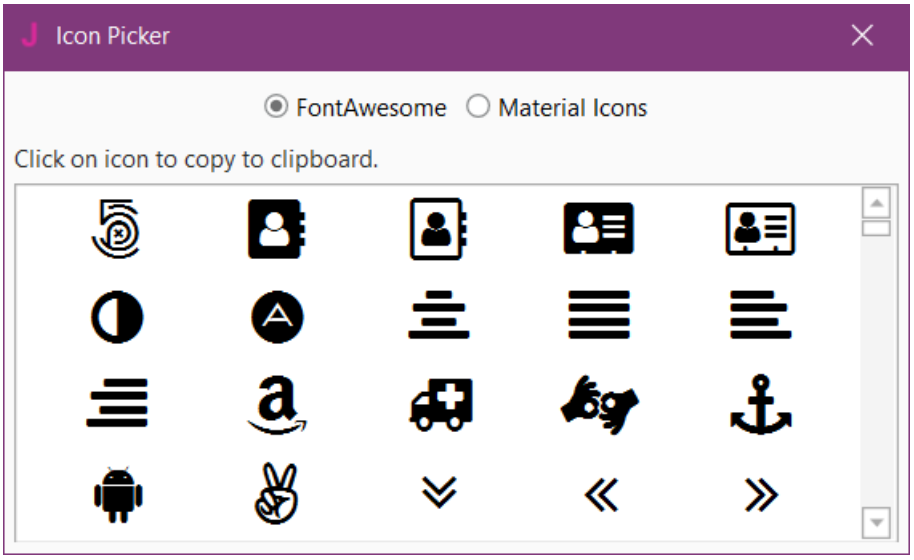
In the code, right click to show the pop up menu below and click on menu 

.

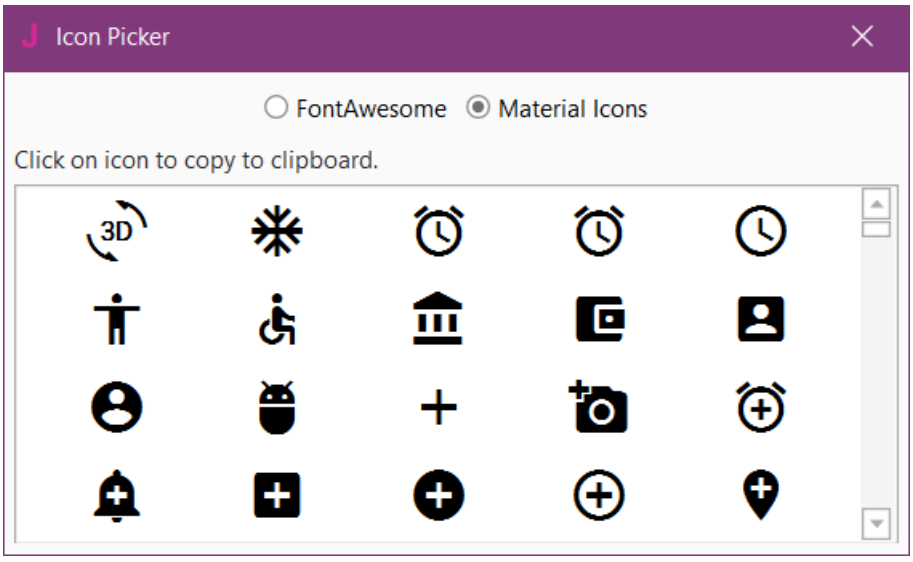
The Color Picker will be displayed.

The IconPicker will be displayed. Two types of icons are available.

Click on an icon to copy it to the Clipboard.



FontAwesome icons.



Material Icons.

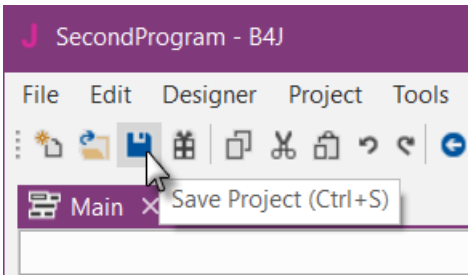
4.2.20 Colors in the left side

Sometimes, you will see yellow or green vertical lines in the left side of the IDE.

As soon as you modify a line it will be marked with a yellow vertical line on the right of the line number meaning that this line was modified.

```
51 Private Sub CheckResult
52   If lblResult.Text = Nu
53     lblComments.Text = "
54     btnAction.Text = "N
55   Else
56     lblComments.Text = "
57   End If
58 End Sub
```

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




```
51 Private Sub CheckResult
52   If lblResult.Text = Nu
53     lblComments.Text = '
54     btnAction.Text = "N
55   Else
56     lblComments.Text = '
57   End If
58 End Sub
```

If we leave the IDE and load the project again the green lines disappear.

4.2.21 URLs in comments and strings are ctrl-clickable

URLs in comments and strings are ctrl-clickable.

In a comment:

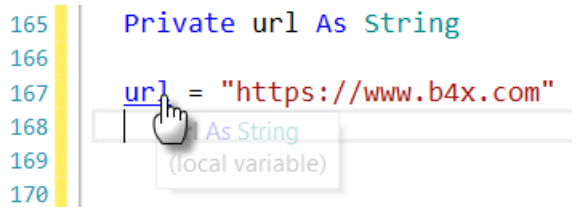
```
162  'https://www.b4x.com
```

If the cursor is on the line and you press Ctrl the url is highlighted in blue and if you click on it the url it is executed. Hovering over the line with Ctrl pressed does also highlight the url.

In a String:

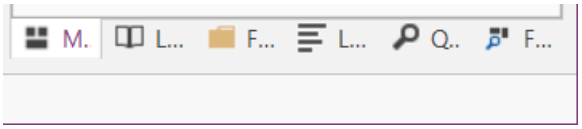
```
165  Private url As String
166
167  url = "https://www.b4x.com"
```

The cursor must be over the String variable and not over text.



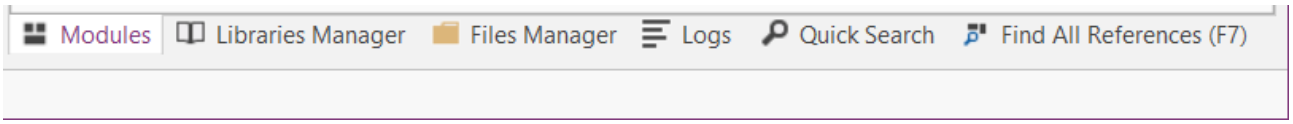
4.3 Tabs

There are 6 tabs at the bottom right corner of the IDE that display different windows.



The short version.

The wide version.

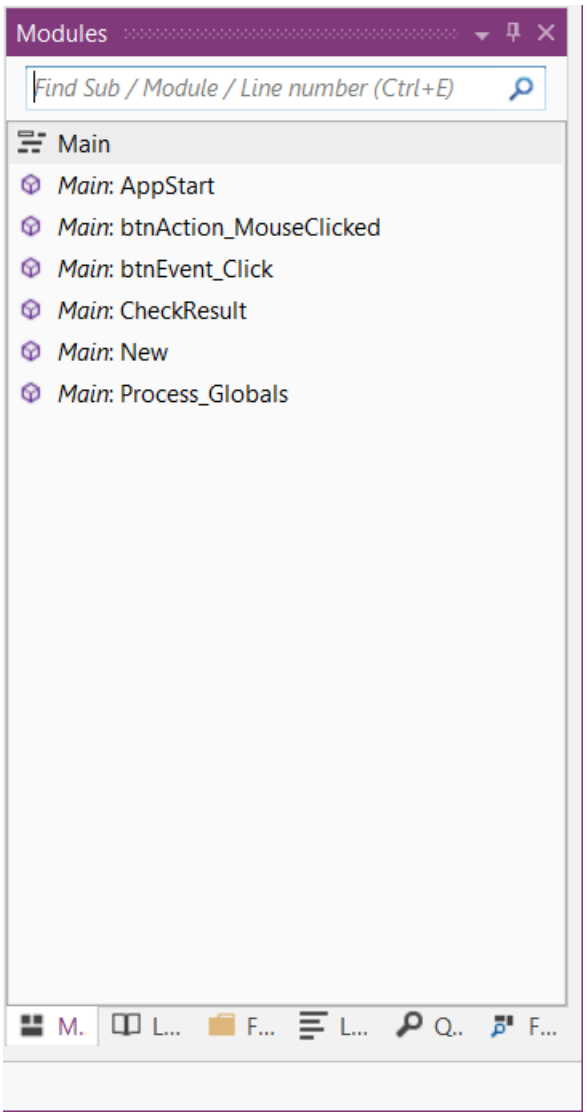


The 6 Tabs are:

- Modules
- Libraries Manager
- Files Manager
- Logs
- Quick Search
- Find All References

4.3.1 Floating Tab windows

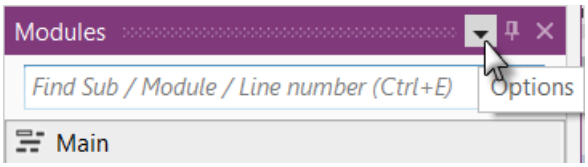
When you start the default IDE all Tab windows are docked in the Tab area.



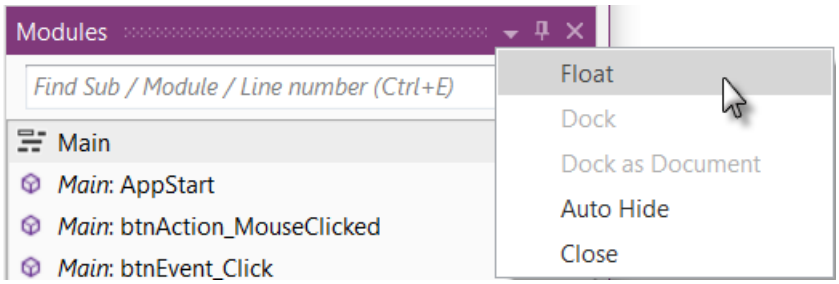
You can set each Tab window as a separate floating window.

4.3.2 Float

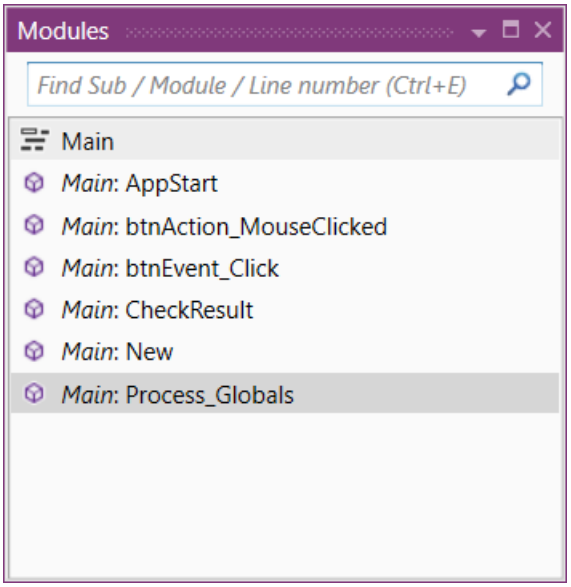
To set the Modules Tab window to floating click in the title on .



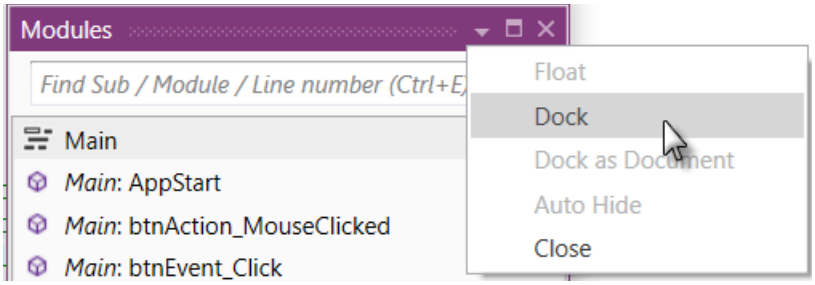
Click on Float.



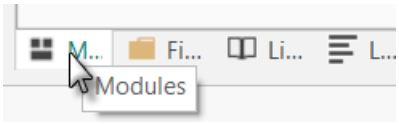
The Modules Tab Window is now floating, you can place it where you want on the screen even on a second monitor.



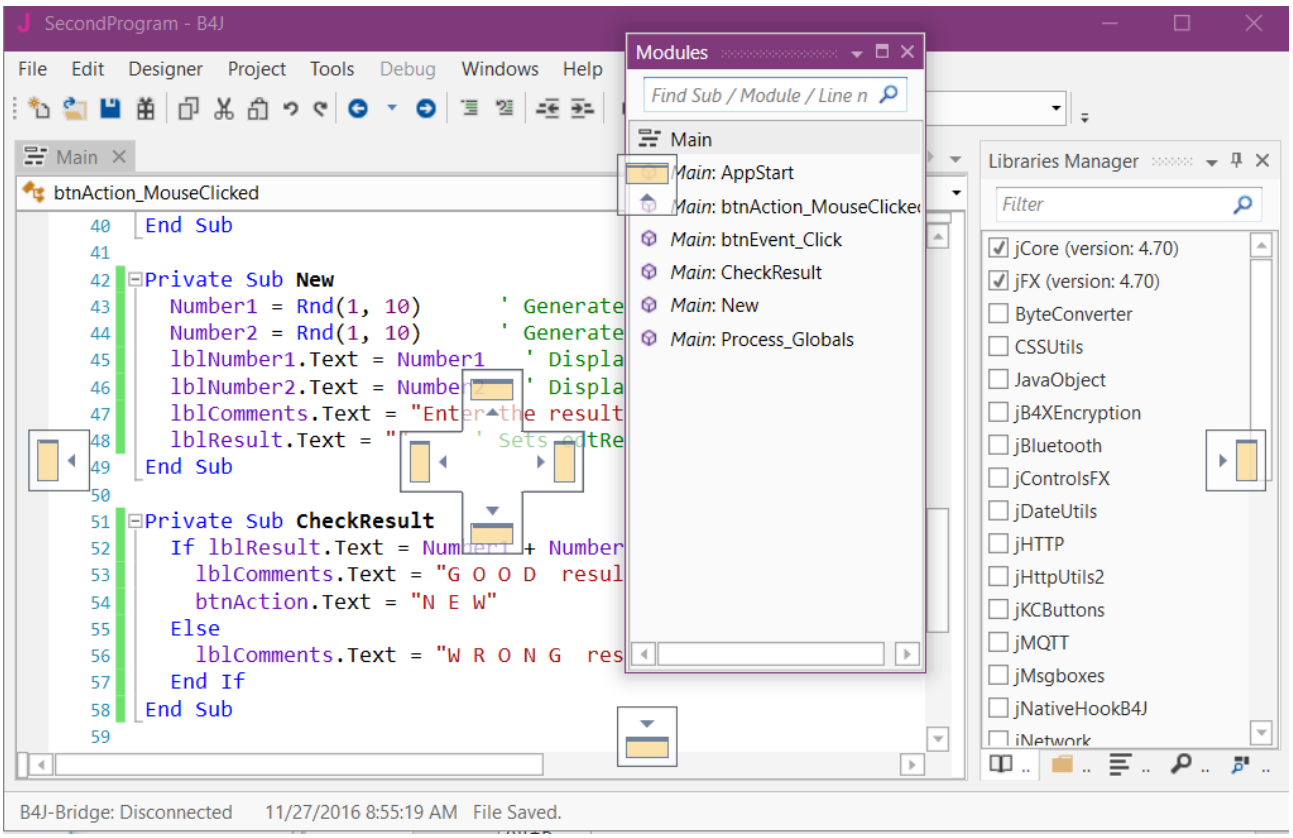
To dock it back to the Tab area click on Dock.



You can also click on a Tab and while maintaining the mouse down, move the Tab.



This will show you all the possible 'docking' areas.



Docking areas:

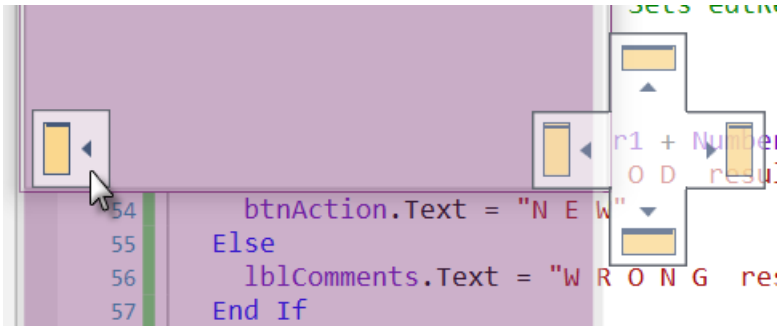

Top


Left

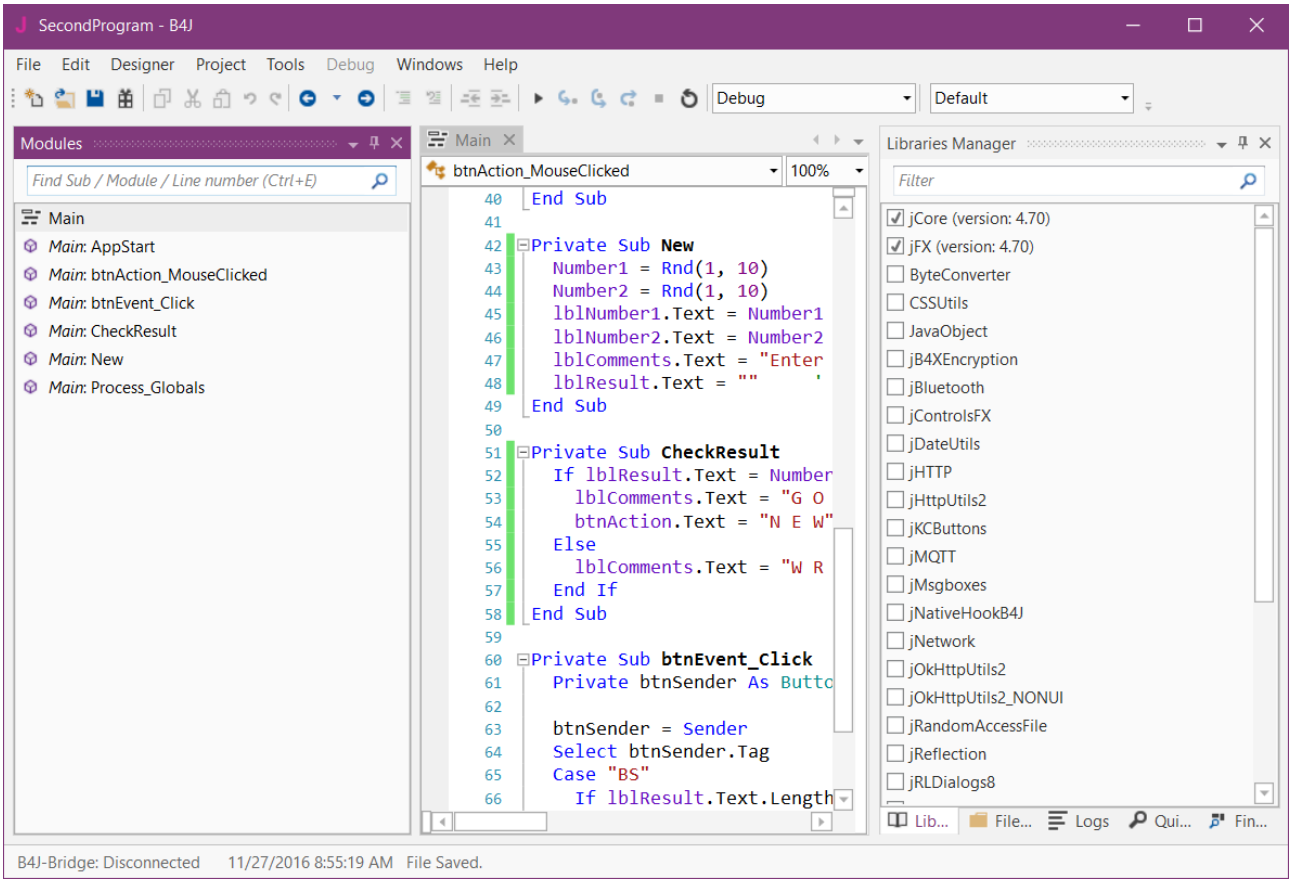

Right


Bottom

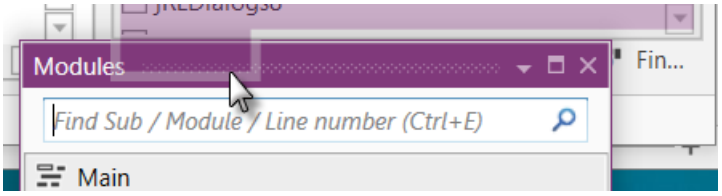
If you move the mouse onto one of the docking area symbol, the Tab window will be either on top, on the left, the right or on the bottom.



And the result.

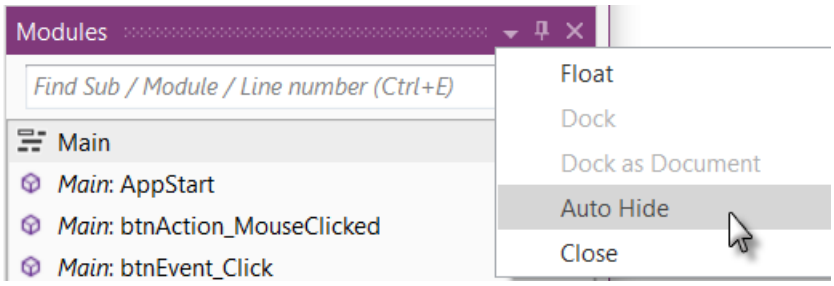


To bring it back to the Tabs, click on the window title and move it back to the Tabs.



4.3.3 Auto Hide

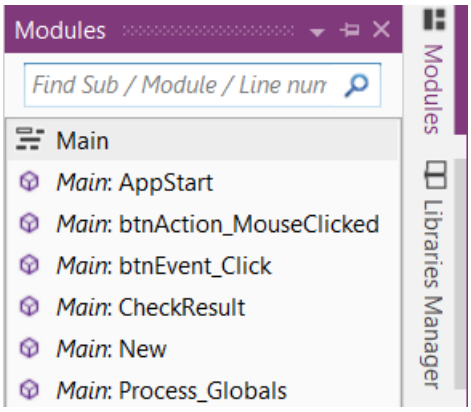
Click on in the title or click on Auto Hide in the Options.



The Tabs move from the bottom of the screen vertically to the right side of the screen and the Tab window is hidden.

Hovering over a Tab highlights it in blue.

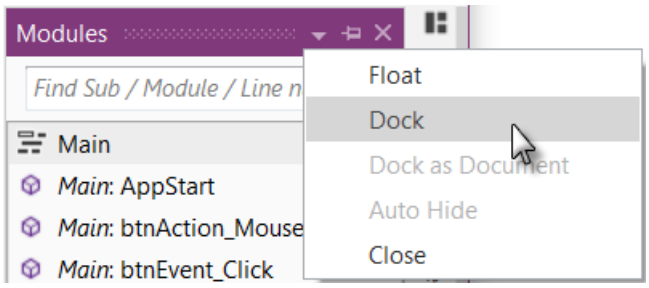
Click on a Tab to show it.



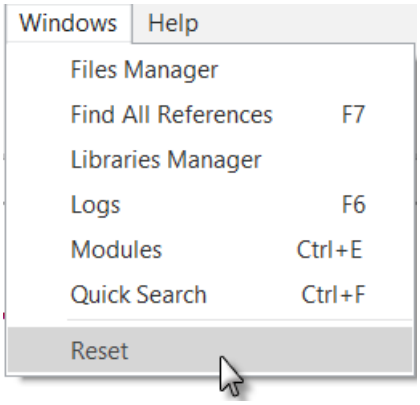
As soon as you click somewhere else in the IDE the Tab is hidden again.

To move the Tabs back to the lower right corner:

Click on Dock in the Options.



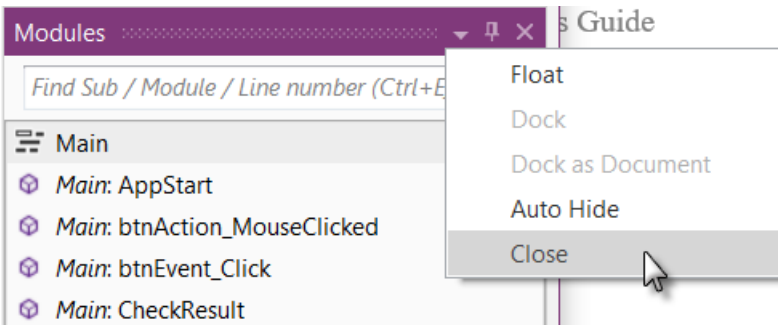
Or click on Reset in the IDE Windows menu.

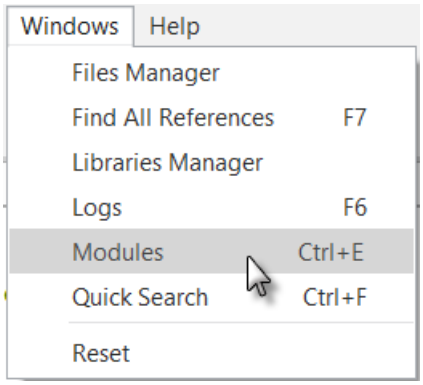


4.3.4 Close

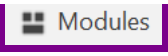
You can close a window, hide it.

Click on ✕ in the title or on Close in the Options.

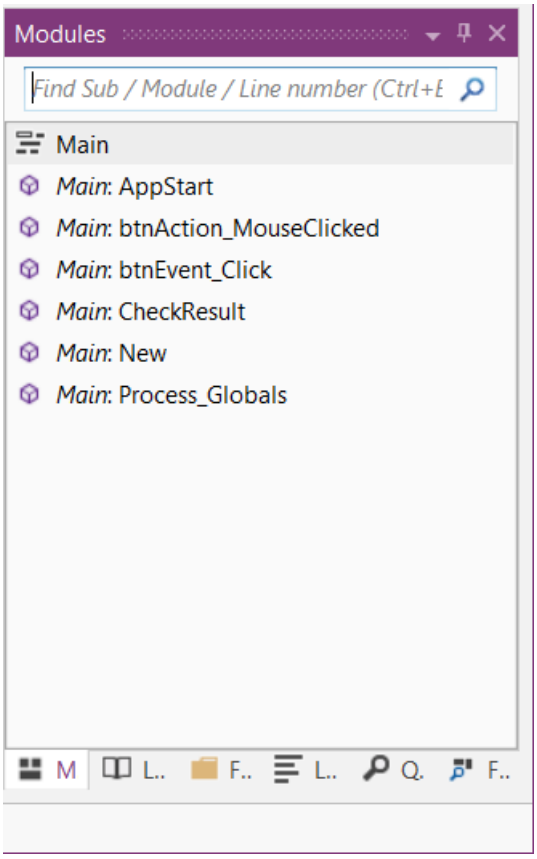


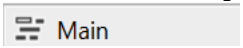


To show it again, in the Windows menu click on the module name you want to show, Modules in our example.

4.3.5 Modules and subroutines list 

All the modules of the project and all subroutines of the selected module are listed in the Modules window. The picture below has been reduced in height.



Module list on top. Only one module in the example.


Clicking on a module shows its code in the code area.

Subroutine list of the selected module.
Clicking on a subroutine shows its code in the middle of the code area.

Note the different icons:  module and  routine.

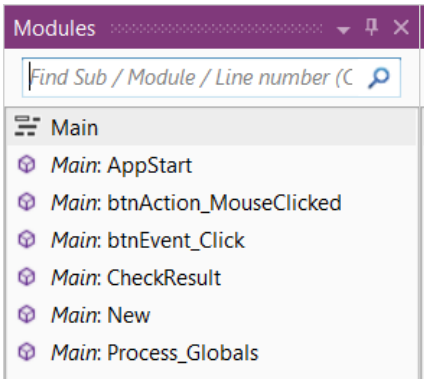
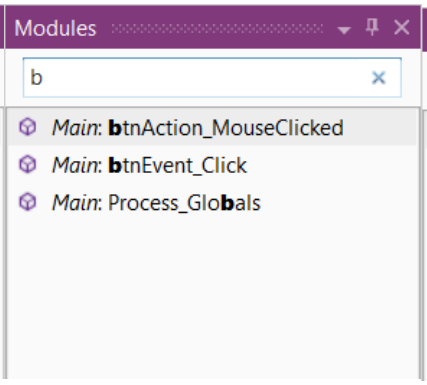
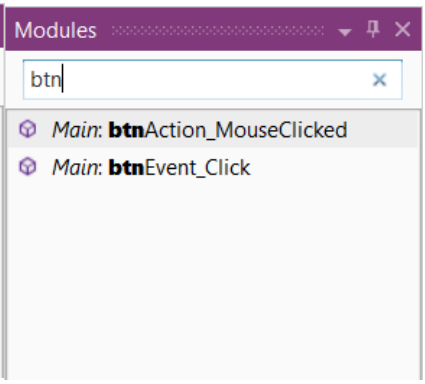
To show a hidden module, click on the module name in the module list.

4.3.5.1 Find Sub Tool (Ctrl + E)

The *Find Sub / Module* function is a search engine, on the Top of the Modules Tab, to find subroutines or Modules with a given name or with a given part of the name.

You can press Ctrl + E in the code to select the Modules Tab with the *Find Sub / Module* function.

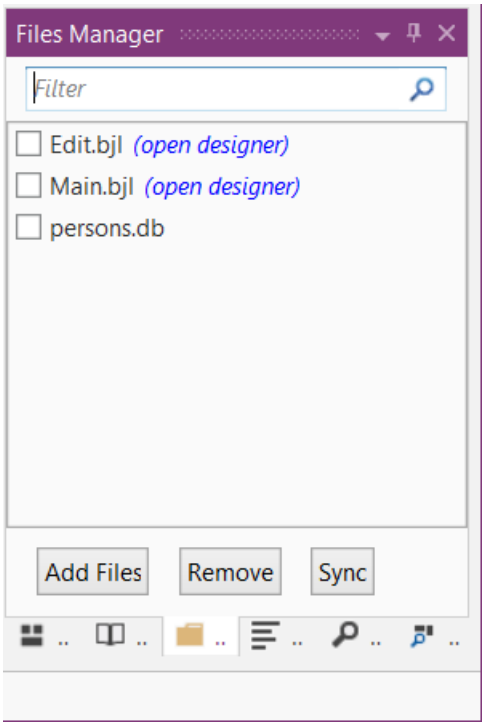
Example with the code of the SecondProgram example.

No text	only the character 'b'	text 'btn'
		
Shows all modules and all routines of the selected Module.	Shows all modules and routines containing 'b'.	Shows all modules and routines containing 'btn'.

Clicking on one item shows the code of the selected module or routine, even if it's in another module than the current one.

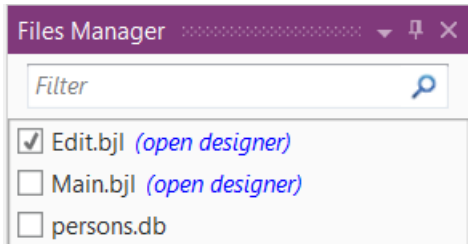
4.3.6 Files Manager

This window lists all the files that have been added to the project.
These files are saved in the 'Files' subfolder under your main project folder.
The files can be of any kind: layouts, images, texts, etc.



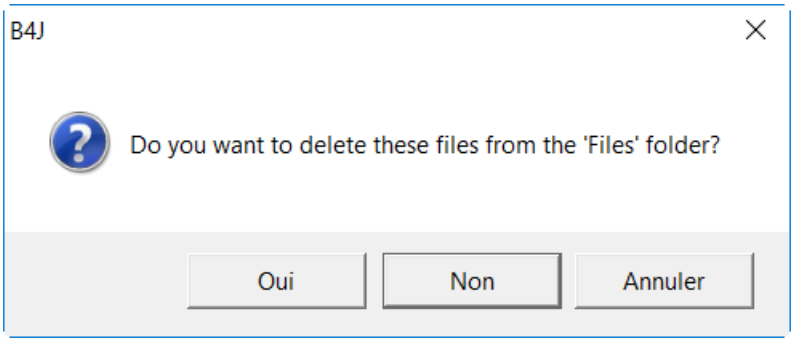
Click on Add Files to add files to the list.
The files in that subfolder can be accessed from your program by using the reference File.DirAssets.

Or click on Sync to add all the files from the projects Files folder into the File Tab.



Checking one or more files enables the Remove button.

Clicking on this button removes the selected files from the list and, if you want, from the Files folder of the project.

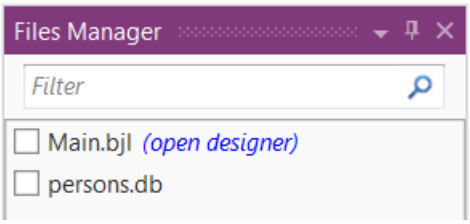


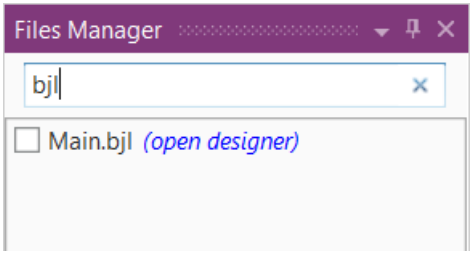
You are asked if you want to delete the files from the 'Files' folder.
Oui = Yes
Non = No
Annuler = Cancel

When you answer Yes (Oui) make sure to have a copy of the files you remove, because they are removed from the Files folder, but not transferred to the Recycle Bin, which means that they are definitely lost if you don't make a copy.

See chapter Files for file handling.

On top of the Files Manager window you can filter the files list.





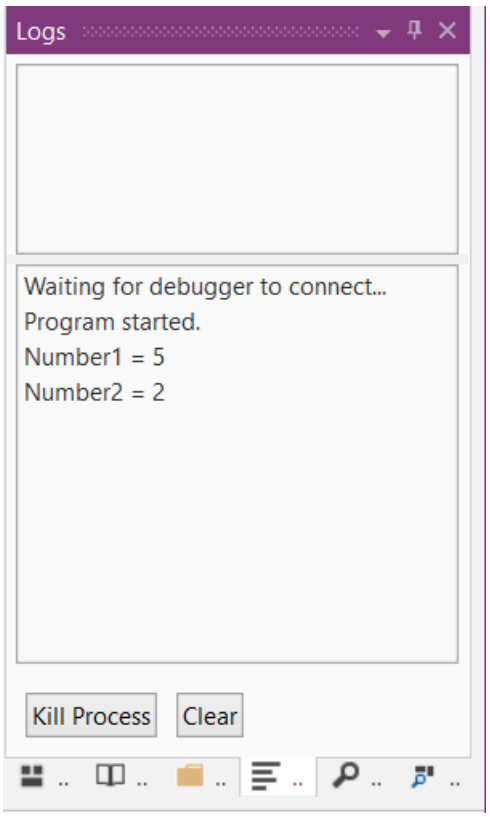
Enter '.bjl' to filter all layout files.

4.3.7 Logs

Display of Log comments generated by the program when it is running.

We add the two lines 51 and 53 in the program 'SecondProgram' in the 'New' routine.
The number of the lines may be different from yours.

```
42 Private Sub New
43     Number1 = Rnd(1, 10)       '
44     Log("Number1 = " & Number1)
45     Number2 = Rnd(1, 10)       '
46     Log("Number2 = " & Number2)
47     lblNumber1.Text = Number1
48     lblNumber2.Text = Number2
49     lblComments.Text = "Enter th
50     CSSUtils.SetBackgroundColor(
51     lblResult.Text = ""        ' Se
52     btn0.Visible = False
53 End Sub
```



Run the program.

In the Logs window we see the flow of the program.

The top area of the window shows Compile Warnings see next page.

In the lower area of the window we see the flow of the program.

Waiting for debugger to connect…
Program started
Number1 = 5 First log message
Number2 = 2 Second log message

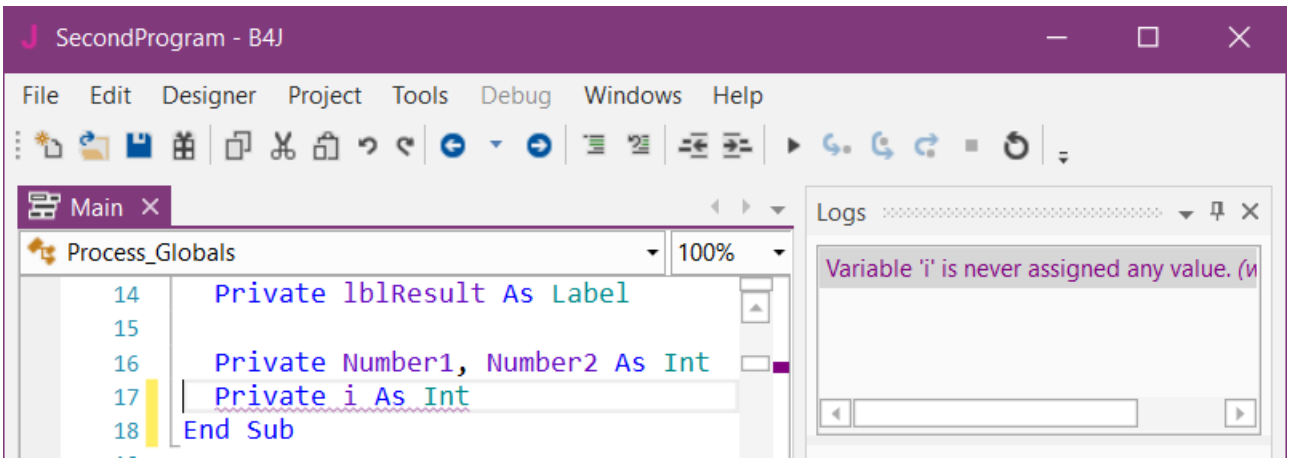
Click Clear to clear the Logs window.

4.3.7.1 Compile Warnings

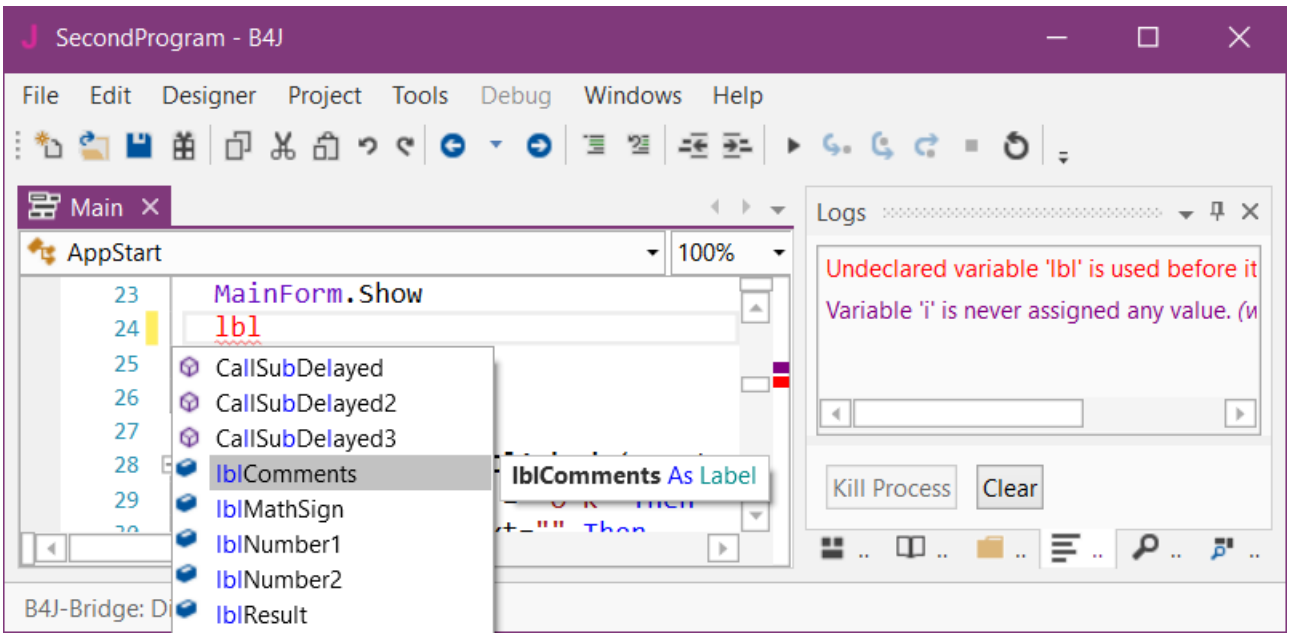
B4J includes a warning engine. The purpose of the warning engine is to find potential programming mistakes as soon as possible. The examples are from the Warnings project.

The compile-time warnings appear above the logs and in the code itself when hovering with the cursor above the code line.
The code lines which cause a warning are underlined like this `Private i As Int`.

Clicking on the warning in the list will take you to the relevant code.



The warning engine runs as soon as you type.

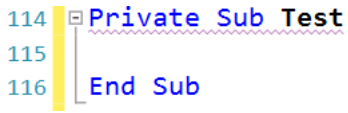


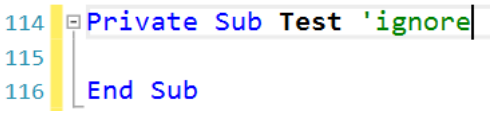
Typing for example ‘lbl’ at the beginning of a line shows immediately:

- lbl in red, because lbl was not yet declared.
- a warning Undeclared variable 'lbl' is used before it was assigned any value.
- the auto complete pop up window with suggestion containing the written characters.

4.3.7.1.1 Ignoring warnings

You, as the developer, can choose to ignore any warning. Adding an "ignore" comment will disable all the warnings for that specific line:





You can also disable warnings from a specific type in the module by adding the #IgnoreWarning attribute in the Project Attributes regions.

For example, to disable warnings #10 and #12:

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

   #IgnoreWarnings: 10, 12
#End Region
```

You find the warning numbers at the end of each warning line.

4.3.7.1.2 List of warnings

List of warnings
1: Unreachable code detected.
2: Not all code paths return a value.
3: Return type (in Sub signature) should be set explicitly.
4: Return value is missing. Default value will be used instead.
5: Variable declaration type is missing. String type will be used.
6:
7: Object converted to String. This is probably a programming mistake.
8: Undeclared variable '{1}'.
9: Unused variable '{1}'.
10: Variable '{1}' is never assigned any value.
11: Variable '{1}' was not initialized.
12: Sub '{1}' is not used.
13: Variable '{1}' should be declared in Sub Process_Globals. ???
14: File '{1}' in Files folder was not added to the Files tab.\n You should either delete it or add it to the project.\n You can choose Tools - Clean unused files.
15: File '{1}' is not used.
16: Layout file '{1}' is not used. Are you missing a call to MainForm.RootPane.LoadLayout?
17: File '{1}' is missing from the Files tab.
18: TextSize value should not be scaled as it is scaled internally.
19: Empty Catch block. You should at least add Log(LastException.Message).
20: View '{1}' was added with the designer. You should not initialize it.
21: Cannot access view's dimension before it is added to its parent.
22: Types do not match.
23:
24: Accessing fields from other modules in Sub Process_Globals can be dangerous as the initialization order is not deterministic.
25: Sub '{1}' not found.
26:
27:
28:
29: This sub should only be used for variables declaration or assignments of primitive values.
30: Variable name is the same as a module name. This can cause problems during debugging.
32: Library ‘xxxx’ is not used.

1: Unreachable code detected.

There is some code which will never be executed.
This can happen if you have some code in a Sub after a Return statement.

2: Not all code paths return a value.

```
Sub Calc(Val1 As Double, Val2 As Double, Operation As String) As Double
   Select Operation
   Case "Add"
      Return (Val1 + Val2)
   Case "Sub"
      Return (Val1 - Val2)
   Case "Mult"
      Return (Val1 * Val2)
   Case "Div"

   End Select
End Sub
```

In the `Case "Div"` path no value is returned!

3: Return type (in Sub signature) should be set explicitly.

Wrong code
```
Sub Calc(Val1 As Double, Val2 As Double, Operation As String)
```

Correct code
```
Sub Calc(Val1 As Double, Val2 As Double, Operation As String) As Double
```

The return type must be declared!

4: Return value is missing. Default value will be used instead.

Wrong code
```
Private Sub CalcSum(Val1 As Double, Val2 As Double) As Double
   Dim Sum As Double

   Sum = Val1 + Val2
   Return
End Sub
```

Correct code
```
Private Sub CalcSum(Val1 As Double, Val2 As Double) As Double
   Dim Sum As Double

   Sum = Val1 + Val2
   Return Sum
End Sub
```

5: Variable declaration type is missing. String type will be used.

Wrong code
```
Private Sub Calc(Val1, Val2 As Double, Operation As String) As Double
```

Correct code
```
Private Sub Calc(Val1 As Double, Val2 As Double, Operation As String) As Double
```

In sub declarations each variable needs its own type declaration.
But in Dim declarations it's allowed, in the line below both variables are Doubles:
```
Dim Val1, Val2 As Double
```

6: The following value misses screen units ('dip' or %x / %y): {1}.
Not used in B4i. Is used in B4A.

7: Object converted to String. This is probably a programming mistake.

8: Undeclared variable '{1}'.

Wrong code
```
Private Sub SetHeight
   h = 10
End Sub
```

Correct code
```
Private Sub SetHeight
   Dim h As Int
   h = 10
End Sub
```

The variable `h` was not declared. You see it also with the red color.

9: Unused variable '{1}'.

```
Private Sub SetHeight
   Dim h As Int
   h = 10
End Sub
```

This warning tells that the variable `h` is not used.
It is declared and assigned a value, but it is not used !

This code gives no warning because variable `h` is used:
```
Private Sub SetHeight
   Dim h As Int
   h = 10
   lblTest.Height = h
End Sub
```

10: Variable '{1}' is never assigned any value.

```
S Private Sub Test
   Dim h As Int

End Sub
```
This warning shows that the variable `h` is declared but not assigned any value.
Correct code see above.

11: Variable '{1}' was not initialized.

Wrong code
```
Dim lst As List
lst.Add("Test1")
```

Correct code
```
Dim lst As List
lst.Initialize
lst.Add("Test1")
```

Variables (objects) like List or Map must be initialized before they can be used.
Views added by code must also be initialized before they can be added to a parent view.

12: Sub '{1}' is not used.

This warning is displayed if a Sub routine is never used.

13: Variable '{1}' should be declared in Sub Process_Globals.
Not used in B4i, no Global routine!

Certain objects like Timers and GPS must be declared in Process_Globals.

```
Private Sub Process_Globals
   Dim Timer1 As Timer
   Dim GPS1 As GPS
```

14: File '{1}' in Files folder was not added to the Files tab.

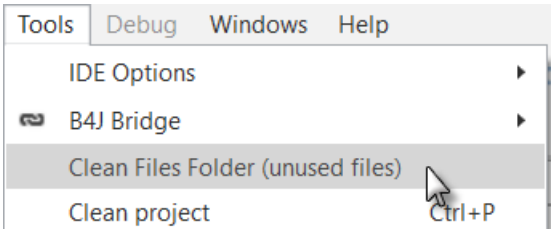
You are using a file which is in the Files folder, but was not added to the Files tab.
You should:
- Make a backup copy.
- Delete it from the Files subfolder.
- Add it to the project in the Files tab.
- Use Clean Files Folder (unused files) in the Tools menu.

15: File '{1}' is not used.

You have files in the Files folder that are not used.
You should remove them from the Files folder.
Or you can clean the Files folder from within the Tools menu (see above).

16: Layout file '{1}' is not used. Are you missing a call to Page.RootPanel.LoadLayout?

You have a layout file in the Files folder that is not used.
You should add LoadLayout or you can remove the layout file from the Files folder.



Or you can clean the Files folder in the Tools menu.

17: File '{1}' is missing from the Files tab.

The given file is in the Files tab but is missing in the Files folder. You should add it.
See chapter Files.

18: TextSize value should not be scaled as it is scaled internally.

Not used. Is used in B4A.

19: Empty Catch block. You should at least add Log(LastException.Message).

Wrong code

```
Try
   imvImage.Bitmap = LoadBitmap(File.DirAssets, "image.jpg")
Catch

End Try
```

Correct code

```
Try
   imvImage.Bitmap = LoadBitmap(File.DirAssets, "image.jpg")
Catch
   Log(LastException.Message)
End Try
```

It is recommended to add at least `Log(LastException.Message)` in the Catch block instead of leaving it empty.

20: View '{1}' was added with the designer. You should not initialize it.

A View defined with the Designer in a layout file must not be initialized!
Only views added by code need to be initialized.

21: Cannot access view's dimension before it is added to its parent.

You must add a view to a parent view before you can access its dimensions.
When you add a view by code its dimensions are defined when you add it with AddView.

22: Types do not match.

23: Modal dialogs are not allowed in Sub Activity_Pause. It will be ignored.

Not used. Is used in B4A.

24: Accessing fields from other modules in Sub Process_Globals can be dangerous as the initialization order is not deterministic.

25: Sub '{1}' not found.

26: Not used.
27: Not used.
28: Not used.

29: This sub should only be used for variables declaration or assignments of primitive values.

30: Variable name is the same as a module name. This can cause problems during debugging.

32: Library 'xxxx' is not used.

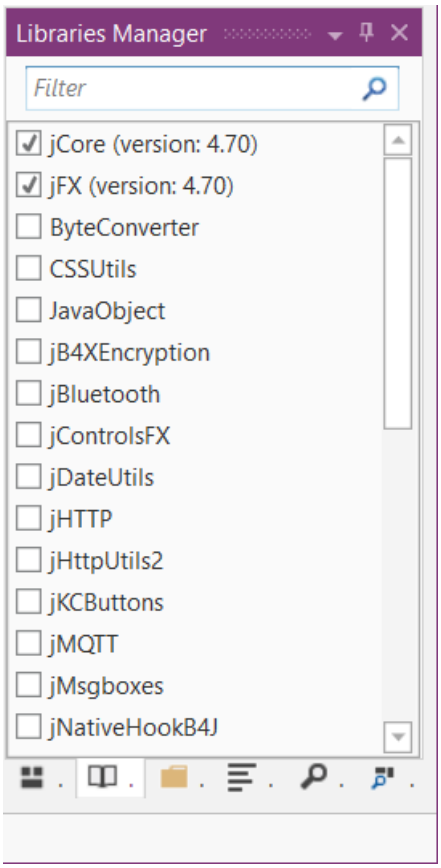
Remove the unused library.

4.3.8 Libraries Manager

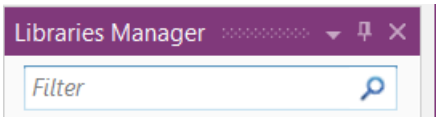
The 'Libraries Manager' tab contains a list of the available libraries that can be used in the project.

All B4J libraries have a "j" prefix, like jCore!

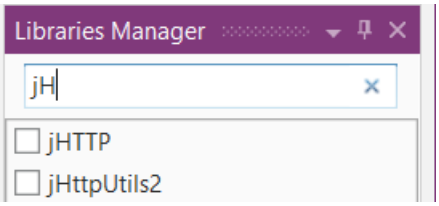
Check the libraries you need for your project.
Make sure that you have the latest version of the libraries.



In the IDE, in the bottom right corner click on Libs.



On the top of the tab you find a field to filter the libraries.



Enter 'jH' in the field and you get all libraries beginning with iH.
Note that in B4J all libraries have the prefix 'j' .

For additional libraries search in the forum.

The documentation for libraries can be found here:
B4J Documentation

Look also at chapter Libraries.

4.3.9 Quick Search

Quick Search allows to search for any text occurrences in the code of the whole project.
Examples with the SecondProgram code.

Several possibilities to select the Quick Search function:

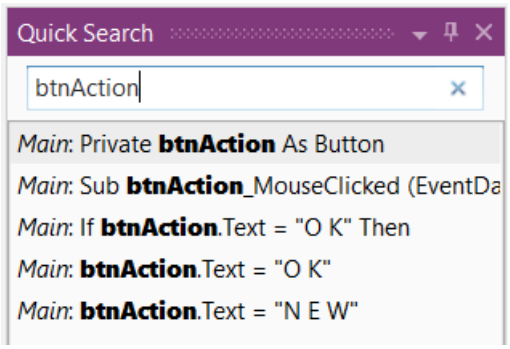
- Ctrl + F, the easiest and most efficient way.
- Click on the Quick Search Tab in the lower right corner of the IDE.
- Click on Quick Search Ctrl+F in the Edit menu.

Example:

```
Private btnAction, btn0 As Button
Private lblMathSign As Label
Private lblComments As Label
```

In the code double click on `btnAction` to select it and press Ctrl + F.

You get the window below in the Tab area.

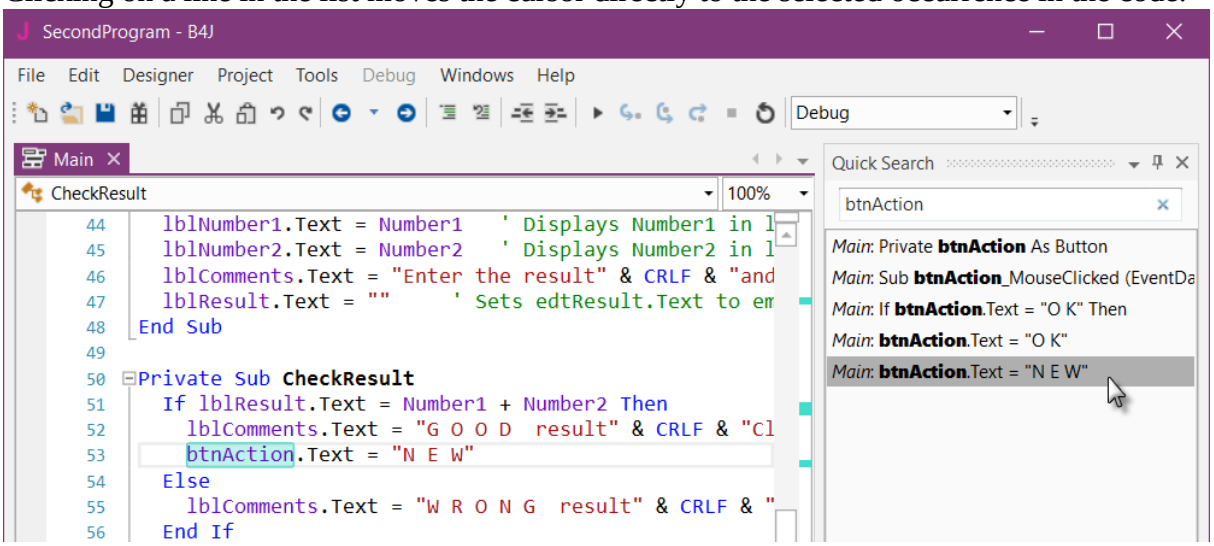


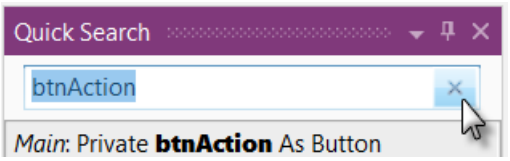
The list shows the occurrences in all Modules.

In each line you find the Module name and the line content.

Main: If **btnAction**.Text = "O K" Then Like in

Clicking on a line in the list moves the cursor directly to the selected occurrence in the code.





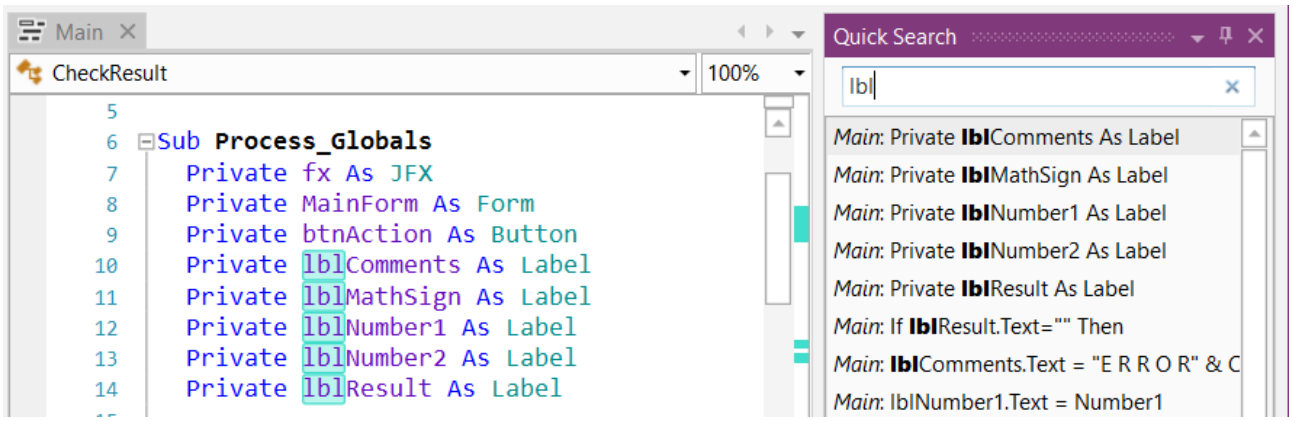
To remove the selection click on on the top right corner of the Quick Search window.

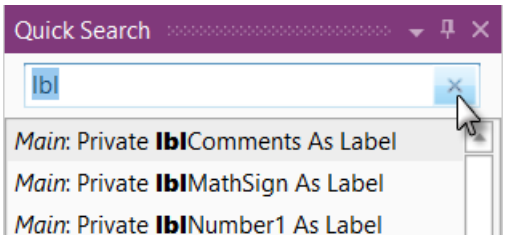
You can also enter any text in the search field:

A as an example, enter *lbl* in the Search field and you get the window below where you find all lines containing the text you entered, *lbl* in this example.

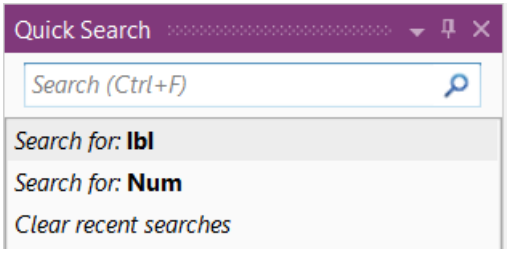
The search text is highlighted in all code lines containing this text.

Clicking on one of the lines in the list jumps directly to this line in the IDE.

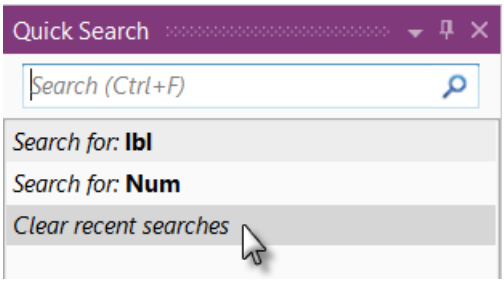




Click on  to remove a search.



You will see a list of the last searches.



Click on Clear recent searches to remove all recent searches.

Items are added to the recent items when:

1. You select one of the results or click enter which selects the first result.
2. You select text in your code and click on Ctrl + F to search for it.

4.3.10 Find All References

Find All References (F7)

This is a search engine to find all references for a given object (node, variable).

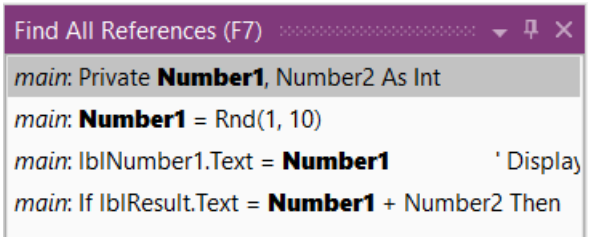
Click on the Find All References (F7) Tab or press F7 to get the screen below showing a list of all code lines with the selected reference or the first object in the current line.

Example with the code of SecondProgram.

Select in the code in line 69 `Number1`.

```
68 Private Sub New
69     Number1 = Rnd(1, 10)          ' Generates a random number
70     Number2 = Rnd(1, 10)          ' Generates a random number
71     lblNumber1.Text = Number1     ' Displays Number1 in label
72     lblNumber2.Text = Number2     ' Displays Number2 in label
```

Click on Find All References (F7) or press F7 and you get the list below with all code lines containing the selected object.



Clicking on a line in the list shows that line in the middle of the IDE code area.

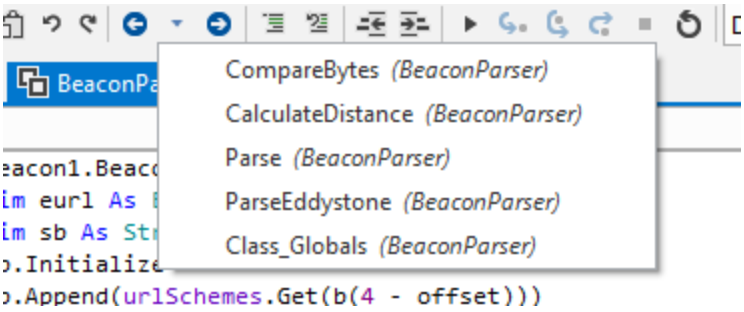
4.4 Navigating in the IDE

4.4.1 Alt + Left / Alt + Right Move backwards and forwards

Moves backwards and forwards based on the navigation stack. This is useful to jump back and forth between the last recent subs.

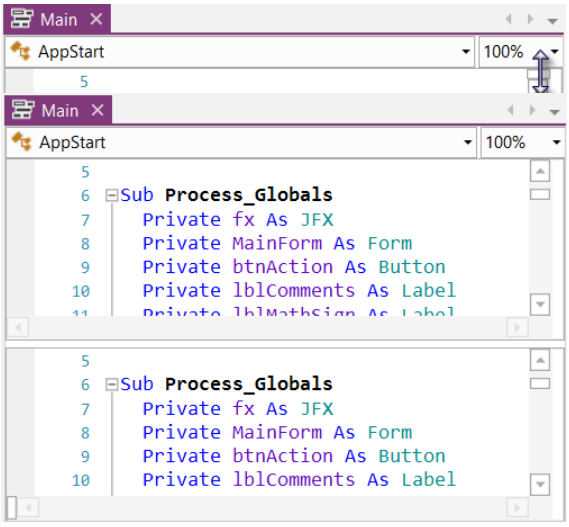
4.4.2 Alt + N Navigation stack menu

Opens the navigation stack menu. You can then choose the location with the up and down keys.

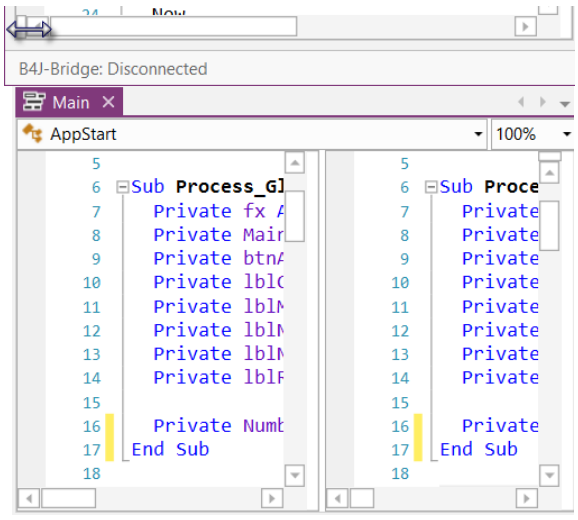


4.4.3 Split the screen

If you are working on two locations in the same module then you can split the code editor (it can be split again vertically):

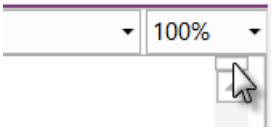


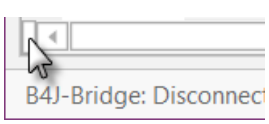
Horizontally



Vertically

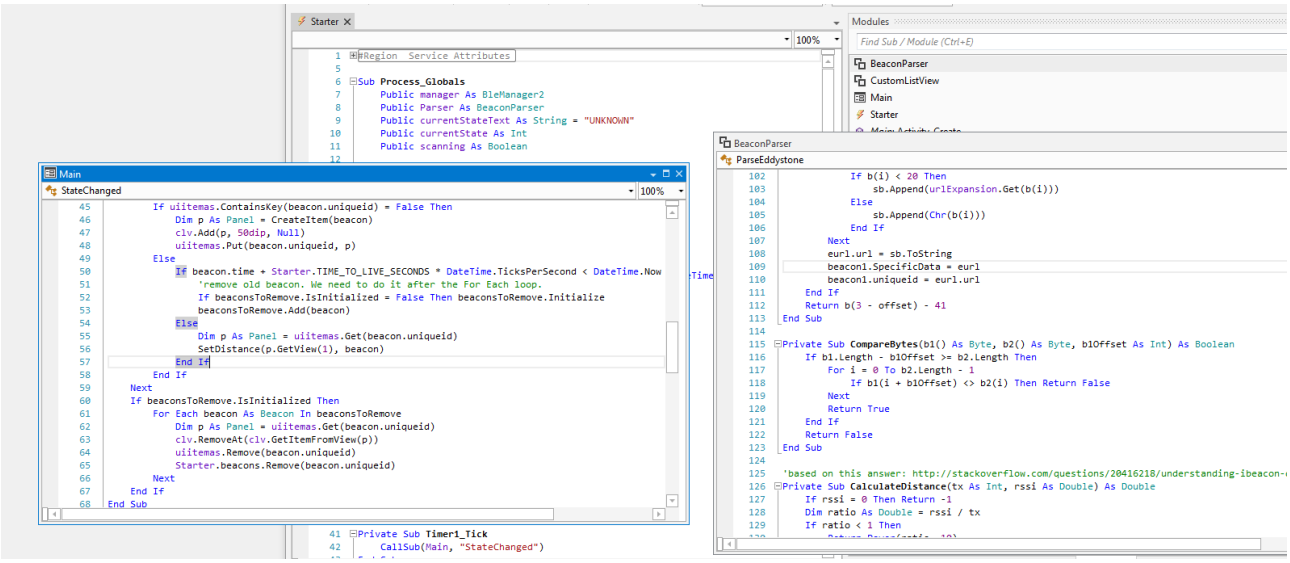
You can also double click on the small rectangles to split the screen.





4.4.4 Multiple windows

If you are working with multiple modules you can move the modules out of the main IDE as separate windows.



4.4.5 Ctrl + E Search for sub or module

Ctrl + E - searches for sub or module. Very useful when working with large projects.

4.4.6 Ctrl + Click on any sub or variable

Ctrl + Click on any sub or variable to jump to the declaration location.

4.4.7 F7 - Find all references

Not exactly related to navigation but is also useful when working with large projects.
Details in Find all references.

4.4.8 Ctrl + F Quick Search

Ctrl + F - Index based quick search. Details in Quick Search.

5 The Visual Designer

Designing layouts is a major concern for developers.
A well organized and nice looking user interface makes a program being accepted immediately by the users or not.
Most users, when they look at a new application, decide in the first minutes if they will go further or not! Me too, when I download an application and there are several with the same purpose, the first impression is crucial. If I don't like the layout I don't keep it.

It's up to you to define what layout you want, what you want to display at the same time and how you want to navigate through the different displays.

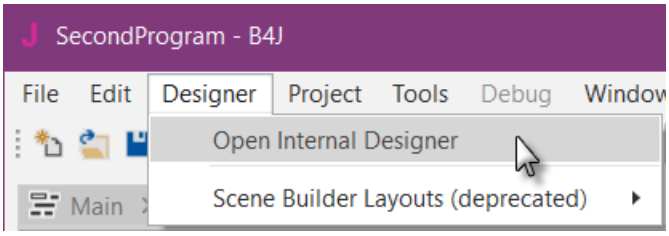
There are no general rules nor templates for user interfaces. They depend on the kind of application, the kind of information to display on what form size, the number of different pages depending on the screen size and the information, etc.

Several tools are at your disposal to design the layouts, these are explained in the following chapters.

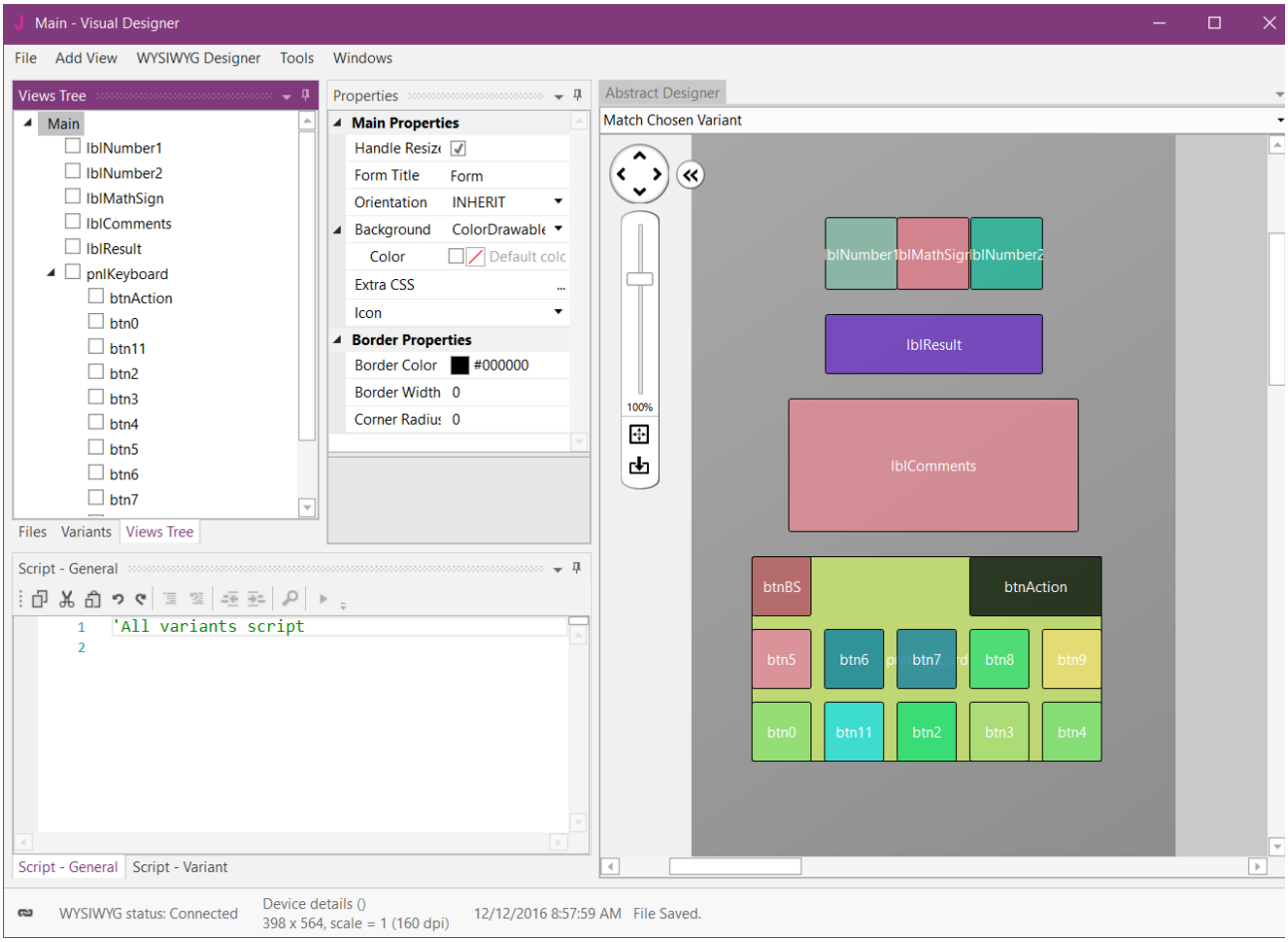
- The Visual Designer.
- The Abstract Designer.
- Anchors.
- Designer Scripts.

The Designer allows generating layouts with the Abstract Designer and a WYSIWYG form.

Launch the Designer in the IDE Menu Designer.

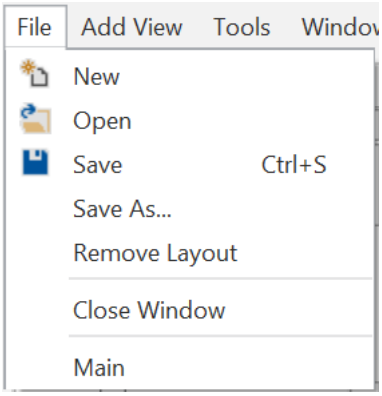


The default Visual Designer looks like this, the layout in the Abstract Designer is from the SecondProgram project.



5.1 The menu

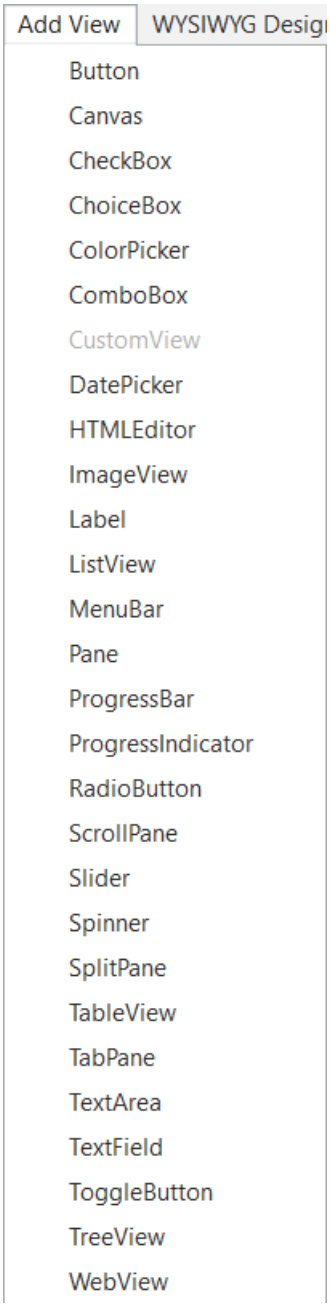
5.1.1 File menu



New	Opens a new empty layout.
Open	Opens an existing layout.
Save	Saves the current layout.
Save As…	Saves the current layout with a new name.
Remove Layout	Removes the layout from the Files directory.
Close Window	Closes the Visual Designer
Main	Layout file list, in this case only one file, 'Main'.

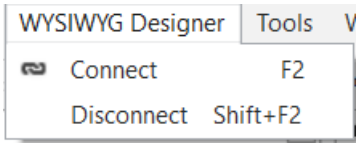
5.1.2 AddView menu

This menu allows you to select the view you want to add to the current layout.



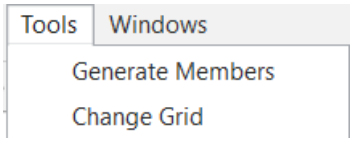
Button	adds a Button
Canvas	adds a Canvas
CheckBox	adds a CheckBox
ChoiceBox	adds a ChoiceBox
ColorPicker	adds a ColorPicker
ComboBox	adds a ComboBox
CustomView	adds a CustomView if there are any.
DatePicker	adds a DatePicker
HTMLEditor	adds an HTMLEditor
ImageView	adds an ImageView
Label	adds a Label
ListView	adds a ListView
MenuBar	adds a MenuBar
Pane	adds a Pane
ProgressBar	adds a ProgressBar
ProgressIndicator	adds a ProgressIndicator
RadioButton	adds a RadioButton
ScrollPane	adds a ScrollPane
Slider	adds a Slider
Spinner	adds a Spinner
SplitPane	adds a SplitPane
TableView	adds a TableView
TabPane	adds a TabPane
TextArea	adds a TextArea
TextField	adds a TextField
ToggleButton	adds a ToggleButton
TreeView	adds a TreeView
WebView	adds a WebView

5.1.3 WYSIWYG Designer menu



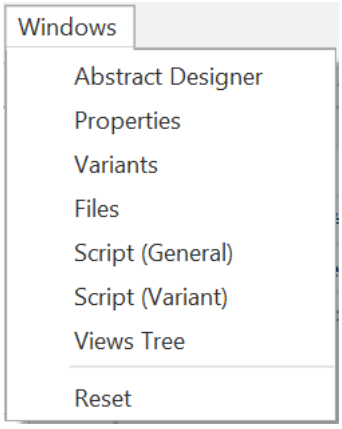
Shows the WYSIWYG form.
Hides the WYSIWYG form.

5.1.4 The Tools menu



Generate Members	Members generator
Change Grid	Allows to change the grid size

5.1.5 Windows menu

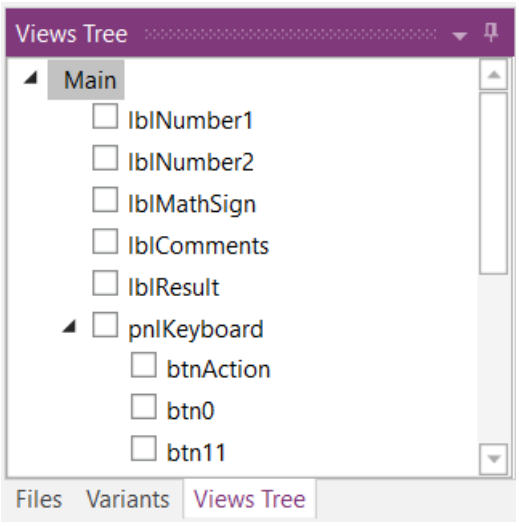


Shows the Abstract Designer window.
Shows the Properties window.
Shows the Variants window.
Shows the Files window.
Shows the Script (General) window.
Shows the Script (Variant) window.
Shows the Views window.
Resets the Visual Designer layout to the default layout.

5.2 Visual Designer Windows

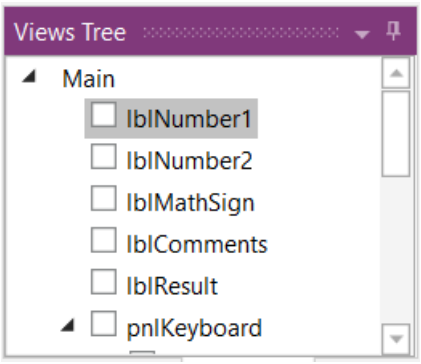
The Visual Designer is composed of different windows.

5.2.1 Views windows Views Tree / Files / Variants



In this Window three windows are combined:
Files, Variants and Views Tree.

5.2.1.1 Views Tree window



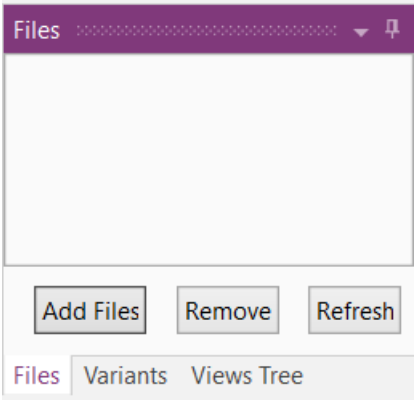
Shows all nodes of the selected layout in a tree.

When you select a node in the list, all the properties of the selected node are displayed in the Properties window.

You can select several nodes at the same time and change common properties.

The selected nodes are highlighted in the Abstract Designer.

5.2.1.2 Files Window

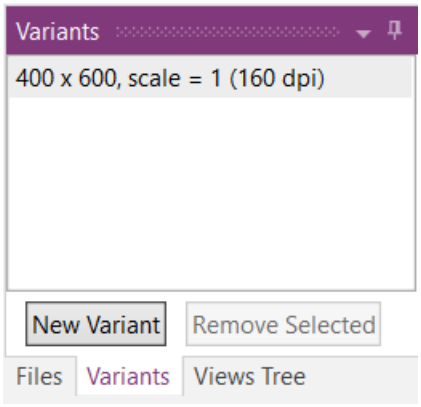


Used to add or remove files to the Visual Designer, mainly image files.

File handling is explained in the Image Files chapter.

The files you add are copied to the Files folder of the project and can be accessed in the code in the File.DirAssets folder.

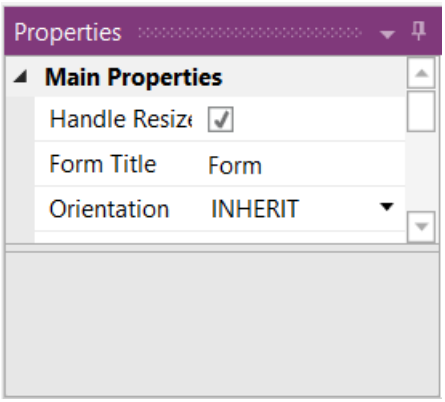
5.2.1.3 Variants window



Used to add and remove layout variants.

Layout variants are explained in the Layout variants chapter.

5.2.2 Properties window



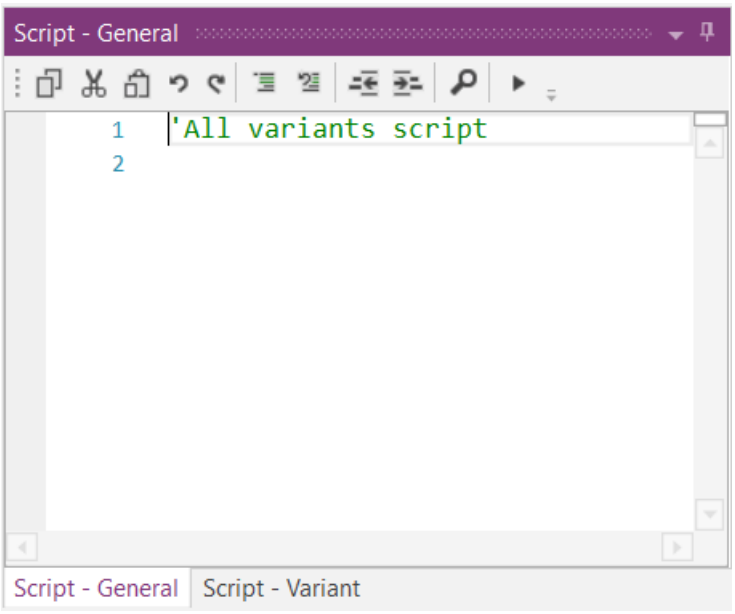
The Properties window shows all properties of the selected node.

The Properties are explained in the Properties list chapter.

5.2.3 Script (General) / (Variant) windows

In the Scrip windows you can add code to position and resize nodes.
Two windows are available:

- **Script - General** Code valid for all layout variants.
- **Script - Variant** Specific code for the selected variant.

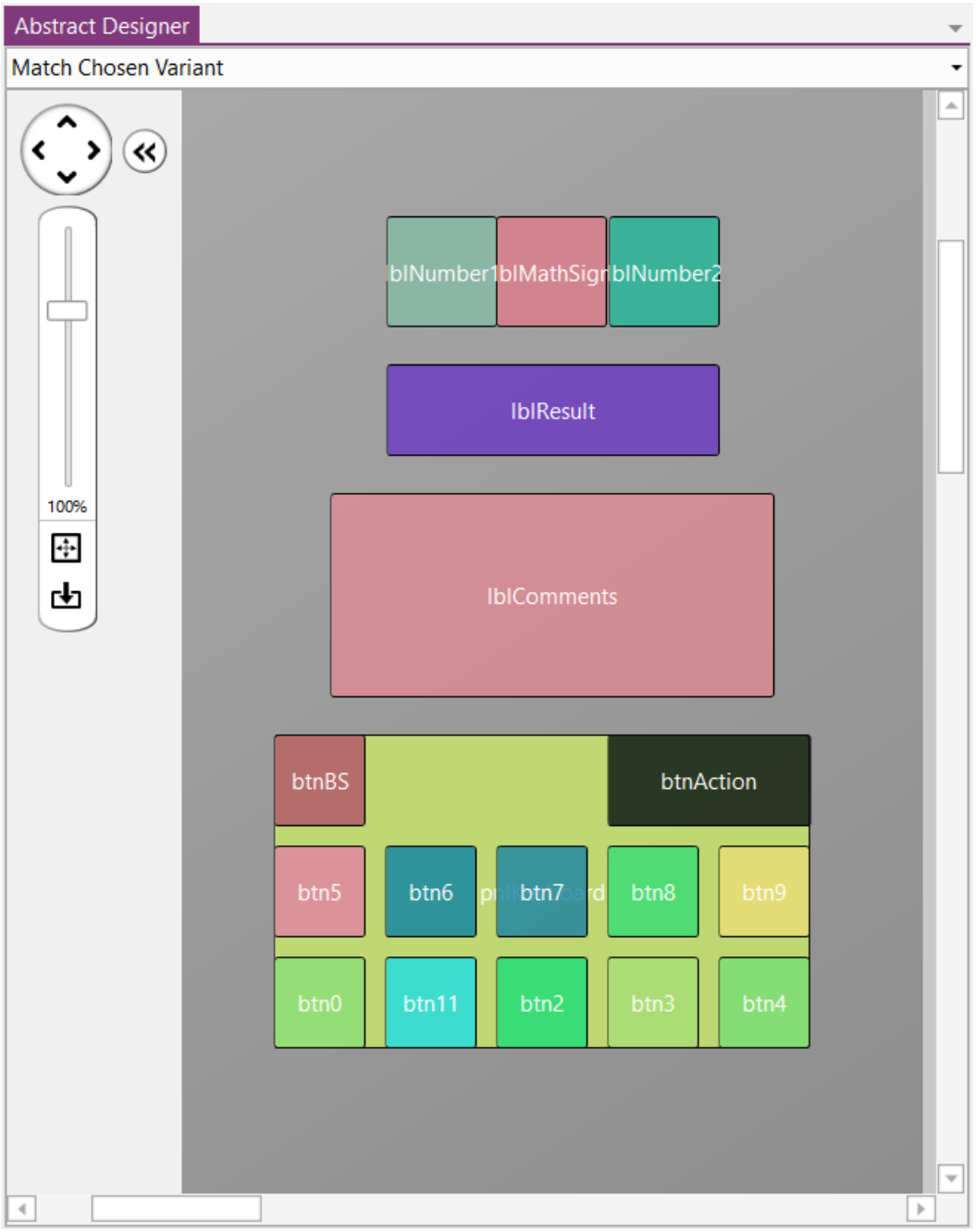


Script code is explained in the Designer Scripts chapter.

5.2.4 Abstract Designer window

The Abstract Designer allows to select, position and resize nodes.
It is not a WYSIWYG Designer, for this you need to connect a real device or an Emulator.

The displayed layout in the picture below is from the SecondProgram project.



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

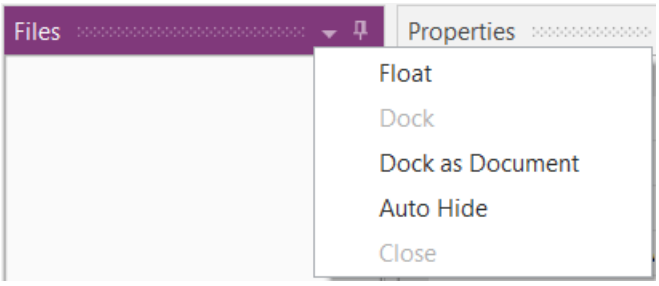
5.3 Floating windows

You can define your own Visual Designer layout, rearrange the windows in size and position, docked or floating.

On the top right of each window two icons allow you to manage the behavior of this window.

Options.

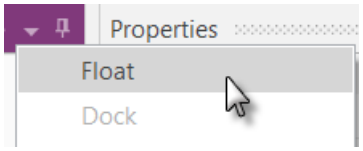
Example with the Files window:



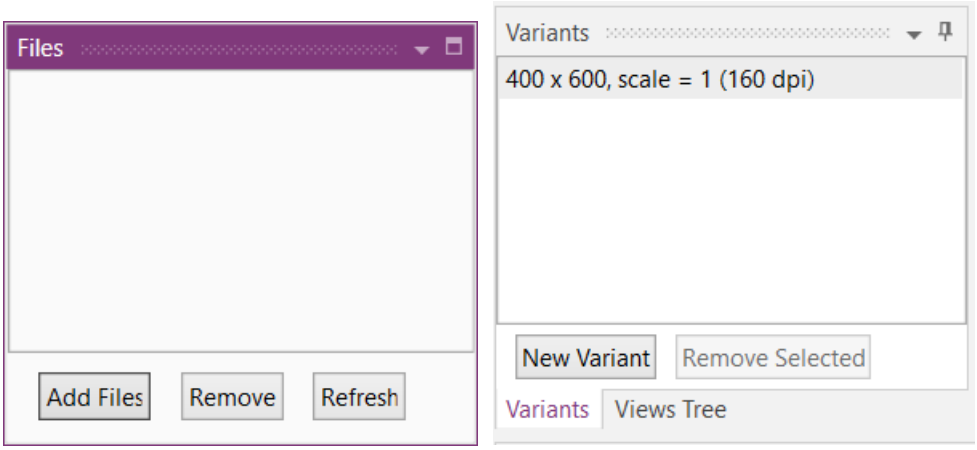
Float sets the window to Float, independent of the Visual Designer window.

Dock as Document.
Auto Hide.

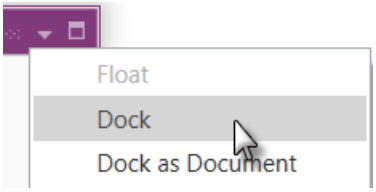
5.3.1 Float

Click on 

The Files windows is independent from the Visual Designer and is removed from the nodes window. You can move it where ever you want on the screen.

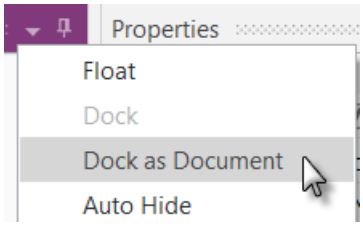


5.3.2 Dock

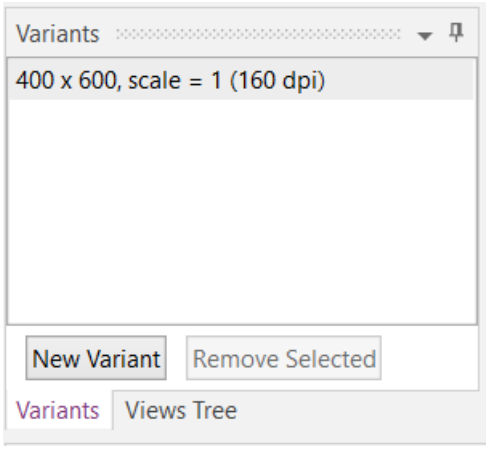
In a floating window click on 

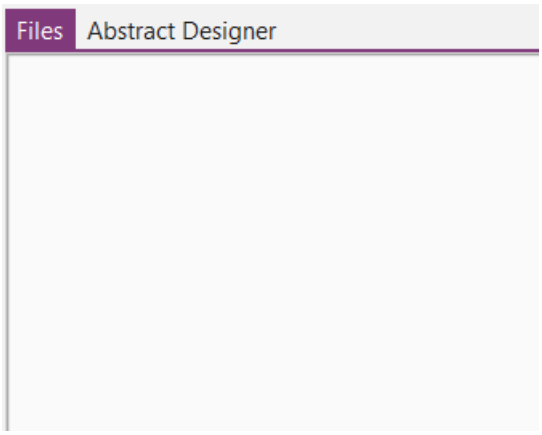
The window is moved back to the Views window.

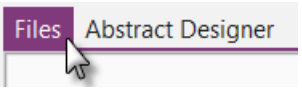
5.3.3 Dock as Document

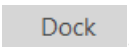
Click on 

The window is removed from its parent window and added to the Abstract Designer window.

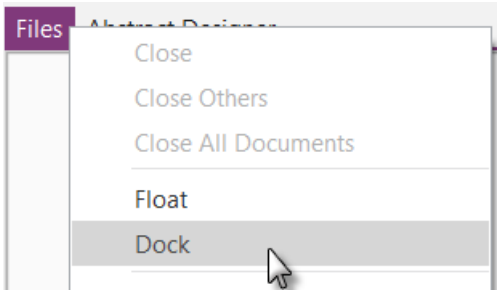




To move it back to its parent window right click on 

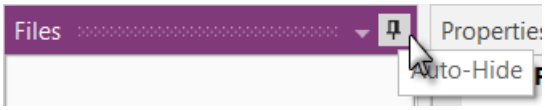
and on 

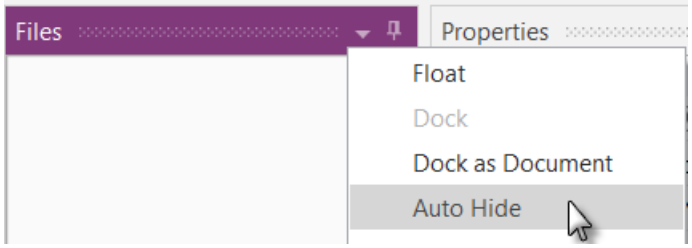
.



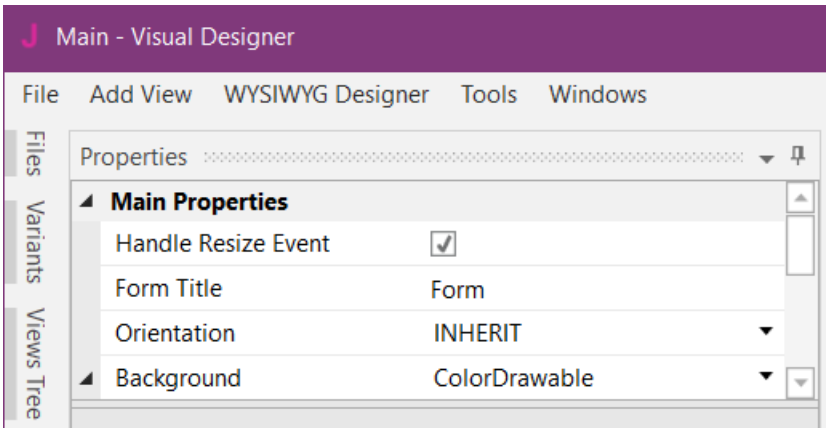
5.3.4 Auto Hide

Click either on  or on Auto Hide.

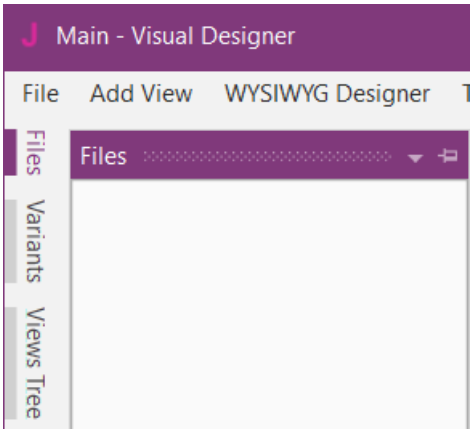




The three windows: Files, Variants and Views Tree are moved as Tabs to the left border of the Visual Designer. The Properties window width is increased.

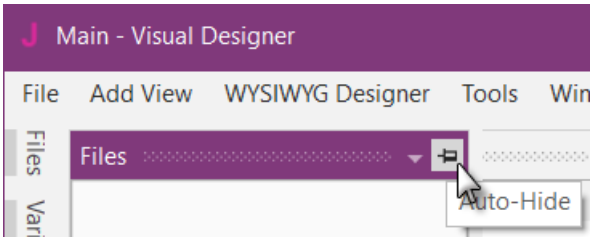


Click on a Tab to show the window.



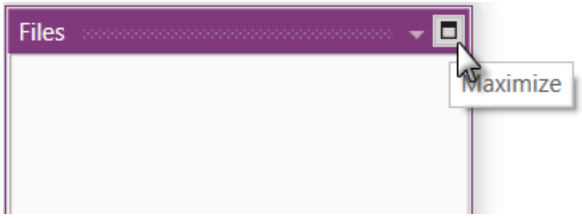
When you click somewhere else, outsides the selected window, hides it automatically.

Click on  in the title to move the windows back to their previous position.

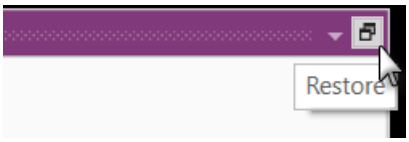


5.3.5 Maximize

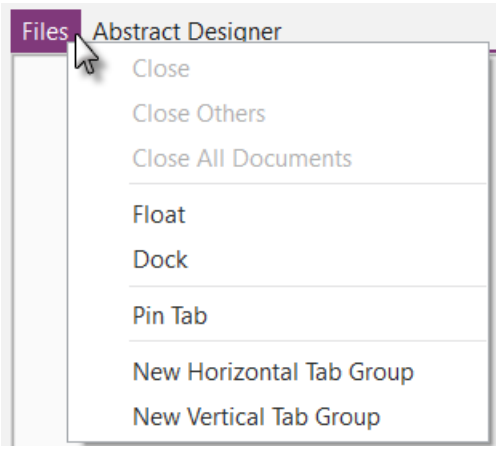
Floating windows can be maximized.



To set it back to its previous size, click on  in the top right corner.

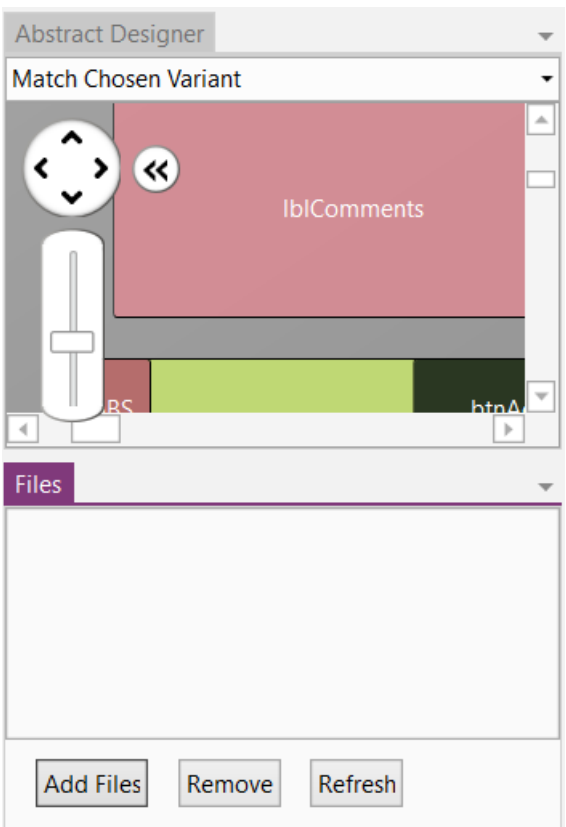


5.3.6 New Horizontal / Vertical Tab Group

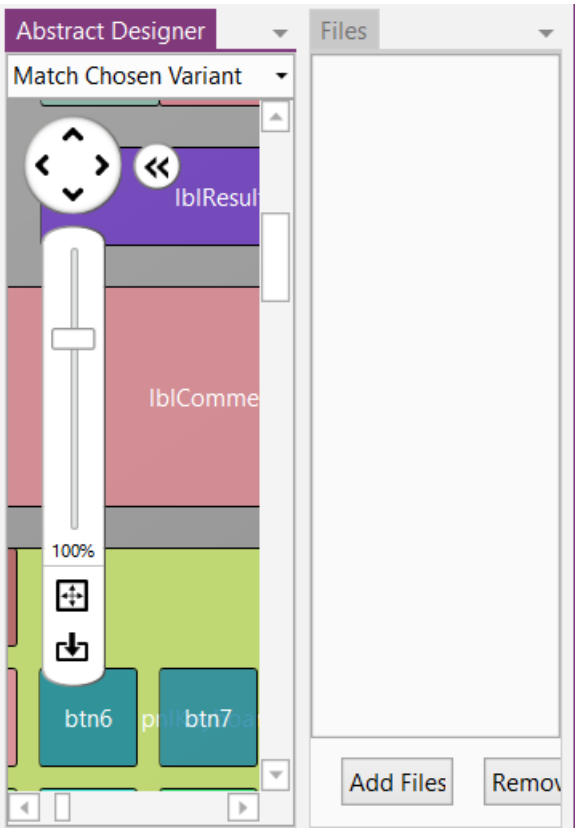


When a window is set as *Dock as Document* two other options are available.

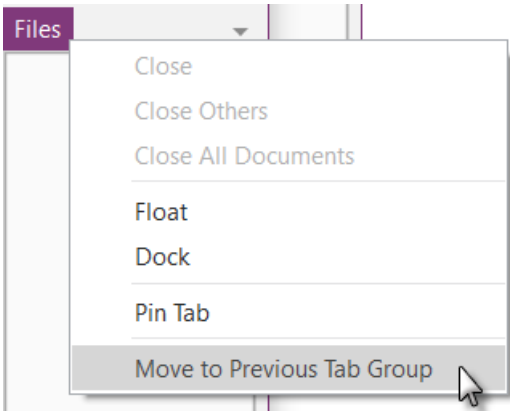
New Horizontal Tab Group
New Vertical Tab Group



New Horizontal Tab Group

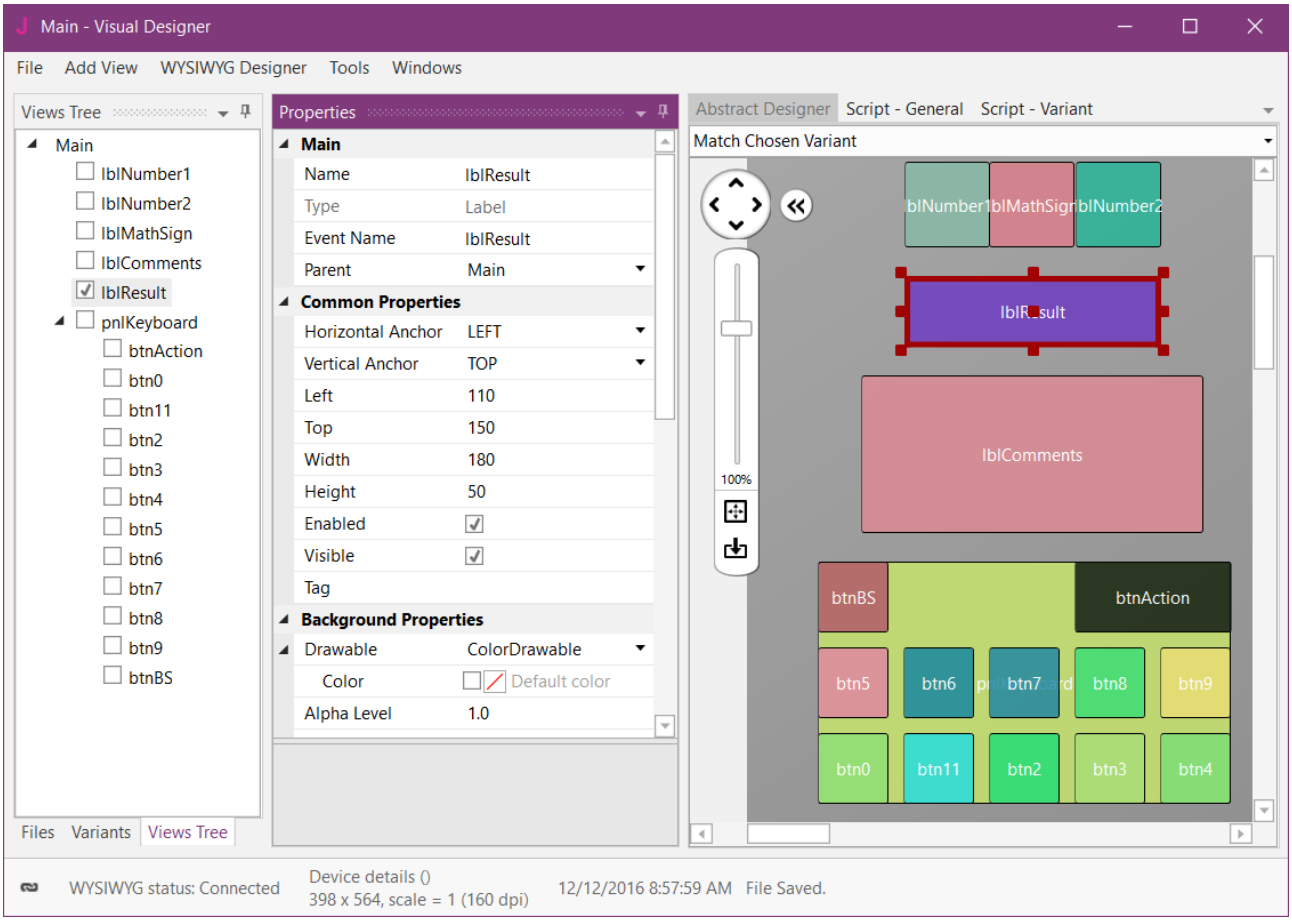


New Vertical Tab Group



To remove Tab Group right click on Files and click on Move to Previous Tab Group.

My preferred Designer layout:

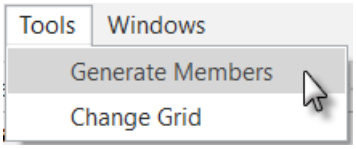


I moved the two Script widows as *Dock as Document* onto the Abstract Designer window. That way the Views and Properties windows is much higher.

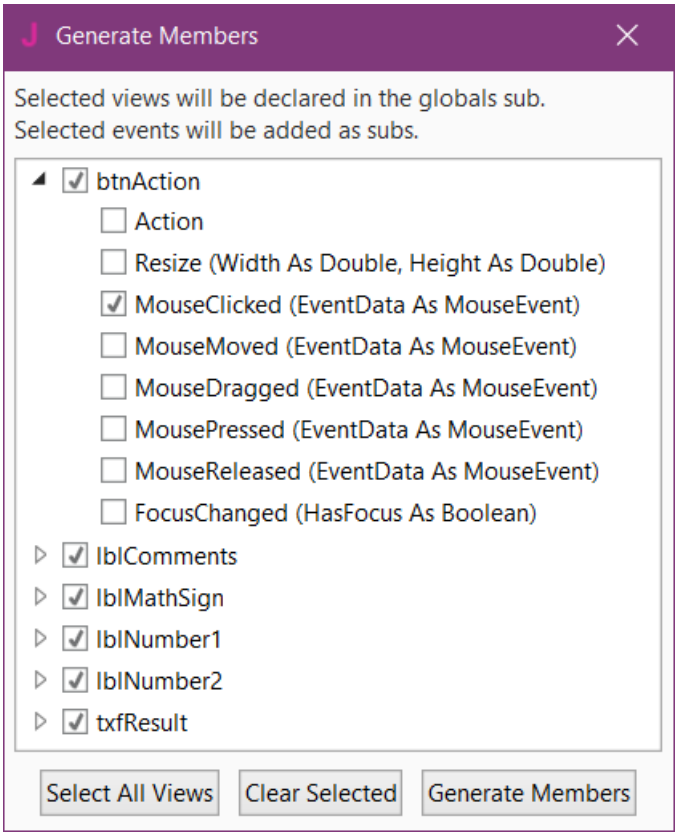
5.4 Tools

5.4.1 Generate Members

Allows generating Dim statements and subroutine frames.
The example is based on the project MyFirstProgram.



In the Tools menu click on Generate Members 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 ▷ ☑ lblNumber1 generates its reference in the Process_Globals Sub in the code. This is needed to make the node being recognized and allow the autocomplete function.

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

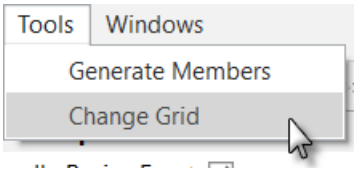
You can open a list for the available events of a node ◢ ☑ btnAction :
Clicking on an event ☑ MouseClicked generates the Sub frame for this event.

```
Sub btnAction_MouseClicked (EventData As MouseEvent)

End Sub
```

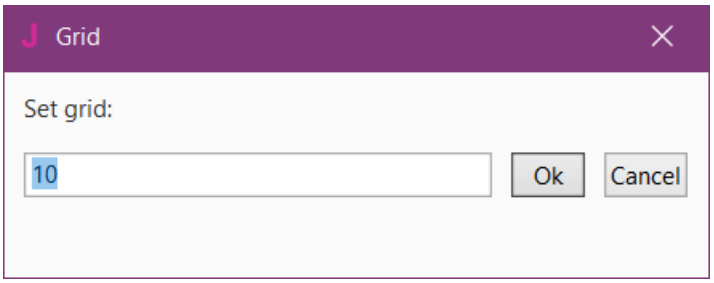
Click on Generate Members to generate the references and sub frames and close the window.
Click on Select All Views to select all nodes in the list,
Click on Clear Selected to clear the current selections.

5.4.2 Change grid

The grid is an invisible grid with a given size. The default grid size is 10 pixels. That means that all positions and dimensions of a node will be set to values in steps corresponding to the grid size. Moving a node will be done in steps equal to the grid size.



Click on Change Grid in the Tools menu.

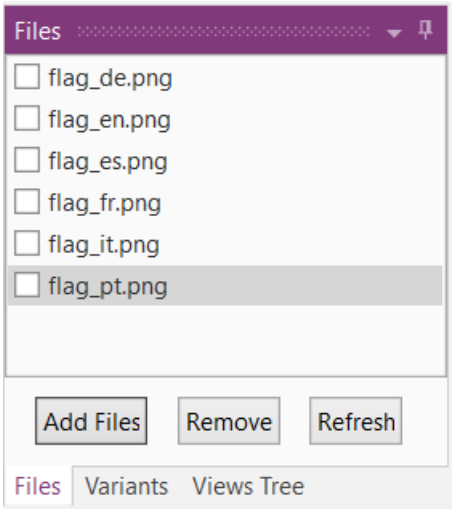


You can change the grid size to the value you want.

The value is saved in the layout file, you will get the same value when you reload this layout.

The default value when you start a new project is 10.

5.5 Image files



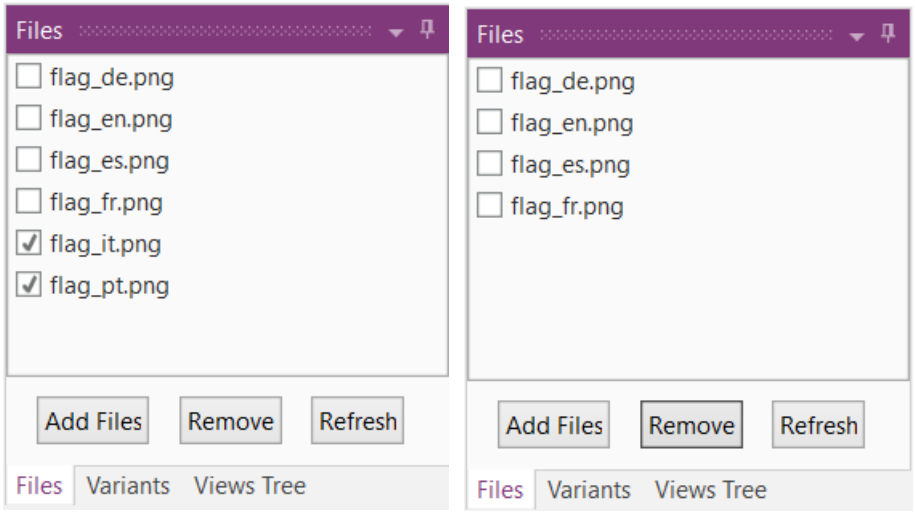
You can add image files to the layout.

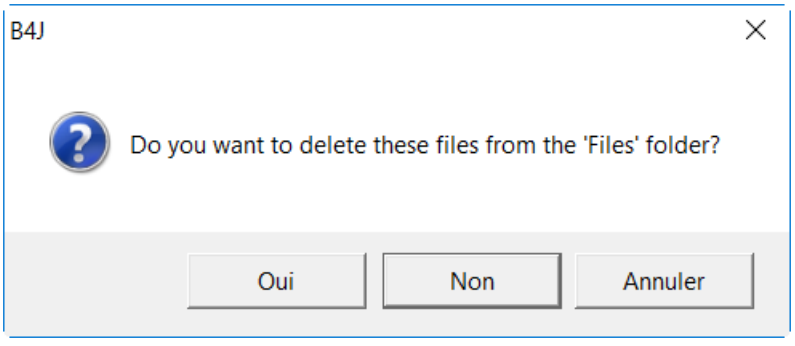
Click on Add Files to select the files(s) to add.

These files will be listed in the Image Files list.

These files are saved to the Files folder of the project and can be accessed in the code in the Files.DirAssets folder.

To remove files, check the files to remove and click on Remove.



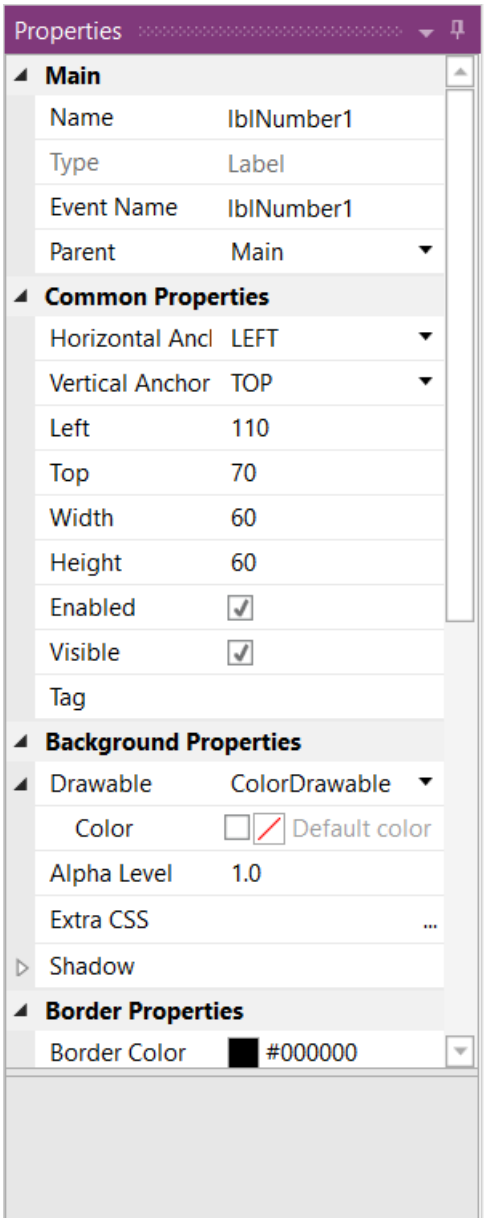


You are asked if you want to delete the files from the 'Files' folder.

Oui = Yes
Non = No
Annuler = Cancel

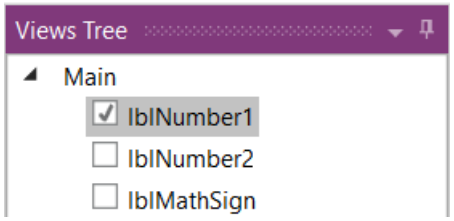
When you answer Yes make sure to have a copy of the files you remove, because they are removed from the Files folder, but not transferred to the Recycle Bin, which means that they are definitely lost if you don't make a copy.

5.6 Properties list



The example is based on the SecondProgram code.

Select for example lblNumber1 in the Views Tree list.



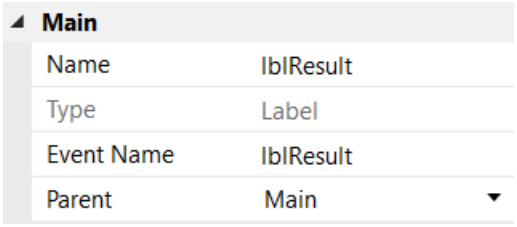
All the properties of lblNumber1 are displayed.
These are organized in groups.

All properties can be modified directly in the list.

All properties in the Main group and some of the properties in the other groups are common to all node types.

Explanation of some general properties for all types of nodes in the next chapters.

5.6.1 Main properties



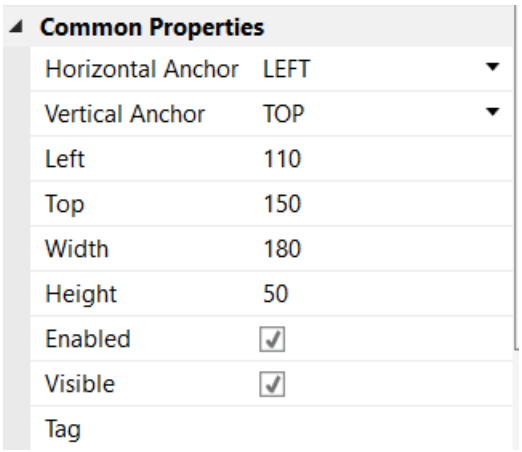
Name Name of the node. It is good practice to give meaningful names. Common usage is to give a 3 character prefix and add the purpose of the node. In the example, the node is of type Label and its purpose is to show a result. So we give it the name "lblResult", "lbl" for Label and "Result" for the purpose. This does not take much time during the design of the layout but saves a lot of time during coding, debugging and maintenance of the program.

Type Type of the node, not editable. It is not possible to change the type of a node. If you need to, you must remove the node and add a new one.

Event Name Generic name for the subroutines that manages the node's events. By default, the Event Name is the same as the node's name like in the example. The Events of several nodes can be redirected to a same subroutine. In that case you must enter the name of that routine. Look at the SecondProgram example for the Click event management for the buttons of the keyboard in the btnEvent_MouseClicked routine.

Parent Name of the parent node. Main, in the example. The parent node can be changed in selecting the new one in the list.

5.6.2 Common properties



HorizontalAnchor Horizontal Anchor function. Possible values LEFT, RIGHT or BOTH

VerticalAnchor Vertical Anchor function. Possible values TOP, BOTTOM or BOTH

Left X coordinate of the left edge of the node from the left edge of its parent node, in points.

Top Y coordinate of the upper edge of the node from the upper edge of its parent node, in points.

Width Width of the node in pixels.

Height Height of the node in pixels.

Enabled Determines if the node is enabled or not. Not enabled means no user interface.

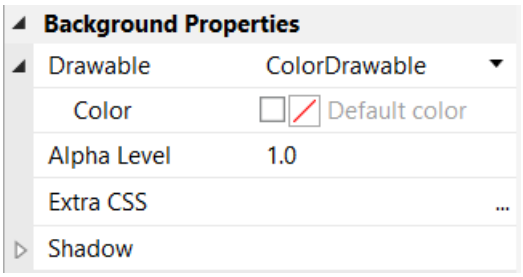
Visible Determines if the node is visible to the user or not.

Tag This is a place holder which can be used to store additional data. Tag can simply be text but can also be any other kind of object.
Tag is used in the SecondProgram example for the numeric buttons click events management in the btnEvent_Click routine.

Background color Color of the nodes background.

Alpha level Sets the transparency, 1 = opaque 0 = transparent.

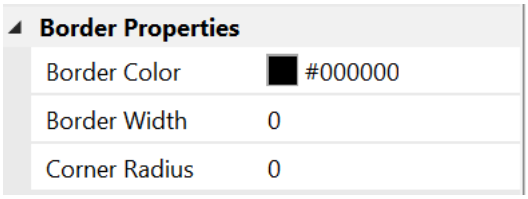
5.6.3 Background properties



Drawable
Color
Alpha Level

Extra CSS
Shadow

5.6.4 Border properties

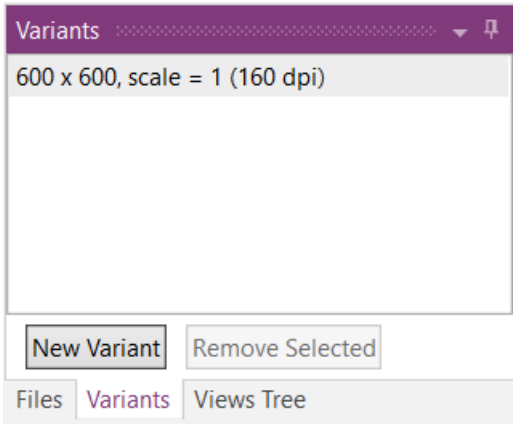


Border color Color of the nodes border.
Border Width Width of the nodes border in points.
Corner Radius Sets the corner radius in points.

5.7 Layout variants

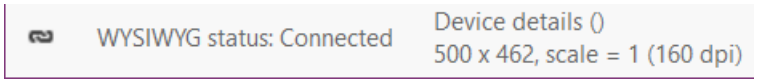
Layout variants are much less important in B4J than in B4A and B4i.

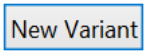
The big difference is that in B4A and in B4i the Activity size or Page size is equal to the screen size.
In B4J, Forms can have any size and are by default resizable.
Therefor you can use one layout and adjust the Nodes with Anchors and in the Designer Scripts.



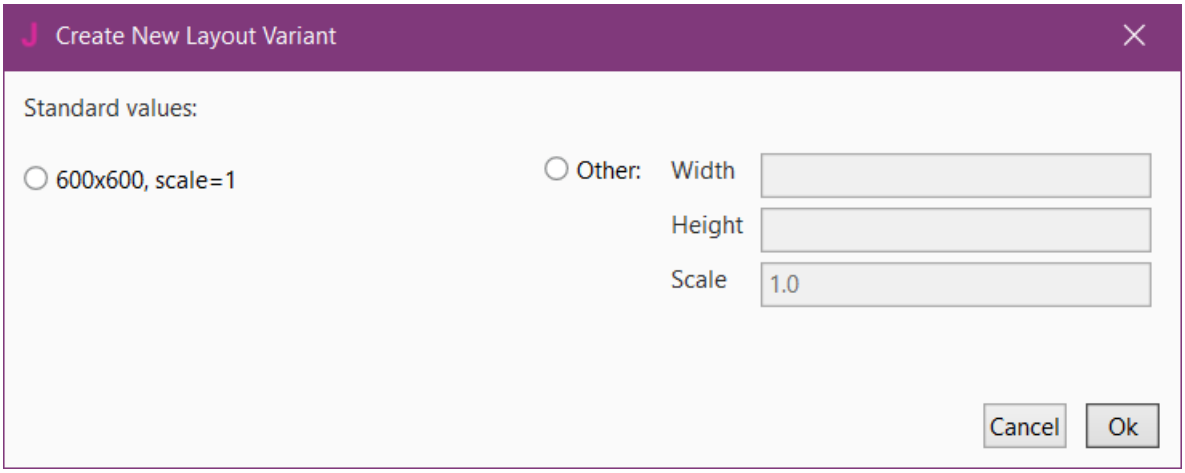
The default variant is 600 x 600 pixels.
Different layout variants can be managed in a same layout file.

In the bottom left corner of the Visual Designer you find the size of the WYSIWYG form. If you resize the form the values are automatically updated.



Click on  to add a new variant.

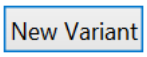
There is only one Standard variant 600 x 600 pixels.



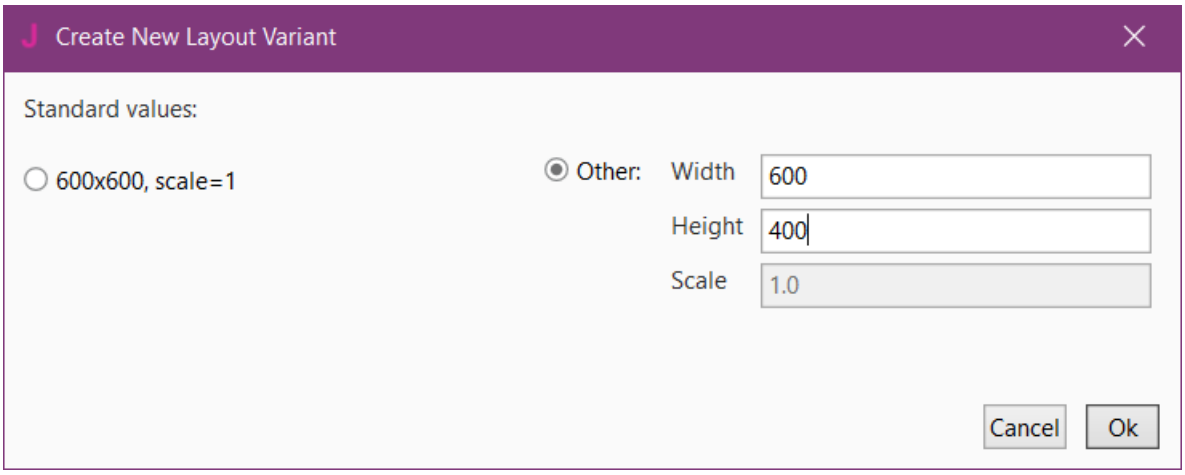
You can add any other layout variant.

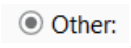
Let us make an example project: LayoutVariants1
The source code is saved in the Guide\SourceCode\LayoutVariants\LayoutVariants1 directory.
- Run the IDE.
- Run the Designer.

On top, we add a Label with the whole with and 4 Labels one in each corner.
We'll define 3 layout variants: 600 x 400, 1200 x 600 and 600 x 1000.

Click on 

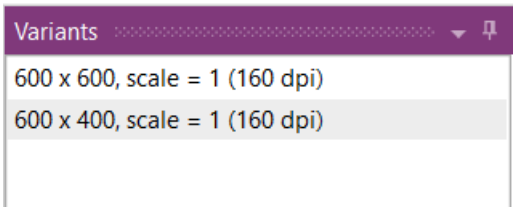
to add a new variant.



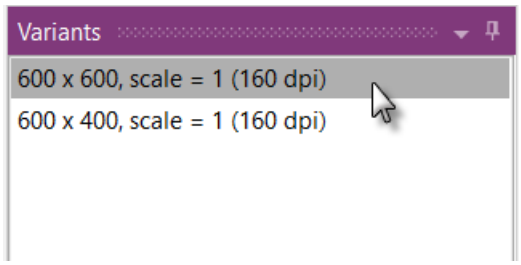
Click on 

, enter the two values and click on 

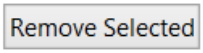
.



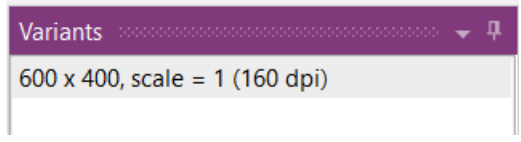
The new variant is added in the list.



Now, we remove the original variant.

Select the 600 x 600 variant
and click on 

.



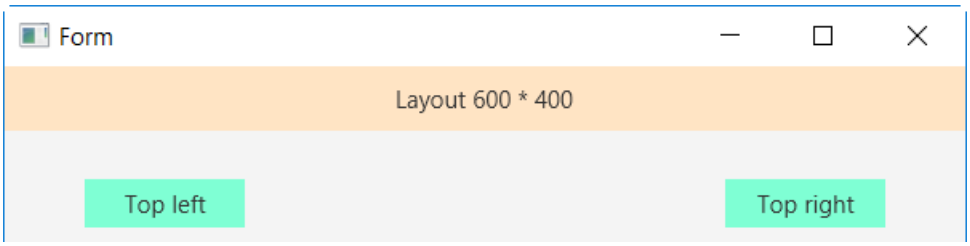
The scope of this chapter is to show how layout variants work and not define a layout.
Load the LayoutVariants1 project and look at the layout variants. To know which variant was selected by the system I added a text in each Scipt-Variant displayed in the title label.

Which layout variant is selected by the program when you run it?
The system selects the layout variant that fits best the given size.

On top of the code we have the Form size:

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

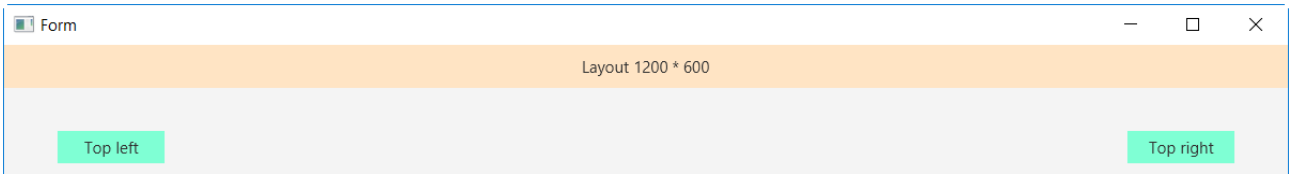
When we run it, we see that the program selected the 600 x 400 layout.



Now, change the size to:

```
#Region Project Attributes
   #MainFormWidth: 1200
   #MainFormHeight: 600
#End Region
```

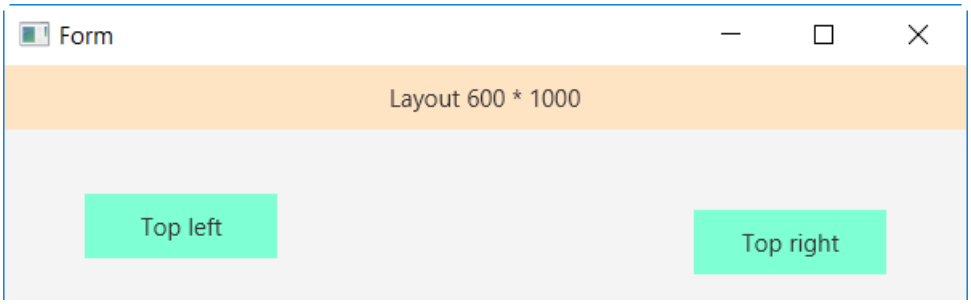
When we run it, we see that the program selected the 1200 x 600 layout.



Now, change the size to:

```
#Region Project Attributes
   #MainFormWidth: 600
   #MainFormHeight: 1000
#End Region
```

When we run it, we see that the program selected the 600 x 1000 layout.

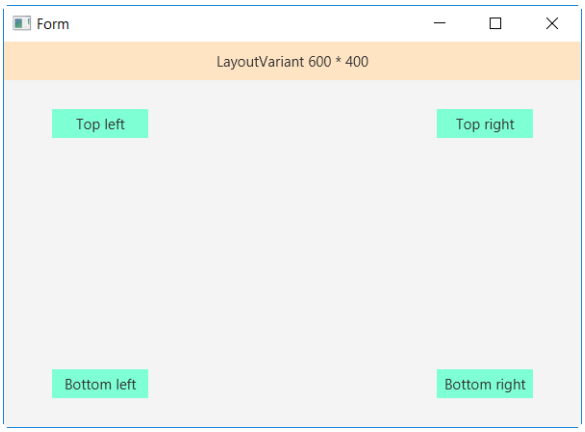


If the form size is not exactly the same, B4J will choose the closest layout variant.

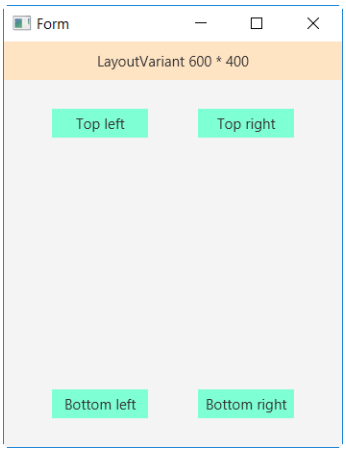
In the LayoutVariants2 project we use one layout variant 600 x 400 and use anchors.

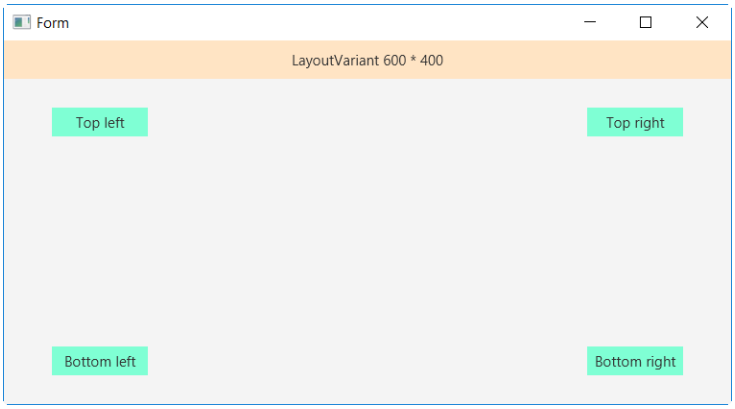
Node	Anchor	Value
lblTitle	Horizontal anchor:	BOTH
lblTopRight	Horizontal anchor:	RIGHT
lblBottomLeft	Vertical anchor:	BOTTOM
lblBottomRight	Horizontal anchor:	RIGHT
lblBottomRight	Vertical anchor:	BOTTOM

And the result:



If we resize the form, we see that the title always fills the whole width and the 4 other labels remain in their corner.
Some results (the images have been reduced to fit into the page):

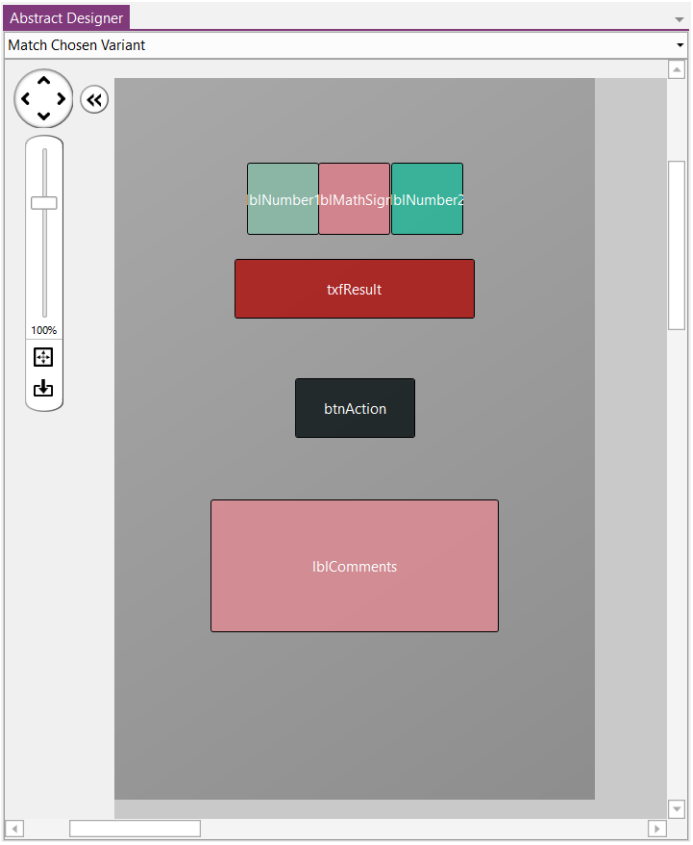




You can add size limits with:
SetWindowSizeLimits(MinWidth As Double, MinHeight As Double, MaxWidth As Double, MaxHeight As Double)

More details in the Anchors chapter.

5.8 The Abstract Designer

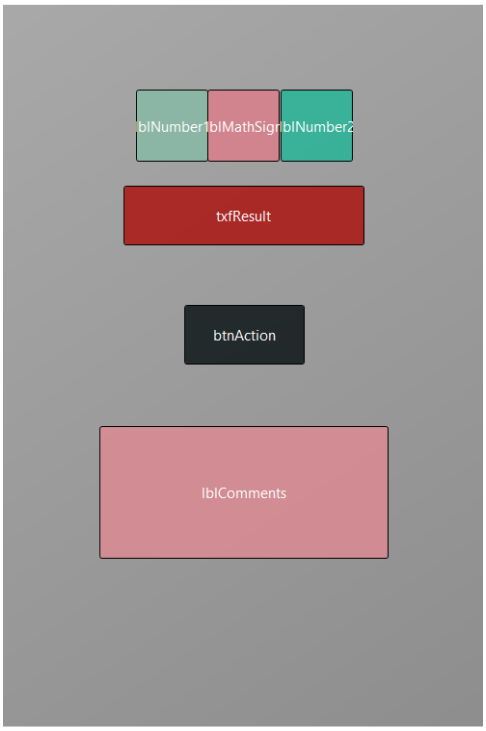


The Abstract Designer is a tool that shows the layout in a separate window and is part of the Visual Designer.
Its main purpose is to create layouts, you can create different layout variants.

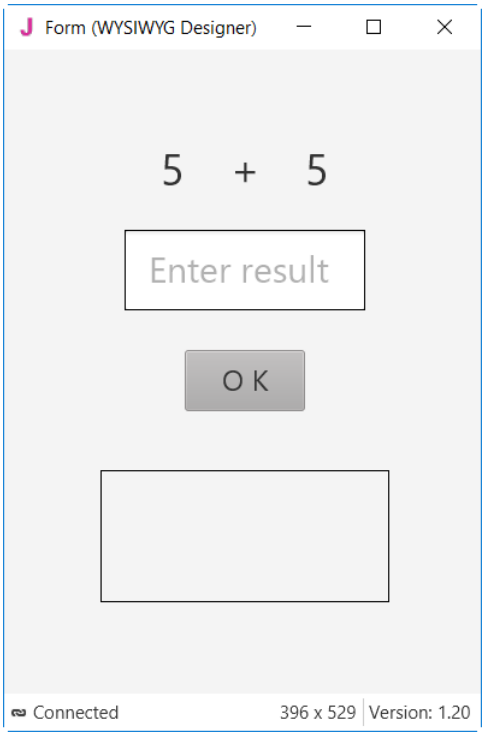
The different nodes are not shown with their exact shape but only as colored rectangles. Clicking on a node shows its properties in the Properties window.

You see the exact shape of the current layout in the WYSIWYG form .

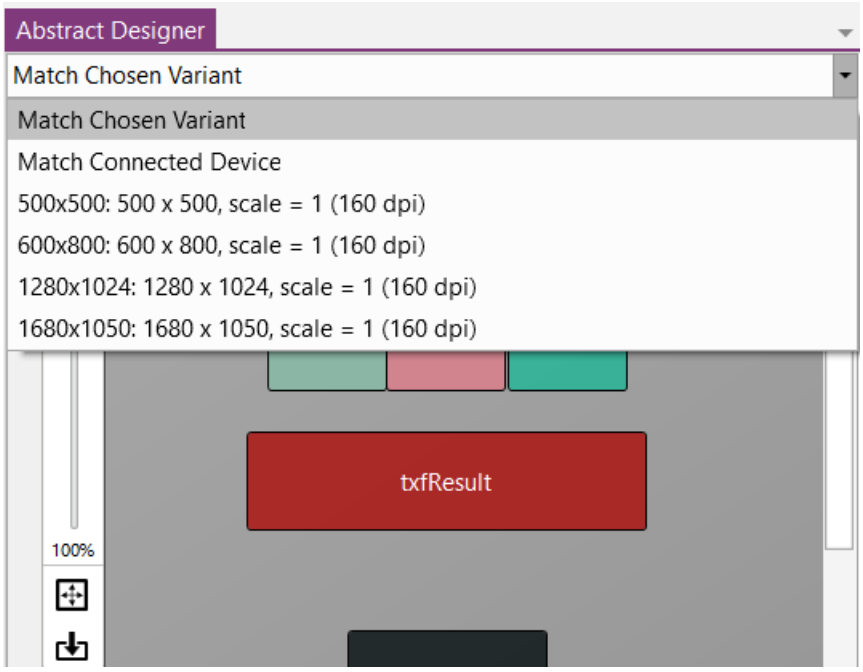
Abstract Designer



WYSIWYG form



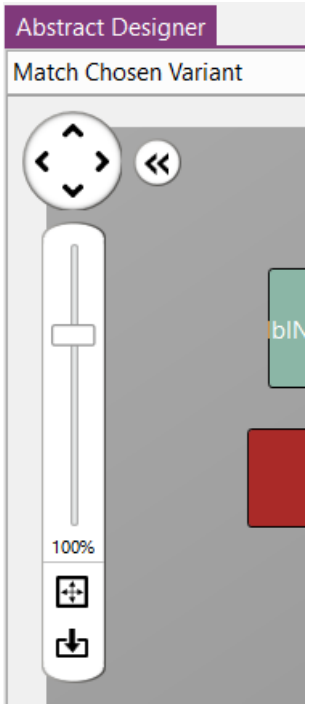
5.8.1 Selection of a screen size



On top you can select different screen sizes:

Match chosen Variant.
Matches the variant selected in the Variant window.
Match Connected Device.
Matches the size of the WYSIWYG form.
If you change the WYSIWYG form size you will see the change in the Abstract Designer.
Different 'standard' sizes.
This allows you to see how a layout looks on different form sizes.

5.8.2 Zoom



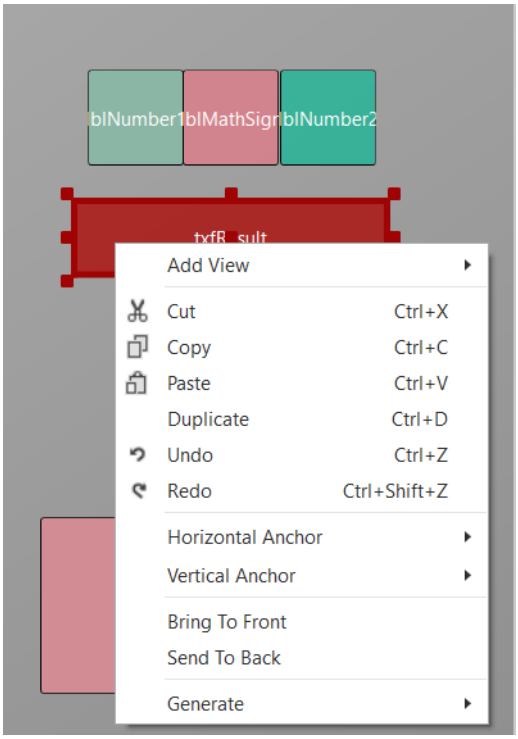
With  you can hide the zoom cursor and show it again with  .

With the cursor you can set the zoom level you want.

With  you can zoom to fit the selected screen size.
With  you can reset the zoom back to 100%.

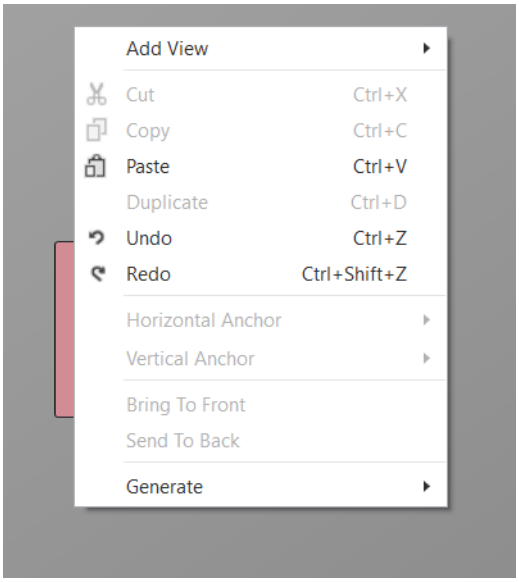
With the bottom and side cursors you can move the layout vertically or horizontally.

5.8.3 Context menus



Most editing functions can be accessed in a popup menu which is displayed when you right click on a node.

Add View
Cut
Copy
Paste
Duplicate
Undo
Redo
Horizontal Anchor
Vertical Anchor
Bring To Front
Send To Back
Generate



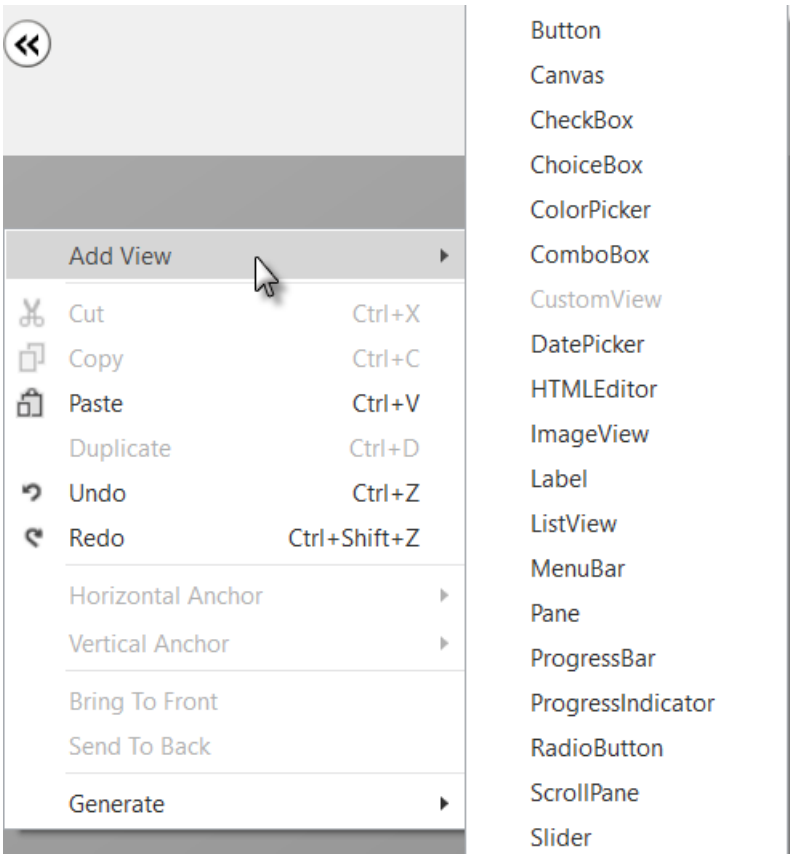
Right clicking somewhere on the Form area shows the context menu with some functions disabled which are not relevant for a Form.

Only Add View, Paste, Undo, Redo and Generate are available for a Form.

5.8.3.1 Add node

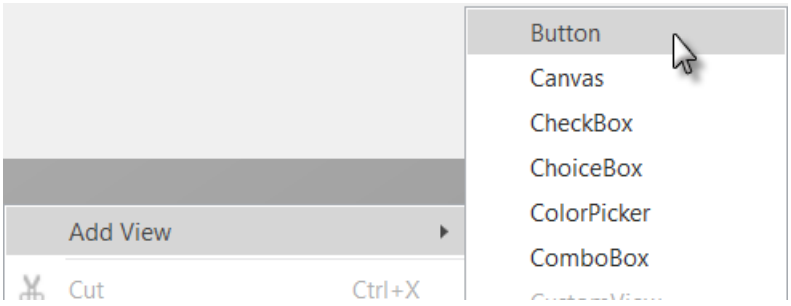
Right click somewhere on the parent node, Form or Pane, where you want to add a new node and move the cursor onto Add View.

This function is the same as the Add View function in the Visual Designer menu.

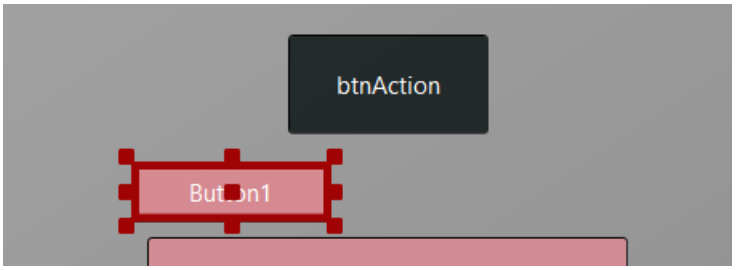


The list of all available nodes is displayed.

Click on the desired node to add it. Not all nodes are displayed in the picture.

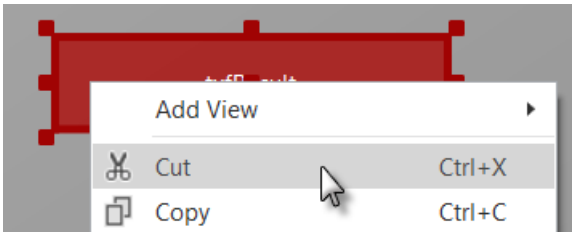


Example for a Button.



The Button is added to the layout.

5.8.3.2 Cut



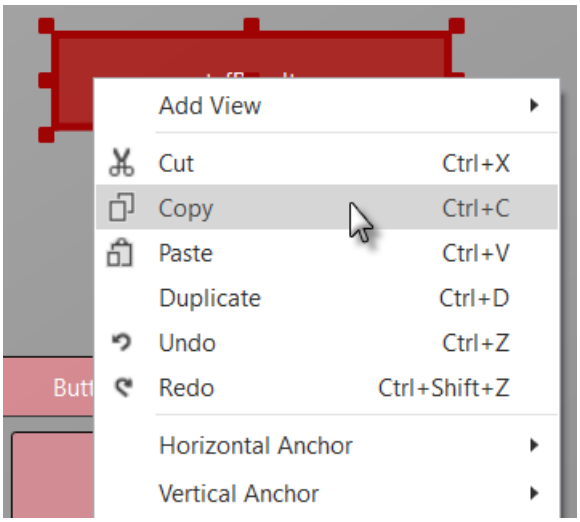
Right click on a node to select it and
click on Cut Ctrl+X to cut it.

There is no message for confirmation.

If you selected a Panel, it will be removed with all its child nodes!

If you cut it by accident click on Undo Ctrl+Z to recover it.

5.8.3.3 Copy / Paste / Duplicate Selected nodes

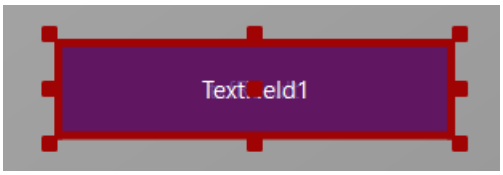


You can copy and paste nodes onto the same layout or from one layout to another one.

To copy and paste nodes on the same layout use Duplicate Ctrl+D it's faster,
one click to copy and paste.

Right click on a node to select it and
click on Copy Ctrl+C to copy it to the clipboard.

You can select several nodes and copy them.



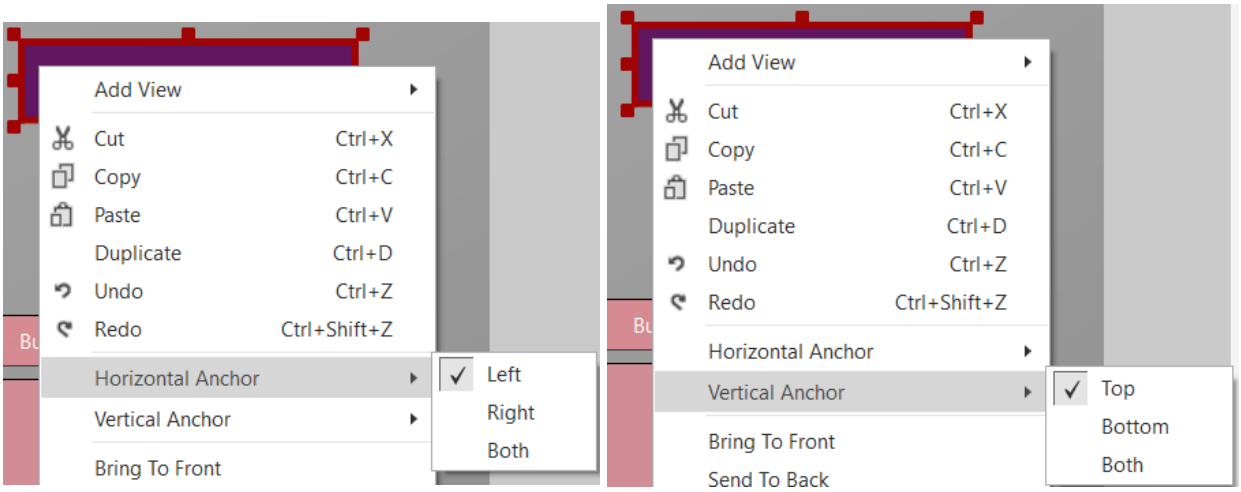
Right click again somewhere else and
click on Paste Ctrl+V
to paste the node, it will be copied over the original.

5.8.3.4 Undo / Redo

Undo Ctrl+Z Redo Ctrl+Shift+Z

These two functions allow you to undo or redo the last operations.

5.8.3.5 Anchors horizontal / vertical



Right click on a node and click on

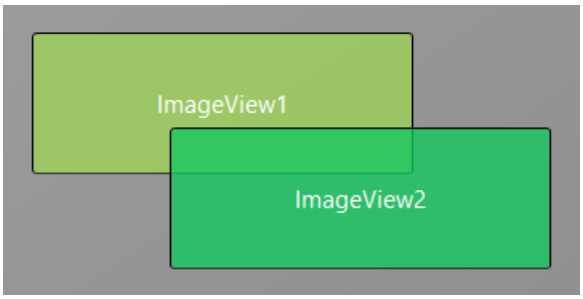
Horizontal Anchor or Vertical Anchor

and select the desired anchor.

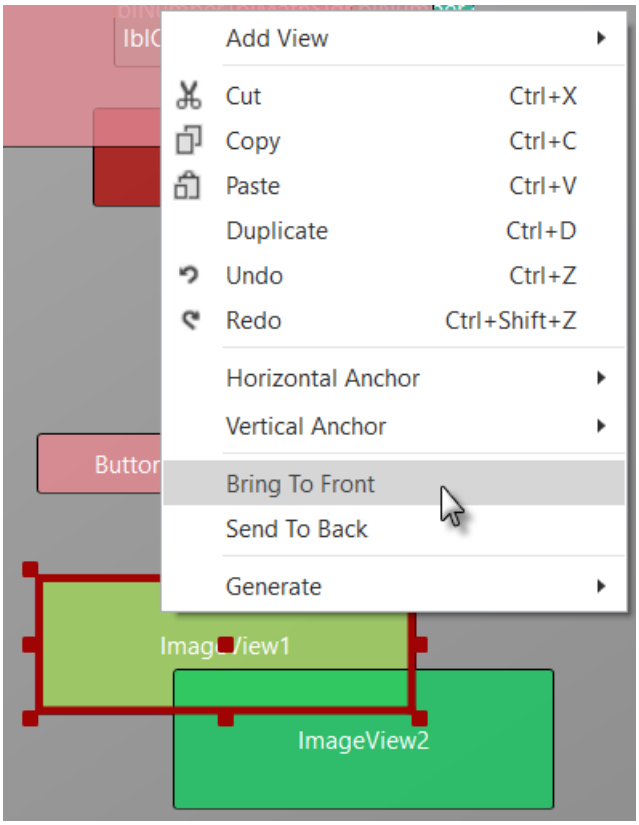
More details in the Anchors chapter.

5.8.3.6 Bring To Front

Bring To Front Moves the selected node on top of the layout.

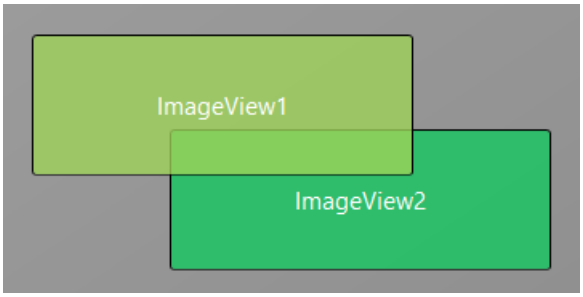


In the picture, ImageView2 is over ImageView1. You see it with the border color.



Right click on ImageView1 and click on Bring To Front to move ImageView1 to front of all other nodes.

And the result:



5.8.3.7 Send To Back

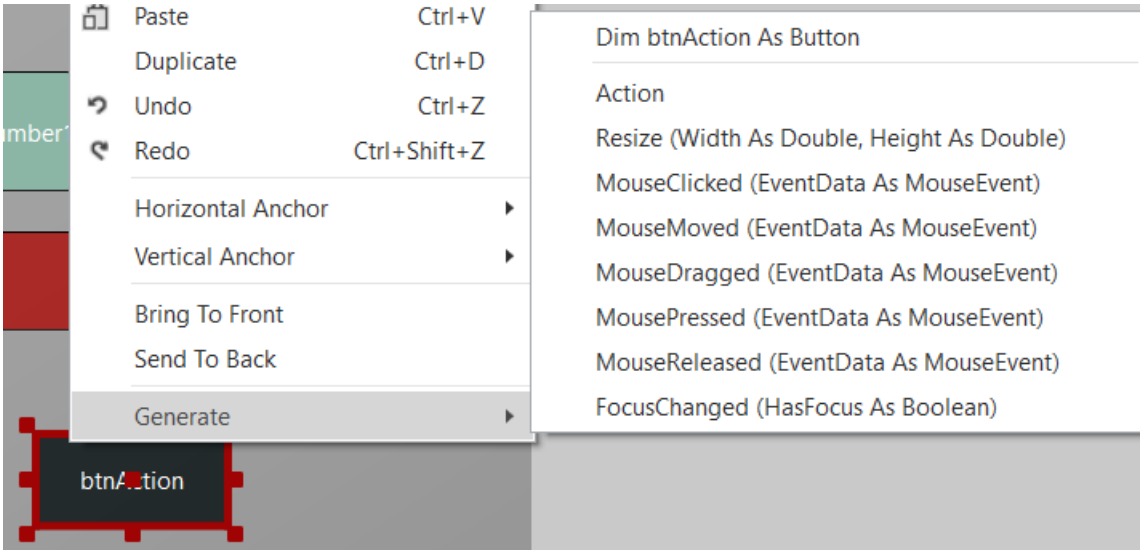
Send To Back Is the inverse function of the *Bring To Front* function above.

5.8.3.8 Generate

Generates the declaration statement or an event routine frame for the selected node. It is a shortcut of the Generate Members function in the VisualDesigner Tools menu but only for the selected node.

A popup menu allows you to select what code you want to generate, the possibilities depend on the type of the selected node.

Example with a Button:



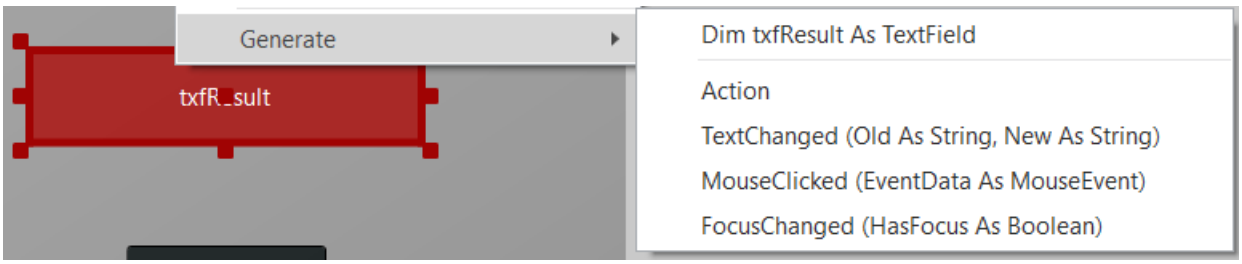
Dim btnAction As Button
Generates the declaration statement in the Globals routine.

```
Private btnAction As Button
```

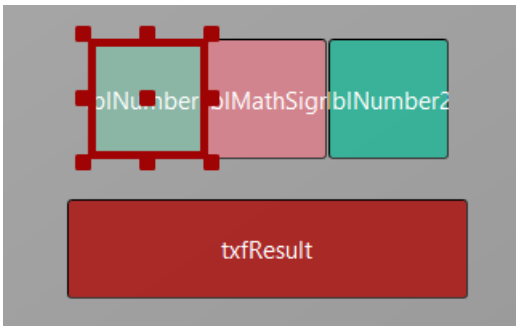
MouseClicked (EventData As MouseEvent)
Generates the MouseClicked event routine frame.

```
Sub btnAction_MouseClicked (EventData As MouseEvent)

End Sub
```

Example with a TextField node:

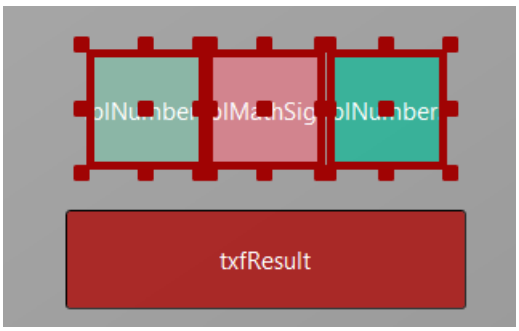


5.8.4 Select nodes



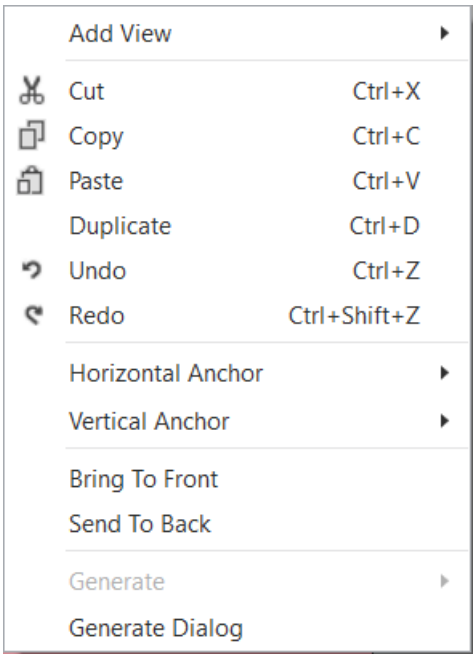
Select a single node:
Click on the node.

The node is highlighted.



Select several nodes:
Click on the first node.
Press the Ctrl key,
Select the following nodes.

The selected nodes are highlighted.

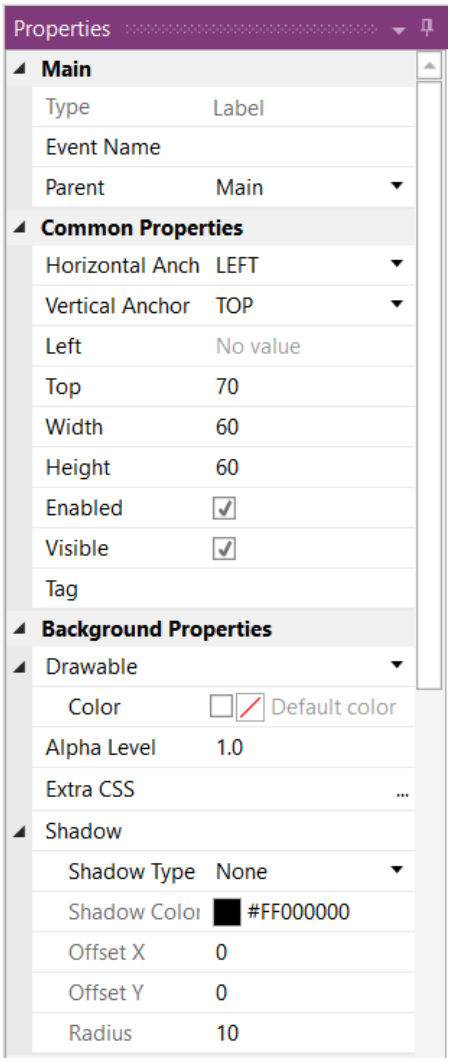


After the selection you can:

- Move the selected nodes with the mouse or with the arrow keys of the keyboard in the four directions.
- Right click on one of the selected nodes to show the context menu.

The functions are the same as for a single node, but a new function, GenerateDialog, is available to Generate Members. This is the same function as in the Visual Designer Tools menu.

- In the Properties window you can change all properties common to the selected nodes.

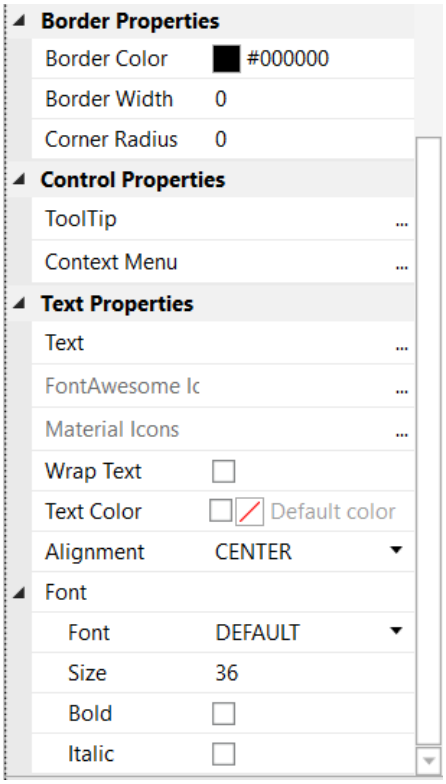


You can change the parent node.

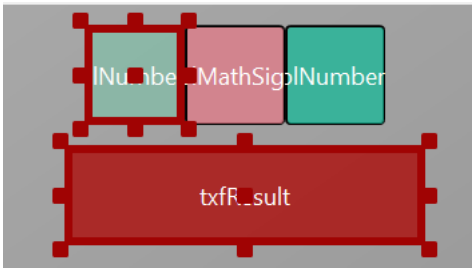
All properties with No value cannot be changed.

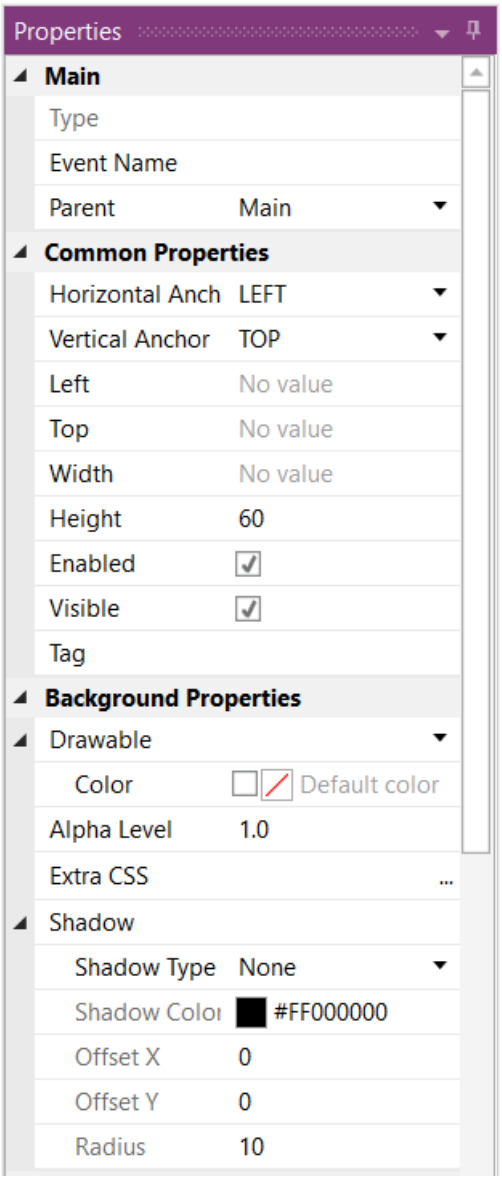
You can change all properties with a value because they are the same for all the selected nodes.

Changing, for example, the Height property will change it for all the selected nodes.



If you select nodes of different types, only the properties common to the selected nodes can be changed.





The Left, Top and Width properties cannot be changed because they are different for the selected nodes.

The Height property can be changed because, in this particular case, its value is the same for both nodes, even for nodes of different types.

5.9 Adding nodes by code

It is also possible to add nodes by code instead of using the Designer.

- Advantage: None.
- Disadvantage: You have to define almost everything.

Note that you should avoid adding node**s in code but use the Designer with Designer Scripts.**

The source code is in the SourceCode directory: AddViewsByCode

In B4J there are no dip, %x nor %y values, only pixel values.

If you want use %x and %y values, you must do it in the MainForm_Resize event routine.
The MainForm_Resize event is also raised when the program starts, just after AppStart.

Example:

Let us put a Label on top of the form, a Pane below it with a Label on it and a Pane at the bottom of the form with three Buttons.

Code explanations:

Declaring the nodes in Process_Globals.

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form

   Private lblTitle, lblPanelTitle As Label
   Private pnlTest, pnlTools As Pane
   Private btnCut, btnCopy, btnPaste As Button
End Sub
```

Initializing and adding the different nodes in AppStart:

`MainForm.Title = "Add nodes by code"` Sets the Form title
`MainForm.Resizable = True` Sets the Form being resizable.
`MainForm.SetWindowSizeLimits(310, 300, 1200, 800)` Sets the min and max form sizes.

`lblTitle.Initialize("")` Initializes the Label, no EventName required.
`CSSUtils.SetBackgroundColor(lblTitle, fx.Colors.Red)` Sets the Background color to red.
`lblTitle.TextSize = 20` Sets the text size to 20.
`lblTitle.TextColor = fx.Colors.White` Sets the text color to white.
`lblTitle.Alignment = "CENTER"` Sets the label text alignment to ‘CENTER’.
`lblTitle.Text = "Title"` Sets the label text to 'Title'.

If the Label had been added in the Designer, all the above code wouldn't have been necessary because the properties would already have been defined in the Designer.

`pnlTest.Initialize("")` Initializes the Panel, no EventName required.
`CSSUtils.SetBackgroundColor(pnlTest, fx.Colors. RGB(240, 255, 240))`
Sets the Background color to white.
`CSSUtils.SetBorder(pnlTest, 1, fx.Colors.Black, 0)`
Sets the Border color to black and the corner radius to 0.

The rest of the initialization code is similar to the code above.

We add the nodes. In B4J Views are called Nodes, therefore AddNode instead of AddView.
We add `lblTitle`, `pnlTest` and `pnlTools` to the `MainForm.RootPane`.

```
MainForm.RootPane.AddNode(lblTitle, 0.1 * 600, 10, 0.8 * 400, 30)
MainForm.RootPane.AddNode(pnlTest, 0.1 * 600, lblTitle.Top + lblTitle.Height + 10, 0.8
* 600, 0.8 * 400)
```

And we add `lblPanelTitle` to `pnlTest and the buttons to pnlTools`.

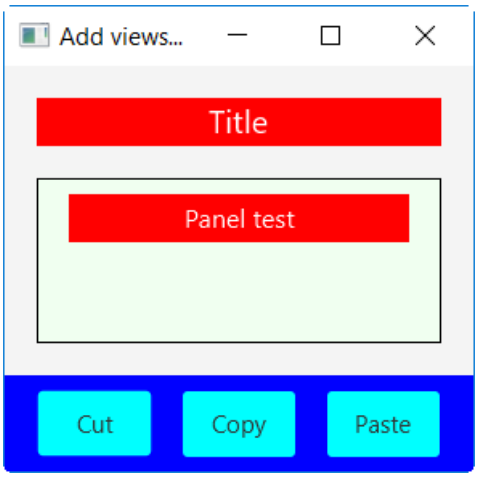
```
MainForm.RootPane.AddNode(lblTitle, 20, 20, 400, 30)
MainForm.RootPane.AddNode(pnlTest, 20, 20, 50, 50)
MainForm.RootPane.AddNode(pnlTools, 10, 10, 50, 60)
pnlTest.AddNode(lblPanelTitle, 20, 10, 100, 30)
pnlTools.AddNode(btnCut, 50, 10, 70, 40)
pnlTools.AddNode(btnCopy, 50, 10, 70, 40)
pnlTools.AddNode(btnPaste, 50, 10, 70, 40)
```

In the `Private Sub MainForm_Resize` routine we resize the nodes.

```
Private Sub MainForm_Resize (Width As Double, Height As Double)
   lblTitle.Left = 20
   lblTitle.SetSize(Width - 40, 30)

   pnlTools.Left = 0
   pnlTools.SetSize(Width, 60)
   pnlTools.Top = Height - pnlTools.Height

   pnlTest.Left = lblTitle.Left
   pnlTest.Top = lblTitle.Top + lblTitle.Height + 20
   pnlTest.SetSize(Width - 40, pnlTools.Top - lblTitle.Height - 60)

   lblPanelTitle.Left = 20
   lblPanelTitle.SetSize(Width - 80, 30)

   btnCopy.Left = (Width - btnCopy.Width) / 2
   btnCut.Left = btnCopy.Left - btnCut.Width - 20
   btnPaste.Left = btnCopy.Left + btnCut.Width + 20
End Sub
```

And the result:

The positions and sizes are adjusted to the form size.



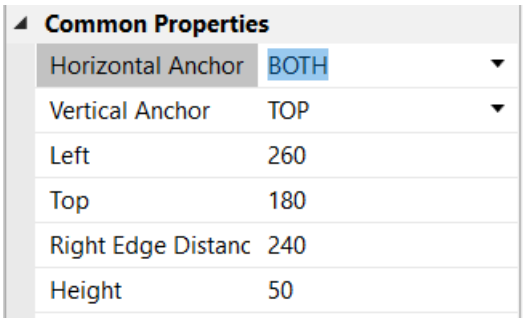
Or resized to the minimum size.

The same can be better done in the Designer with anchors and Designer Scripts !
See next chapter.

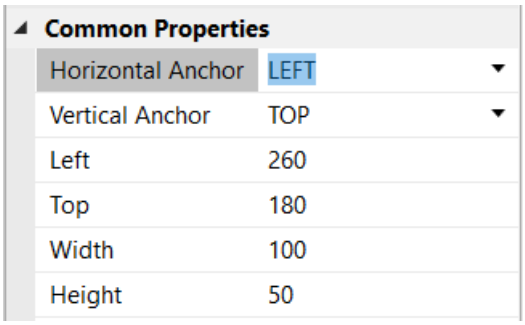
5.10 Anchors

The Designer has two 'special' features to size nodes, the Horizontal Anchor and the Vertical Anchor.

Horizontal Anchor

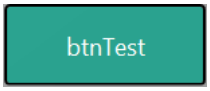


The horizontal anchor property can take three values:

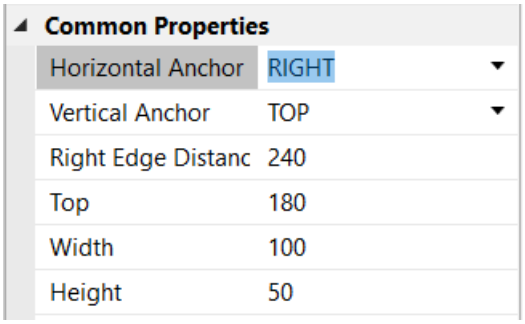


- LEFT

LEFT is the default value.
The left edge is anchored to the left edge of the parent node with the distance given in the Left property.



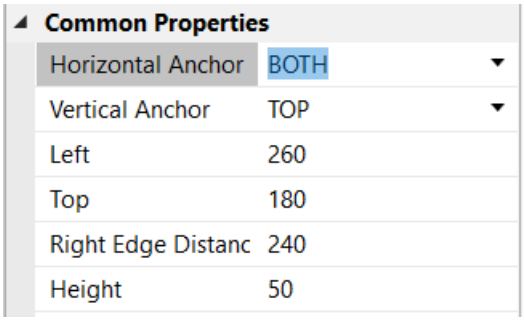
No anchor symbol is shown.



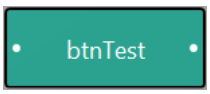
- RIGHT

The right edge is anchored to the right edge of the parent node with the distance given in the Right Edge Distance property.
The Left property is no more available because it is defined by the width and the right anchor!

The dot on the right edge shows the anchor.



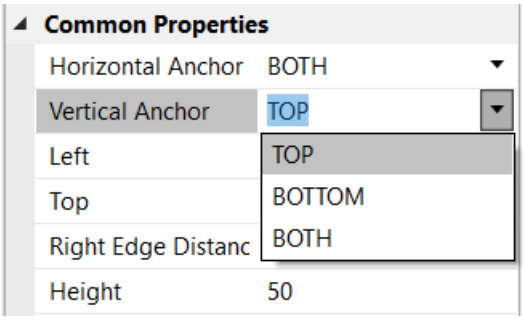
- BOTH



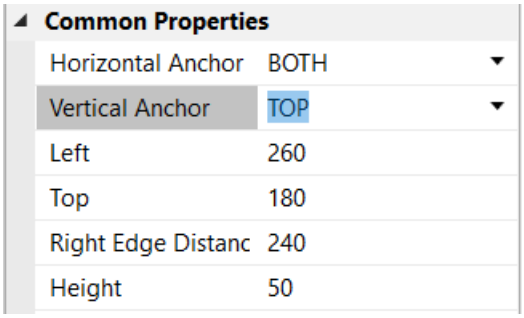
Both edges are anchored to the parent node with distances defined in the Left and Right Edge Distance properties.
The Width property is no more available because it is defined by the anchors!
The dots on the two edges show the anchors.

Setting the Horizontal Anchor property to BOTH is similar to the SetLeftAndRight function in the Designer Scripts.

Vertical Anchor

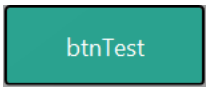


The vertical anchor property can take three values:

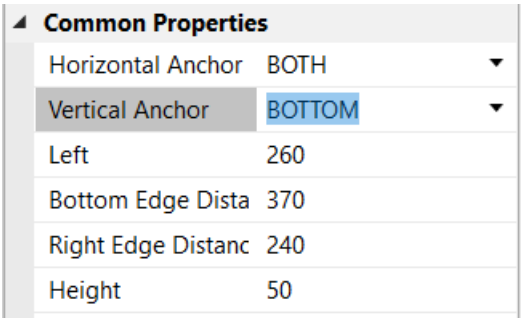


- TOP

TOP is the default value.
The top edge is anchored to the top edge of the parent node with the distance given in the Top property.



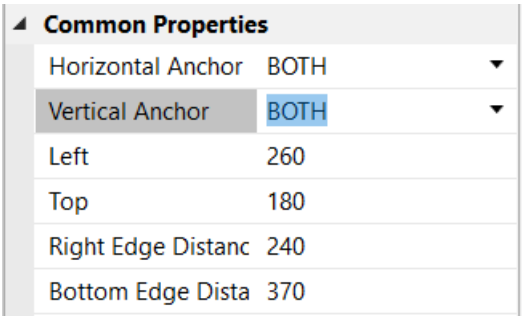
No anchor symbol is shown.



- BOTTOM

The bottom edge is anchored to the bottom edge of the parent node with the distance given in the Bottom Edge Distance property.
The Top property is no more available because it is defined by the Height and the bottom anchor!

The dot on the bottom edge shows the anchor.



- BOTH

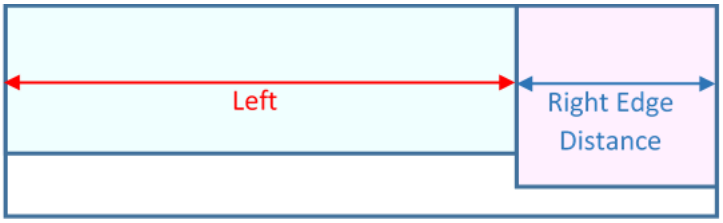
Both edges are anchored to the parent node with distances defined in the Top and Bottom Edge Distance properties.
The Height property is no more available because it is defined by the anchors!

The dots on the two edges show the anchors.

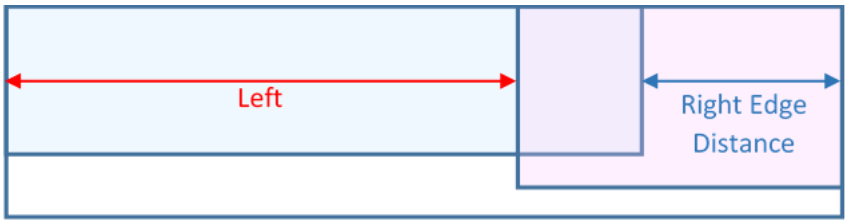
Setting the Vertical Anchor property to BOTH is similar to the SetTopAndBottom function in the Designer Scripts.

What happens when we set the horizontal anchor of the two nodes below to BOTH and change the parent node width?

The left nodes right edge is anchored to the right edge of the parent node with the Right Edge Distance.
The right nodes left edge is anchored to the left edge of the parent node with the Left distance.



If we increase the width of the parent node we get the layout below.



The left nodes right edge is still at the Right Edge Distance from the parent nodes right edge.
The right nodes left edge is still at the Left distance from the parent nodes left edge.
The result is an overlapping of both nodes.

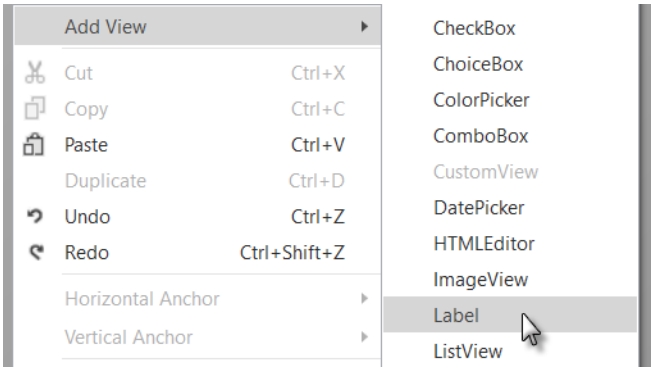
In this case you must adjust the nodes in the Designer Scripts with the `SetLeftAndRight` method!

For example:
```
LeftView.SetLeftAndRight(0, 67%x)
RightView.SetLeftAndRight(33%x, 100%x)
```

5.10.1 First example program

The example shown in this chapter is the *Anchors1* project in the *Anchors* folder.
The layout variant is 400 x 600.

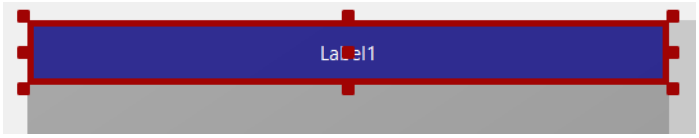
First, we add a label on top of the screen which should cover the whole width and stay on top.



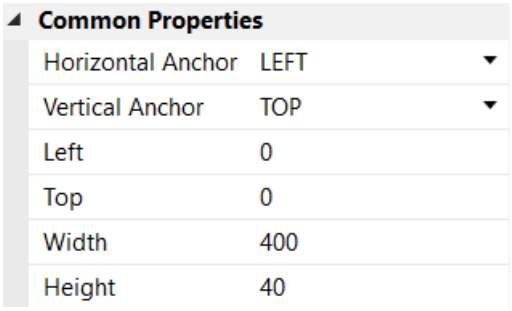
In the AbstractDesigner right click somewhere on the screen, the menu on the left will be displayed:

Click on Add View.

Click on Label.

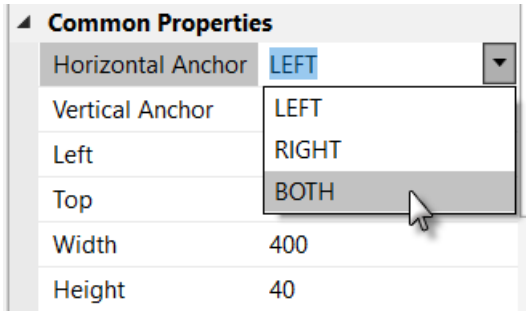


Move the labels upper left corner to the upper left corner of the screen and stretch it to fill the whole width of the screen.



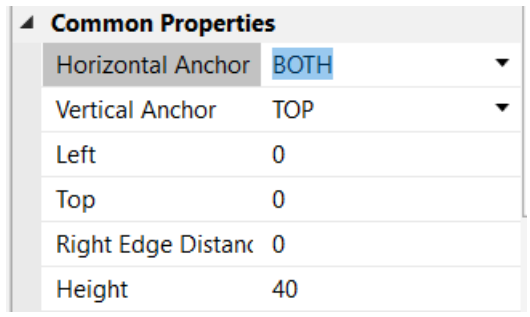
We see these properties:

Left = 0
Top = 0
Width = 400 full layout width
Height = 40 Set the height to 40

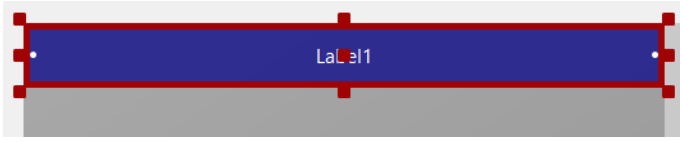


Now we change le 'Horizontal Anchor' property:

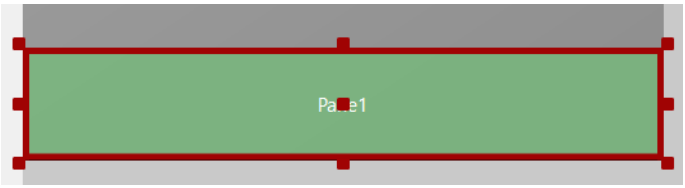
Click on BOTH.



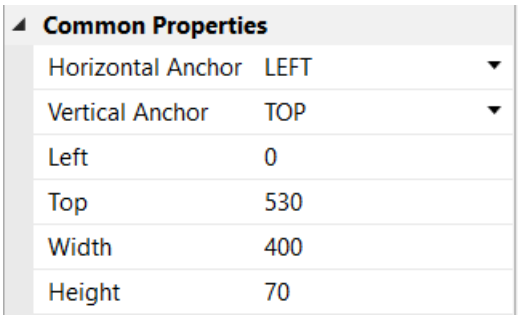
We see that the properties changed:
Left, Top and Height are still the same.
But Width has disappeared and is replaced by
Right Edge Distance = 0
Its value = 0 because the right edge is on the right edge of the screen.



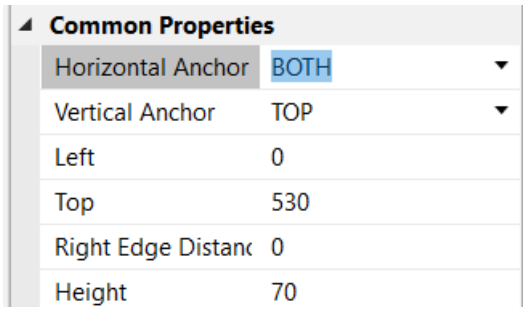
Now we see the anchor dots on the left and right edges.



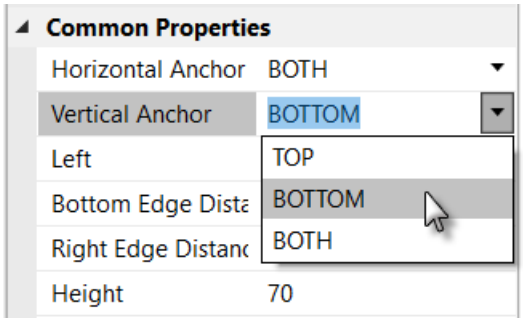
Now, we add a Pane at the bottom of the screen covering also the whole screen width.



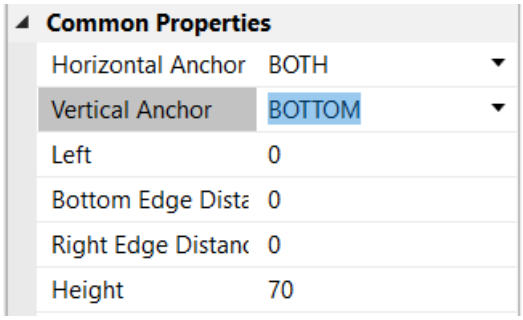
The properties should look like in the picture.



We set the Horizontal Anchor to BOTH.
Same as for Label1.

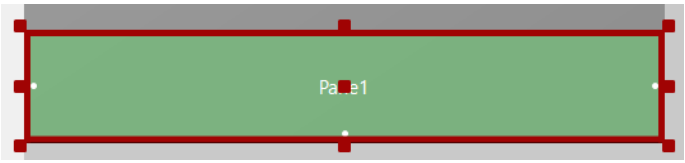


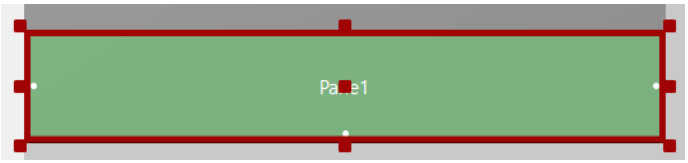
We set the Vertical Anchor to BOTTOM.



The Top property is replaced by the:
Bottom Edge Distance = 0 property.
Its value = 0 because we anchor the bottom edge of Panel1 to the screens bottom edge.

We see the anchor dots on the left, right and bottom edges.

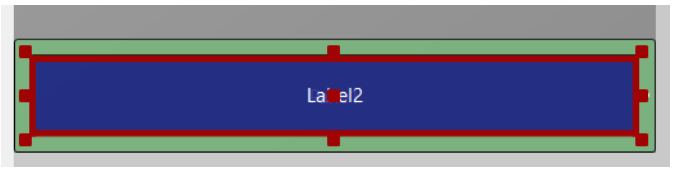




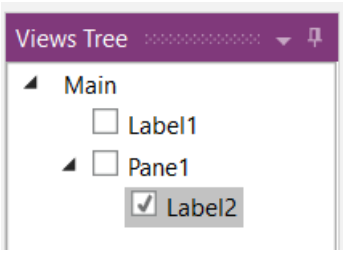
Now we add a second label onto Panel1.

Click on Panel1 to select it.

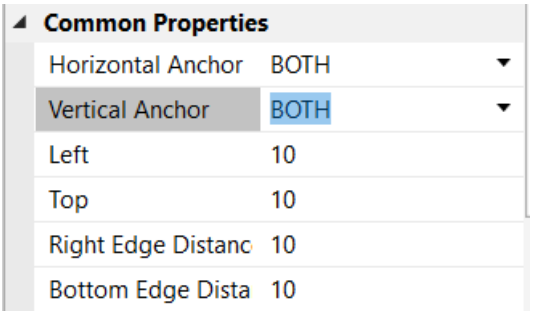
Add the label.



Move and size the label like in the picture with the Left, Top, Width and Height properties like in the list below.

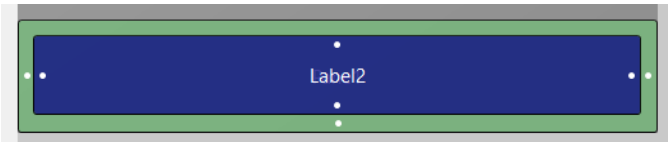


In the Views Tree window, we see that Label2 is shifted to the right because its parent node is Panel1 and not Main like for Label1 and Pane1!



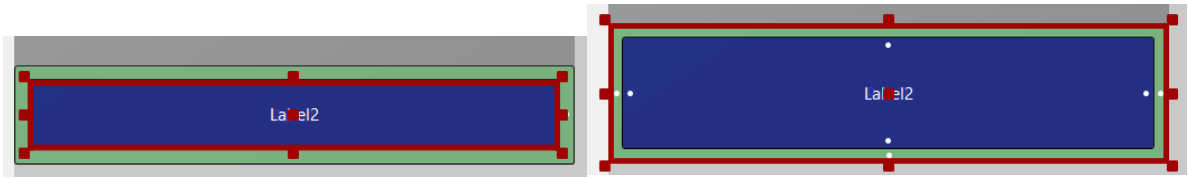
We set the two Anchors to BOTH.
The properties
Left = 10 and
Top = 10 remain the same.
Right Edge Distance = 10 and
Bottom Edge Distance = 10
The two values are equal to 10 because we want a 'frame' around Label2.

We see the anchor dots on all four edges.



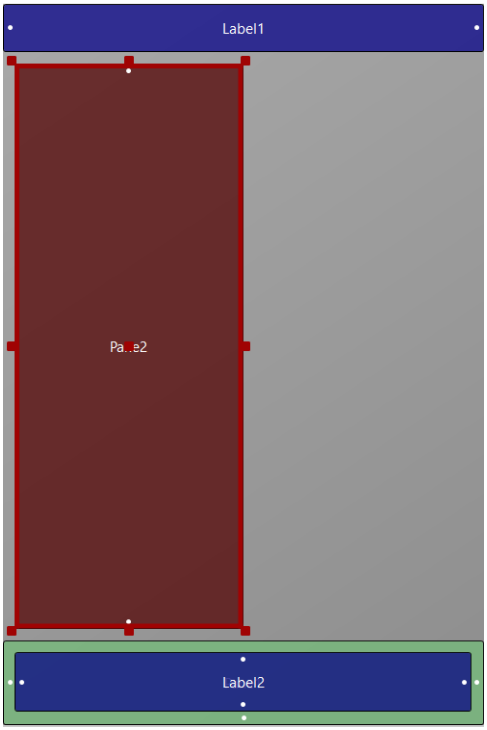
And the result looks like the pictures below in portrait and landscape screen orientations.

To demonstrate the anchor feature we move, in the Abstract Designer, the top edge of Pane1 upwards.

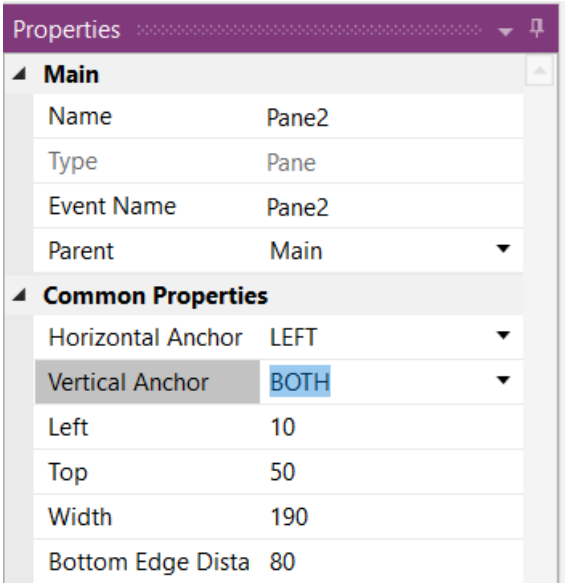


We see that the top edge of Label2 moves with the top edge of Pane1 and the bottom edge remains at its place!

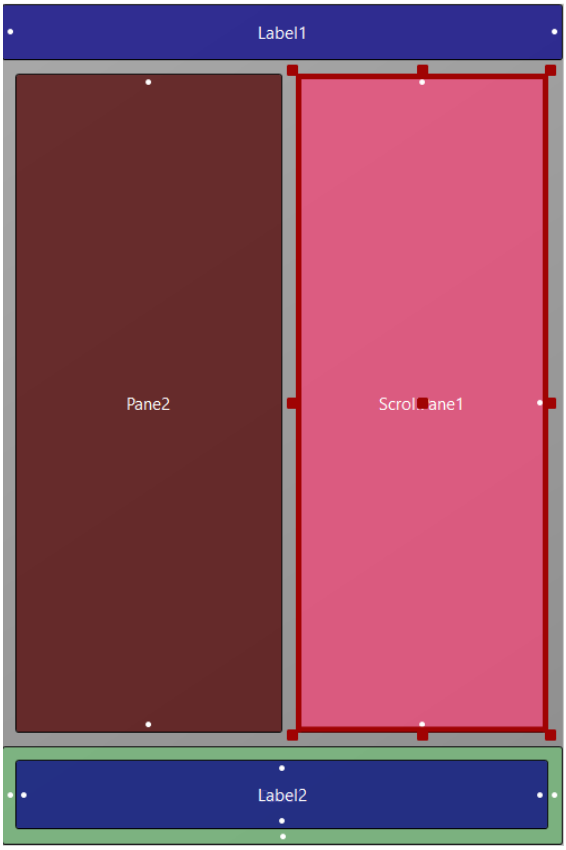
Then move it back to a height of 70.



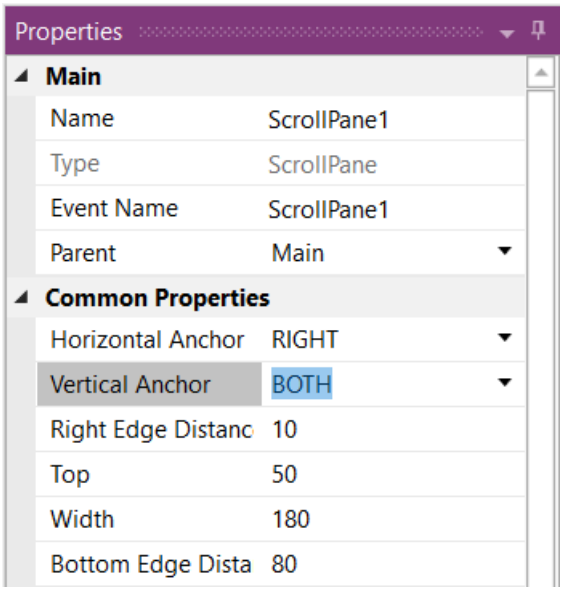
Now, we add another Pane onto the left half of the screen and vertically positioned between Label1 and Pane1 leaving a small space.



We set the vertical anchor to BOTH.

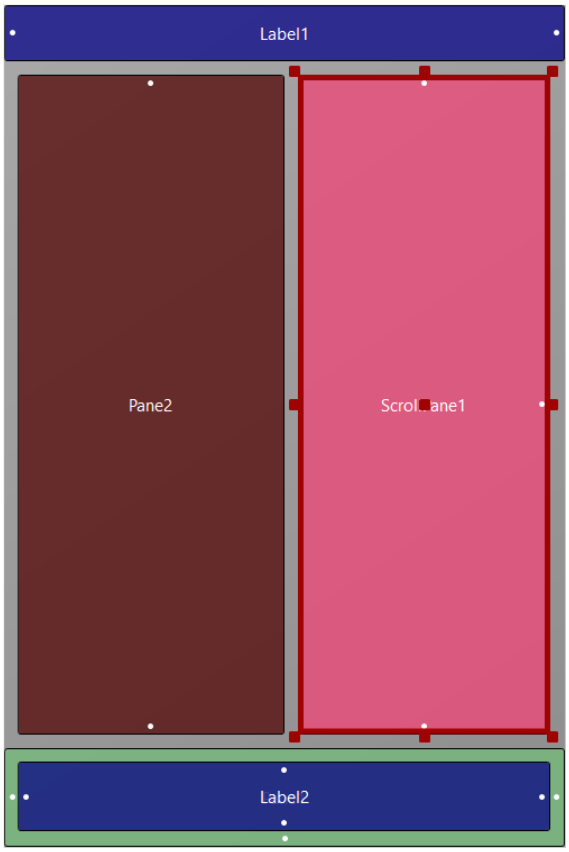


Now, we add a ScrollPane on the right half of the screen also positioned between Label1 and Pane1 leaving a small space.



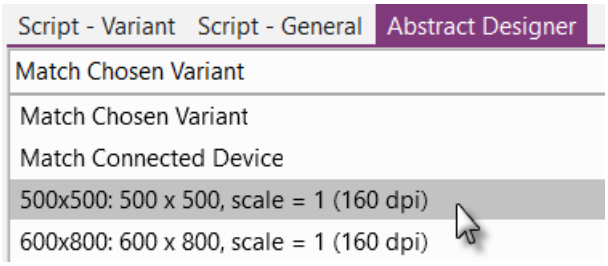
We set the horizontal anchor to RIGHT.
We set the vertical anchor to BOTH.

And the result:

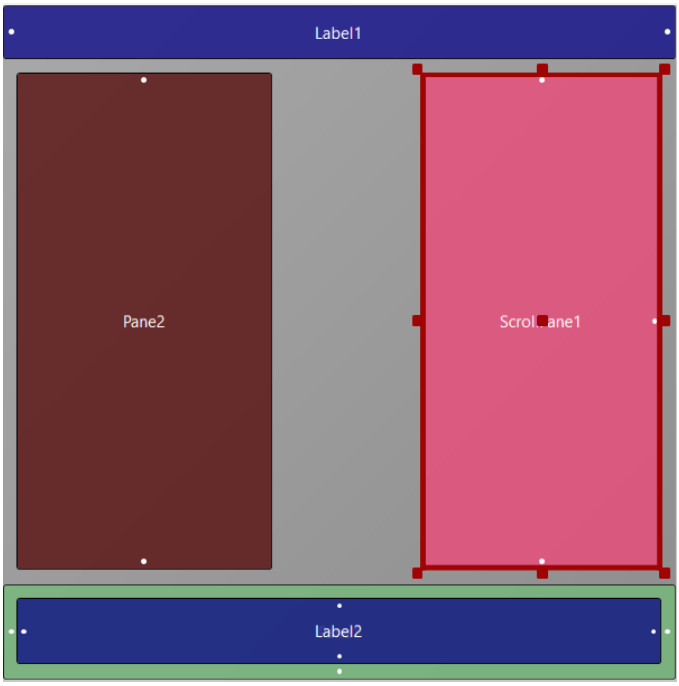


We see that the anchors work fine.
But, we see that there is a big gap between Pane2 and the ScrollPane1.

If we select another layout variant (500 x 500).



We get the result below.



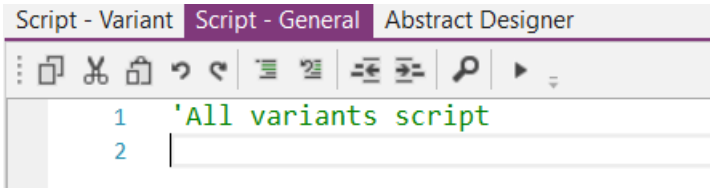
Why do we have this gap?

Because we set the Horizontal Anchor of the Panel1 to LEFT
and the Horizontal Anchor of the ScrollPane1 to RIGHT.

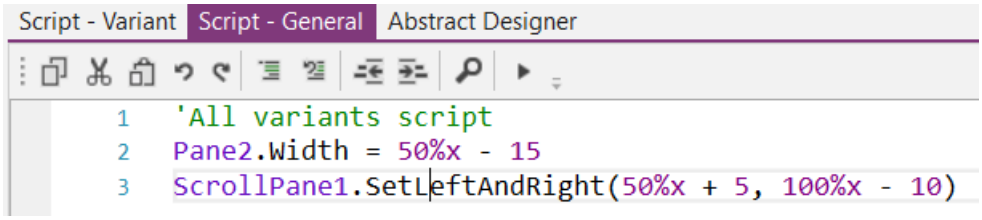
But the Width properties remains the same and that's why we get the gap between the two nodes when the screen width is wider than the layout screen width.

To adjust the width, we add two lines in the DesignerScripts.

Click on Designer Scripts to show the DesignerScripts window.

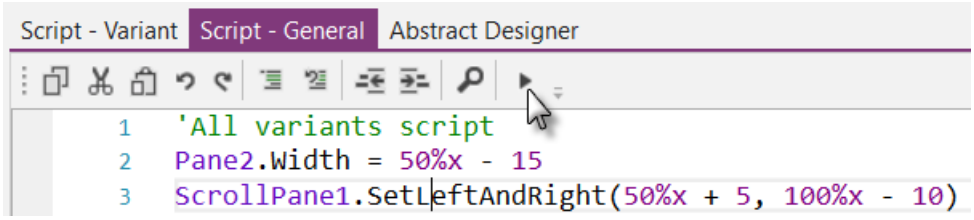


Here we add the following two lines:



The anchors are valid in the AbstractDesigner but not in Designer Scripts.

For `Pane2` it's enough to set its Width property.
But for `ScrollPane1` we need to define both properties `Left` and `Right` which is done with `SetLeftAndRight` because the RIGHT anchor is lost.



In the Script-General window click on ▶ to refresh the Abstract Designer.

And the result in the Abstract Designer.

I added some flashy colors to show the nodes when you run the program.
I find that showing how to change the default colors is out of the scope of this example.

5.10.2 Second example program

This program shows exactly the same form and layout as the AddViewsInCode program.
But the layout is defined in the Designer with the Abstract Designer, anchors and Designer Scripts.
Source code, Anchors2, in the Anchors subfolder.

In the layout, we have a Label on top of the form, a Pane below it with a Label on it and a Pane at the bottom of the form with three Buttons.

The top Label, lblTitle has the horizontal anchor set to BOTH.
The center Pane, pnlTest has the horizontal anchor and the vertical anchor set to BOTH.
The Label, lblPanelTitle, in pnlTest has the horizontal anchor set to BOTH.
The bottom Pane has the horizontal anchor set to BOTH and the vertical anchor set to BOTTOM.
The Buttons have no anchors, they are positioned in the Designer Scripts.

Code in the Designer Scripts:

```
'All variants script
btnCopy.HorizontalCenter = 50%x
btnCut.Right = btnCopy.Left - 10
btnPaste.Left = btnCopy.Right + 10
```

`btnCopy.HorizontalCenter = 50%x`
Centers `btnCopy` in the middle of the form (`50%x`).

`btnCut.Right = btnCopy.Left - 10`
Sets the Right property of `btnCut` to the Left property of `btnCopy` minus 10 pixels.

`btnPaste.Left = btnCopy.Right + 10`
Sets the Left property of `btnPaste` to the Right property of `btnCopy` plus 10 pixels.

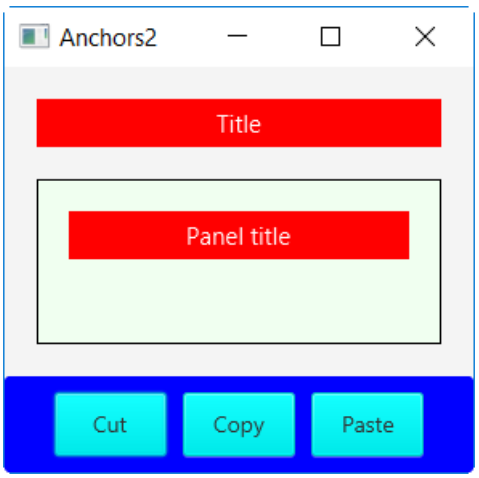
The Designer Script accepts %x and %y values, which are not supported in the IDE.

Code in the IDE:

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form
End Sub

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

   'limits the form size
   MainForm.SetWindowSizeLimits(310, 300, 1200, 800)

   MainForm.Show
End Sub
```

Nothing added in **`Process_Globals`**.

In **`AppStart`** we add this line to set the size limits of the form.
SetWindowSizeLimits(MinWidth As Double, MinHeight As Double, MaxWidth As Double, MaxHeight As Double)
`MainForm.SetWindowSizeLimits(310, 300, 1200, 800)`



If you resize the form, the anchors are considered and the code in Designer Scripts is executed.

The form with its minimum size.

5.11 Designer Scripts

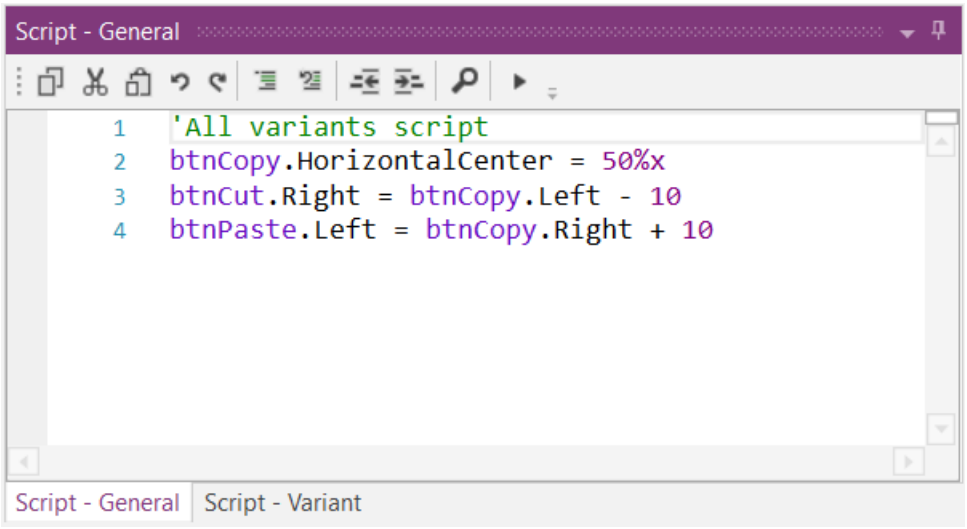
The "Designer Scripts" tool will help you fine tune your layout and easily adjust it to different screens.

It is not recommended to create many layout variants in the Designer.

The idea is to combine the usefulness of the visual designer with the flexibility and power of programming code.

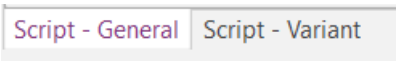
Layout variants are less important in B4J than in B4A and B4i because you don't have to handle different device dimensions.
Therefore, AutoScale doesn't exist in B4J!

You can write a simple script to adjust the layout based on the dimensions of the current device and immediately see the results. You can immediately see the results on the Abstract Designer. This allows you to test your layout on different screen sizes.



5.11.1 General

Every layout file can include script code. The script is written inside the Visual Designer in the Script windows:



There are two types of scripts:
- Script – General, the general script that will be applied to all variants.
- Script – Variant, specific code can be written for each variant.

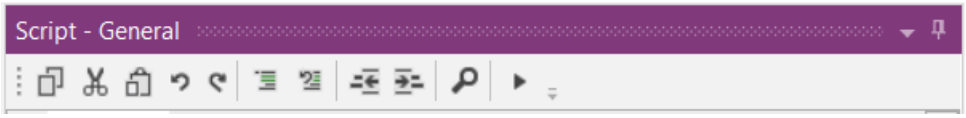
Once you press, in the Script window menu, on the Run Script button  (or F5), the script is executed and the connected device / emulator and abstract designer will show the updated layout.

The same thing happens when you run your compiled program. The (now compiled) script is executed after the layout is loaded.

The general script is first executed followed by the variant specific script.

The script language is very simple and is optimized for managing the layout.

5.11.2 The menu



	Ctrl + C	Copy
	Ctrl + X	Cut
	Ctrl + V	Paste
	Ctrl + Z	Undo
	Ctrl + Shift + Z	Redo
	Ctrl + Q	Block Comment
	Ctrl + W	Block Uncomment
		Outdent
		Indent
	F3	Find / Replace
	F5	Run

5.11.3 Supported Properties

The following properties are supported:
- **`Left / Right / Top / Bottom / HorizontalCenter / VerticalCenter`**
  Gets or sets the node 's position. The node 's width or height will not be changed.

- **`Width / Height`** - Gets or Sets the node 's width or height.

- **`TextSize`** - Gets or sets the text size.

- **`Text`** - Gets or sets the node 's text. TextSize and Text properties are only available to node s that show text.

- **`Visible`** - Gets or sets the node 's visible property.

5.11.4 Supported Methods

- **`SetLeftAndRight`** (Left, Right) - Sets the node 's left and right properties. This method changes the width of the node based on the two values.

- **`SetTopAndBottom`** (Top, Bottom) - Sets the node 's top and bottom properties. This method changes the height of the node based on the two values.

5.11.5 Supported Keywords

- **`And / Or`** - Same as the standard And / Or keywords.

- **`False / True`** - Same as the standard False / True keywords.

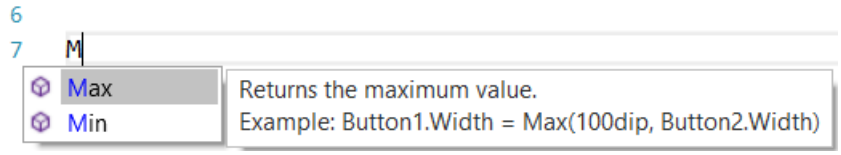
- **`Min / Max`** - Same as the standard Min / Max keywords.

- **`ActivitySize`** - Returns the approximate form size measured in inches.

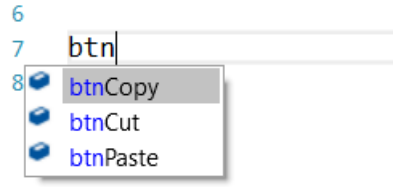
- **`If . Else If . Else .`** condition blocks - Both single line and multiline statements are supported. The syntax is the same as the regular If blocks.

5.11.6 Autocomplete

When you begin writing the Autocomplete function shows all possible keywords or node names containing the written text with the help of the selected keyword.
Example: M, shows keywords containing *m*.



Example: bt, shows all buttons.



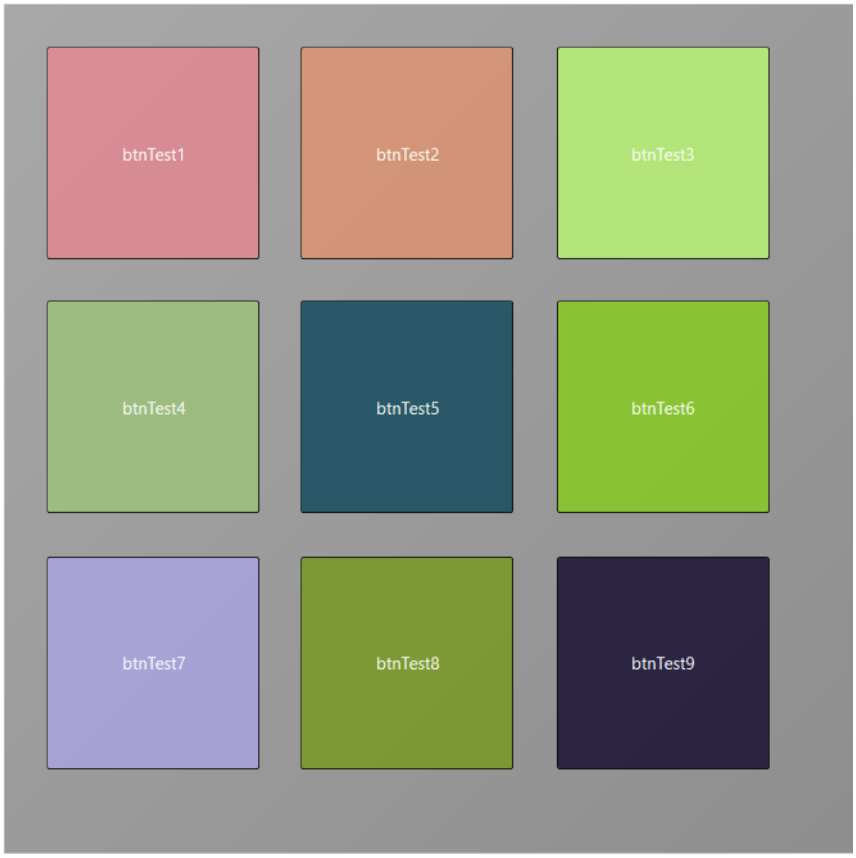
5.11.7 Notes and tips

- %x and %y values are relative to the node that loads the layout.
  Usually it will be the Form. However, if you use Pane.LoadLayout then the values are relative to the size of this pane.

- Variables - You can use variables in the script. You do not need to declare the variables before using them (there is no Dim keyword in the script).

5.11.8 Example

In this example, we build the following layout with 9 square Buttons:
The source code is in the *Designer\Scripts* folder.



btnTest5 should be centered in the middle of the screen.
A space of 30 pixels should be between the form edges of the smallest form side and between the buttons. The button width should be increased with the screen size.

The first step is to add the node s and position them with the visual designer (you do not need to be accurate).
Now we will select the designer scripts tab and add the code.
Note that the node s are locked when the designer scripts tab is selected.

The code in the IDE:

We load the layout and set the form size limits.

```
Sub AppStart (Form1 As Form, Args() As String)
   MainForm = Form1
   MainForm.RootPane.LoadLayout("Main") 'Load the layout file.
   MainForm.SetWindowSizeLimits(400, 400, 800, 800) 'Set the form size limits
   MainForm.Show
End Sub
```

In the Designer Scripts we calculate two variables: Space and Width.

```
'calculate the variables Scale, Space, Width
Space = 30
If 100%x < 100%y Then
     Width = (100%x - 4 * Space) / 3
Else
     Width = (100%y - 4 * Space) / 3
End If
TextSize = btnTest1.TextSize * Scale
```

This could also have been calculated this way:

```
'calculate the variables Scale, Space, Width
Space = 30
Width = (Min(100%x, 100%y) - 4 * Space) / 3
TextSize = btnTest1.TextSize * Scale
```

Set the Width and Height properties of the buttons:

```
'set width and height
btnTest1.Width = Width
btnTest1.Height = Width
btnTest2.Width = Width
btnTest2.Height = Width
btnTest3.Width = Width
btnTest3.Height = Width
btnTest4.Width = Width
btnTest4.Height = Width
btnTest5.Width = Width
btnTest5.Height = Width
btnTest6.Width = Width
btnTest6.Height = Width
btnTest7.Width = Width
btnTest7.Height = Width
btnTest8.Width = Width
btnTest8.Height = Width
btnTest9.Width = Width
btnTest9.Height = Width
```

Set the TextSize properties of the buttons:

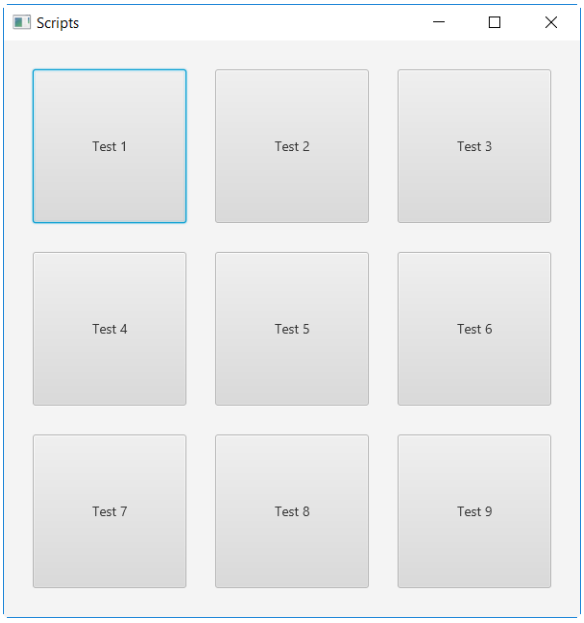
```
'set the TextSize property
btnTest1.TextSize = TextSize
btnTest2.TextSize = TextSize
btnTest3.TextSize = TextSize
btnTest4.TextSize = TextSize
btnTest5.TextSize = TextSize
btnTest6.TextSize = TextSize
btnTest7.TextSize = TextSize
btnTest8.TextSize = TextSize
btnTest9.TextSize = TextSize
```

We center button btnTest5

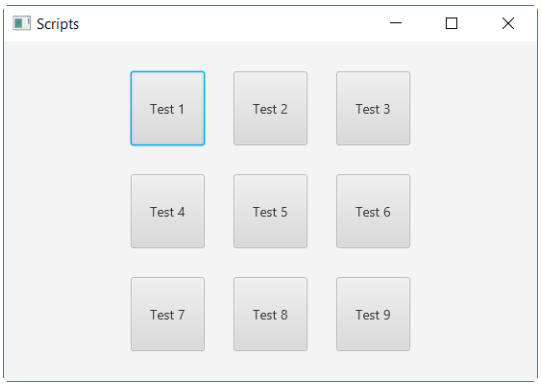
```
'position btnTest5 in the screen middle
btnTest5.HorizontalCenter = 50%x
btnTest5.VerticalCenter = 50%y
```

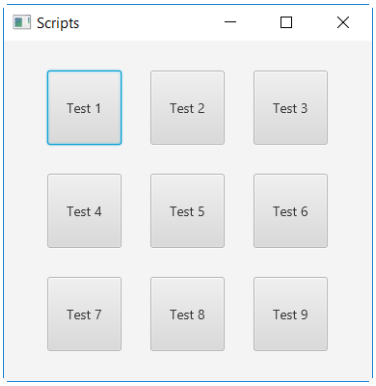
We position the other buttons:

```
'position the other buttons acording to btnTest5
btnTest1.Right = btnTest5.Left - Space
btnTest1.Bottom = btnTest5.Top - Space

btnTest2.Left = btnTest5.Left
btnTest2.Bottom = btnTest5.Top - Space

btnTest3.Left = btnTest5.Right + Space
btnTest3.Bottom = btnTest5.Top - Space

btnTest4.Right = btnTest5.Left - Space
btnTest4.Top = btnTest5.Top

btnTest6.Left = btnTest5.Right + Space
btnTest6.Top = btnTest5.Top

btnTest7.Right = btnTest5.Left - Space
btnTest7.Top = btnTest5.Bottom + Space

btnTest8.Left = btnTest5.Left
btnTest8.Top = btnTest5.Bottom + Space

btnTest9.Left = btnTest5.Right + Space
btnTest9.Top = btnTest5.Bottom + Space
```



The result, the images have been reduced in size:



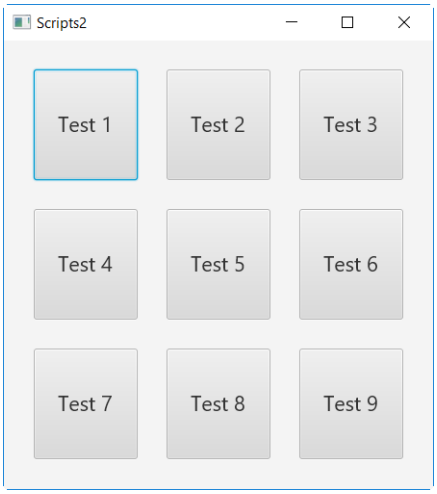


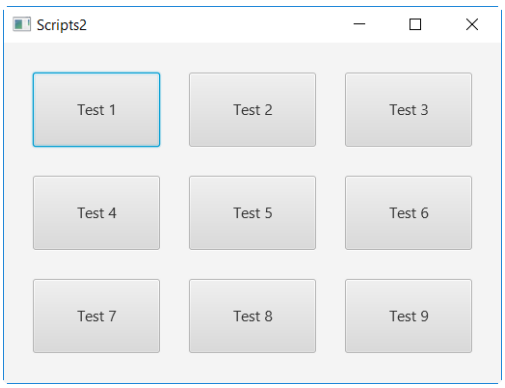
We could improve a bit the layout.
Have the Buttons filling the whole form, they can be rectangular and change the text size per the smallest value of the buttons width or height. The project is Scripts2.

The Script code:

```
'calculâtes the variables Width, Height, TextSize
Space = 30
Width = (100%x - 4 * Space) / 3
Height = (100%y - 4 * Space) / 3
TextSize = Min(Width, Height) / 5

'set width and height
btnTest1.Width = Width
btnTest1.Height = Height
btnTest1.TextSize = TextSize
btnTest2.Width = Width
btnTest2.Height = Height
btnTest2.TextSize = TextSize
btnTest3.Width = Width
btnTest3.Height = Height
btnTest3.TextSize = TextSize
btnTest4.Width = Width
btnTest4.Height = Height
btnTest4.TextSize = TextSize
btnTest5.Width = Width
btnTest5.Height = Height
btnTest5.TextSize = TextSize
btnTest6.Width = Width
btnTest6.Height = Height
btnTest6.TextSize = TextSize
btnTest7.Width = Width
btnTest7.Height = Height
btnTest7.TextSize = TextSize
btnTest8.Width = Width
btnTest8.Height = Height
btnTest8.TextSize = TextSize
btnTest9.Width = Width
btnTest9.Height = Height
btnTest9.TextSize = TextSize
```

The button positioning remains the same.





6 Process life cycle

Each B4J program runs in its own process. A process starts when the user launches your application.

A B4J application is made of one or more Forms.

Forms are Windows Forms and are somewhat similar to B4A Activities or B4i Pages.

6.1 How do we handle it ?

The life cycle of Java applications is quite simple. The two most important event is AppStart.

The standard way to start an application is by clicking on its icon.
This will cause **AppStart** to run. AppStart only runs once when the process starts.
It is always the first sub to run.
You will usually load the layout in this sub and optionally restore the state.
Note that at this point the actual page size is not known.

One more important event is **MainForm_Resize**, this event is raised just after AppStart.
In this event the current size is known allowing to adjust node dimensions if needed and initialize Canvases. The skeleton of this routine is not included in the default template you must add it yourself if you need it. This event will also be raised when the user resizes the Form.

The process will continue running until the user closes the application. This happens when the user presses on the close button of the form  .

When you start the IDE you will see the default template below.

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

Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form
End Sub

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

The jFX library is need in B4J and load automatically to each project.
And initialized in the default project template: `Private fx As JFX`.
A private global variable `MainForm` , is defined, holding a reference to the main form.

6.2 Process global variables

Variables can be either `Public` or `Private`.
Variables declared in Process_Globals as `Private` accessible only in this module whereas variables declared as `Public` can be accessed from everywhere in the project.
`Private` variables are local to the containing sub. Once the sub ends, these variables no longer exist.
`Public` variables can be accessed from all subs. The MainForm_Resize event routine is not part of the default template, it's up to you to add it if needed.

All 'global' variables must be declared here, 'global' means outsides sub routines.
If you need variables accessible from everywhere in the project you must declare them as `Public`, otherwise as `Private`.
`Public` variables live as long as the process lives.
This sub is called once when the process starts (this is true for all modules, not just Main module).
Not all types of objects can be declared as `Public`.

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

This sub is called when the application is started.
The form is created

- when the user first launches the application

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

6.4 Sub MainForm_Resize (Width As Int, Height As Int)

This routine is called each time MainForm size is changed, mostly when the user changes the size.
Generally, there are two types of mechanisms that allow you to save the page state.
Information that is only relevant to the current application instance can be stored in one or more process variables.
Other information should be stored in a persistent storage (file or database).
For example, if the user changed some settings you should save the changes to a persistent storage at this point. Otherwise the changes may be lost.

7 Variables and objects

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

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

7.1 Variable Types

List of types with their ranges:

B4J	Type	min value	max value
Boolean	boolean	False	True
Byte	integer 8 bits	-2^7 -128	$2^7 - 1$ 127
Short	integer 16 bits	-2^{15} - 32768	$2^{15}-1$ 32767
Int	integer 32 bits	-2^{31} -2147483648	$2^{31}-1$ 2147483647
Long	long integer 64 bits	-2^{63} -9223372036854775808	$2^{63}-1$ 9223372036854775807
Float	floating point number 32 bits	-2^{-149} 1.4E-45	$(2-2^{-23}) * 2^{127}$ 3.4028235 E 38
Double	double precision number 64 bits	-2^{-1074} 2.2250738585072014 E -308	$(2-2^{-52}) * 2^{1023}$ 1.7976931348623157 E 308
Char	character		
String	array of characters		

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

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

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

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

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

```
Dim arr(3) As Int
Dim List1 As List
List1.Initialize
For i = 1 To 5
   arr(0) = i * 2
   arr(1) = i * 3
   arr(2) = i * 4
   List1.Add(arr)    'Add the whole array as a single item
Next
arr = List1.Get(0) 'get the first item from the list
Log(arr(0))         'What will be printed here???
```

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

```
Dim arr(3) As Int 'This call is redundant in this case.
Dim List1 As List
List1.Initialize
For i = 1 To 5
   Dim arr(3) As Int
   arr(0) = i * 2
   arr(1) = i * 3
   arr(2) = i * 4
   List1.Add(arr) 'Add the whole array as a single item
Next
arr = List1.Get(0) 'get the first item from the list
Log(arr(0)) 'Will print 2
```

7.2 Names of variables

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

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

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

For nodes, it is useful to add to the name a three character prefix that defines its type.
Examples:

`lblCapital`	lbl > Label	Capital > purpose
`txfInterest`	txf > TextField	Interest > purpose
`btnNext`	btn > Button	Next > purpose

7.3 Declaring variables

7.3.1 Simple variables

Variables are declared with the `Private`, `Public` or `Dim` keyword followed by the variable name and the `As` keyword and followed by the variable type.

`Public` Declares variables accessible from everywhere in the program.
Valid only in `Process_Globals` and `Class_Globals` routines !

`Private` Declares variables accessible only in the environment where they are declared.
Variables declared in a Module are accessible only in this Module.
Variables declared in a Sub are accessible only in this Sub.

`Dim` Can be used in Subs.
Any variable declared in a sub is `Private` and accessible only in this Sub.
Declaring a variable as `Public` in a Sub is non sense.

Examples:

```
Public Capital As Double
Public Interest As Double
Public Rate As Double
```

Declares three variables as Double, double precision numbers.

```
Private i As Int
Private j As Int
Private k As Int
```

Declares three variables as Int, integer numbers.

```
Private txfCapital As TextField
Private txfInterest As TextField
Private txfRate As TextField
```

Declares three variables as TextField nodes.

```
Private btnNext As Button
Private btnPrev As Button
```

Declares two variables as Button nodes.

Variables of same type can also be declared in a short way.

```
Private Capital, Interest, Rate As Double
Private i, j, k As Int
Private txfCapital, txfInterest, txfRate As TextField
Private btnNext, btnPrev As Button
```

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

Following variable declarations are also valid:

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

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

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

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

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

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

7.3.2 Array variables

Arrays are collections of data or objects that can be selected by indices. Arrays can have multiple dimensions.
The declaration contains the `Public` or `Private` keyword followed by the variable name `LastName`, the number of items between brackets `(50)`, the keyword `As` and the variable type `String`.

Examples:

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

The first index of each dimension in an array is 0.
`LastName(0), Matrix(0,0), Data(0,0,0)`

The last index is equal to the number of items in each dimension minus 1.
`LastName(49), Matrix(2,2), Data(2,4,9)`

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

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

A more versatile way to declare arrays is to use:
Variables instead of numbers.

```
Public NbPers = 50 As Int
Public LastName(NbPers) As String
Public FirstName(NbPers) As String
Public Address(NbPers) As String
Public City(NbPers) As String
```

```
Public LastName(50) As String
Public FirstName(50) As String
Public Address(50) As String
Public City(50) As String
```

We declare the variable `NbPers As Int` and set its value to 50, `NbPers = 50`.
Then we declare the arrays with this variable instead of the number 50.
The big advantage is if at some point we need to change the number of items, we change only ONE value.
For the Data array we could use the following code.

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

And the access routine.

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

Filling an array with the Array keyword :

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

7.3.3 Array of Nodes (objects)

Nodes or objects can also be in an Array. The following code shows an example:

In the example below the Buttons are added to the Application by code.

```
Sub Process_Globals
   Private Buttons(7) As Button
End Sub

Private Sub AppStart (Form1 As Form, Args() As String)
   MainForm = Form1
   Private i As Int

   For i = 0 To 6
      Buttons(i).Initialize("Buttons")
      Activity.AddView(Buttons(i), 10, 10 + i * 60, 150, 50)
      Buttons(i).Tag = i + 1
      Buttons(i).Text = "Test " & (i + 1)
   Next
End Sub

Sub Buttons_Action
   Private btn As Button

   btn = Sender

   Activity.Title = "Button " & btn.Tag & " clicked"
End Sub
```

The Buttons could also have been added in a layout file, in that case they must neither be initialized, nor added to the Activity and the Text and Tag properties should also be set in the Designer.
The individual names (btn1, btn2 etc.) must also be declared.
In that case the code would look like this:

```
Sub Process_Globals
   Private btn1, btn2, btn3, btn4, btn5, btn6, btn7 As Button
   Private Buttons() As Button
End Sub

Private Sub AppStart (Form1 As Form, Args() As String)
   MainForm = Form1
   Dim i As Int

   Buttons = Array As Button (btn1, btn2, btn3, btn4, btn5, btn6, btn7)
End Sub

Sub Buttons_Action
   Dim btn As Button

   btn = Sender

   Activity.Title = "Button " & btn.Tag & " clicked"
End Sub
```

7.3.4 Type variables

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

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

```
Public NbUsers = 50 As Int
Type Person(LastName As String, FirstName As String. Address As String, City As String)
Public User(NbUsers) As Person
Public CurrentUser As Person
```

The new personal type is `Person`, then we declare either single variables or arrays of this personal type.

Before you can use a Type variable it must be initialized!

```
User().Initialize
CurrentUser.Initialize
```

To access a particular item use following code.

```
CurrentUser.FirstName
CurrentUser.LastName

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

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

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

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

7.4 Casting

B4J casts types automatically as needed. It also converts numbers to strings and vice versa automatically.
In many cases you need to explicitly cast an Object to a specific type.
This can be done by assigning the Object to a variable of the required type.
For example, Sender keyword references an Object which is the object that raised the event.
The following code changes the color of the pressed button.
Note that there are multiple buttons that share the same event sub.

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form

   Private Btn1, Btn2, Btn3 As Button
End Sub

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

   Btn1.Initialize("Btn")    ' Note the same EventName for all three buttons
   Btn2.Initialize("Btn")
   Btn3.Initialize("Btn")
   MainForm.RootPane.AddView(Btn1, 10, 10, 200, 50)
   MainForm.RootPane.AddView(Btn2, 10, 70, 200, 50)
   MainForm.RootPane.AddView(Btn3, 10, 130, 200, 50)
End Sub
```

The above code could also be written more elegantly:

```
Sub Process_Globals

End Sub

Private Sub AppStart (Form1 As Form, Args() As String)
   Dim i As Int
   MainForm = Form1

   For i = 0 To 9    ' create 10 Buttons
      Dim Btn As Button
      Btn.Initialize("Btn")
      MainForm.RootPane.AddView(Btn, 10dip, 10dip + 60dip * i, 200dip, 50dip)
   Next
End Sub

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

7.5 Scope

7.5.1 Process variables

Variables must be declared in `Process_Globals` or `Class_Globals` routines.
They can be declared with either `Public` or `Private:`
`Public` variables can be accessed from any other module.
`Private` variables can be accessed only from the module they are declared in.

Variables can be declared with the `Dim` keyword:
```
Dim MyVar As String
```
In this case the variable is public same as Public. `Dim` remains for compatibility.
Using `Dim` in `Process_Globals` or `Class_Globals` routines is not recommended!

These subs are called once when the process starts.

To access `Public` variables in other modules than the module where they were declared their names must have the module name they were declared as a prefix.
Example:
Variable defined in a module with the name : *MyModule*
```
Sub Process_Globals
   Public MyVar As String
End Sub
```

Accessing the variable in *MyModule* module:
```
   MyVar = "Text"
```

Accessing the variable in any other module:
```
   MyModule.MyVar = "Text"
```

7.5.2 Local variables

Variables declared in a subroutine are local to this subroutine even if they are declared as `Public` which is none sense.
They are `Private` and can only be accessed from within the subroutine where they were declared.
All objects types can be declared as local variables.
At each call of the subroutine the local variables are initialized to their default value or to any other value you have defined in the code and are 'destroyed' when the subroutine is left.

7.6 Tips

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

For example, the following code disables all Buttons a Page:

```
For Each v As Node In MainForm.RootPane.GetAllViewsRecursive
   If v Is Button Then
      Private b As Button = v
      b.Enabled = False
   End If
Next
```

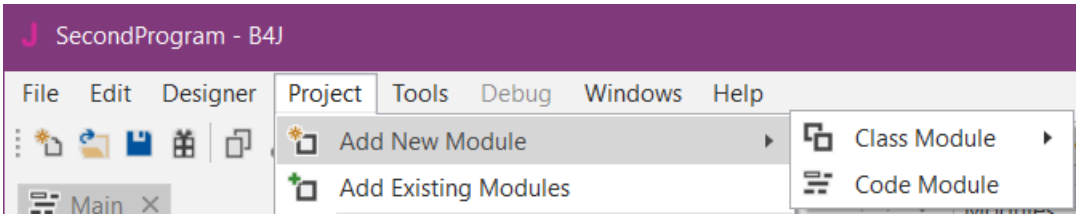
8 Modules

At least one module exists, the main module.
Its name is always **Main** and cannot be changed.

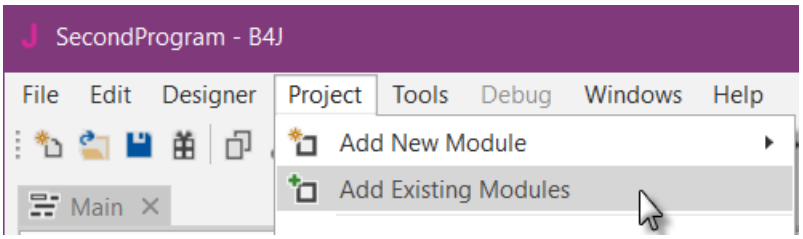
There do exist two types of modules:

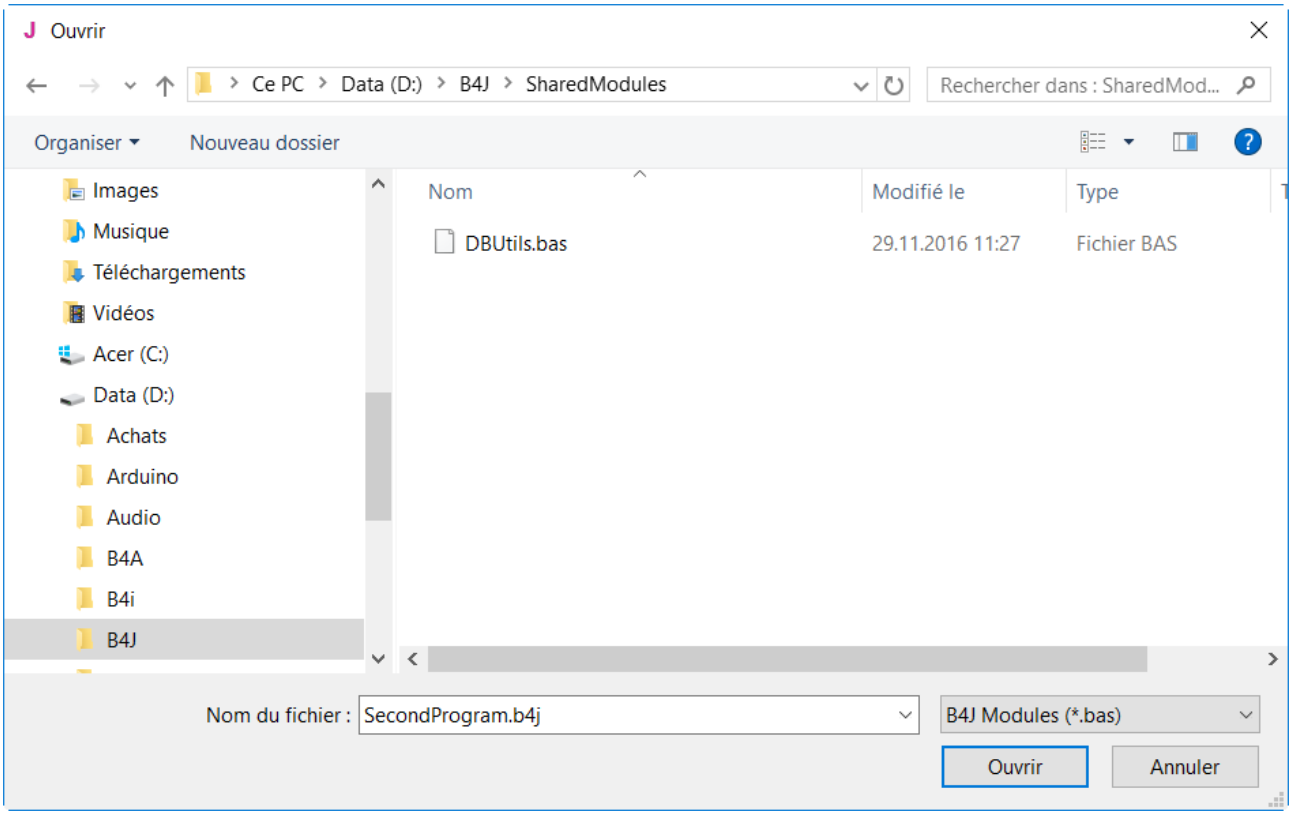
- Class modules
- Code modules

To add a new module click on either Class Module or Code Module in the IDE menu Project / Add New Module.



To add an existing module click on Add Existing Module in the IDE menu Project.





Select the file to add, it will be copied to the project folder.

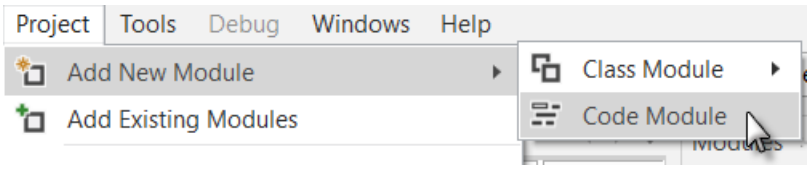
8.1 Code modules

Code modules contain code only.

The purpose and advantage of code modules is sharing same code in different programs, mainly for calculations or other general management.

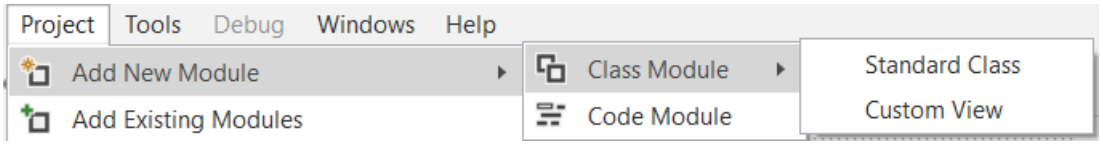
Some code modules, called utilities, are already published by Erel in the forum:

- DBUtils, Database management utilities.
- ChartsFrameWork, Charts drawing module.



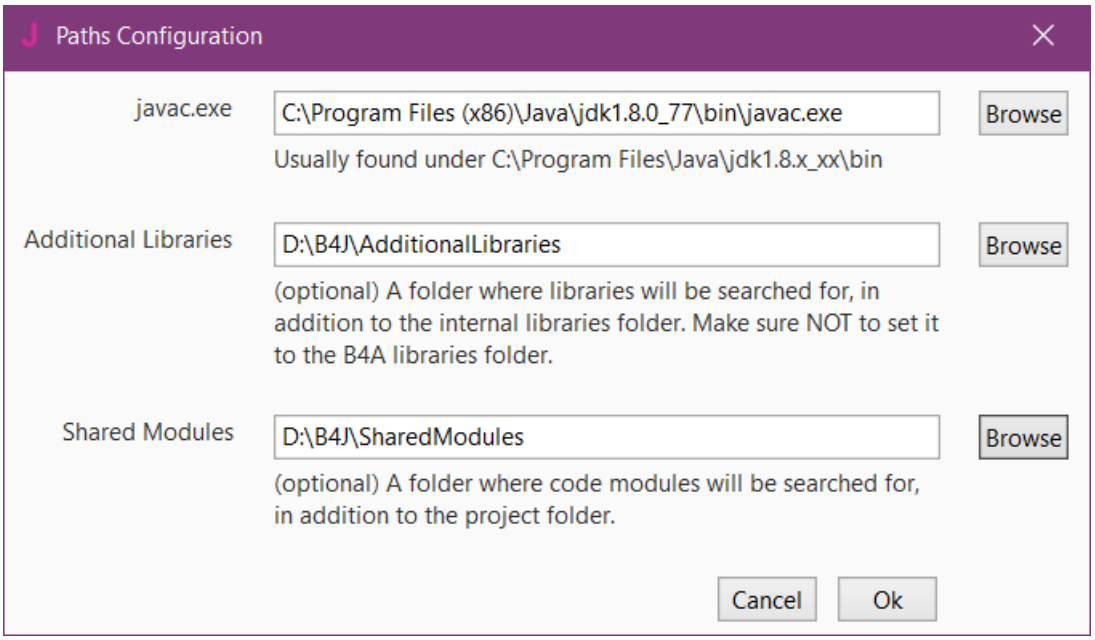
8.2 Class modules

Class modules are out of the scope of this guide.

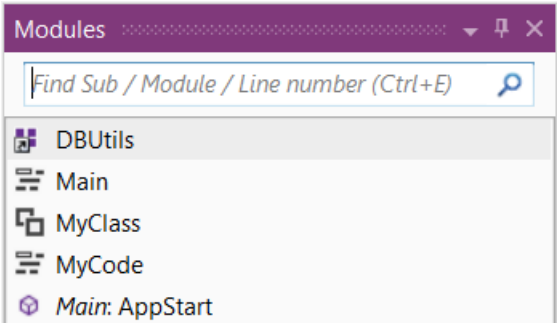


8.3 Shared modules

Modules are in principle saved it the folder of each project using it.
Modules useful in several project can be shared without loading them in each projects folder, that's the purpose of *shared modules*.
Shared module files must be stored in a specific 'Shared Modules' folder which must be defined in the IDE menu *Tools - Configure Paths*.



You can see that a module was loaded from the shared folder in the list of modules:



Main and MyCode are code modules.

MyClass Class module.

DBUtils Shared module.

Adding a shared module to a project is done in the same way as adding a non-shared module.
You choose Project -> Add Existing Module. If the module file is in the 'SharedModules' folder then the module will be loaded as a shared module and will not be copied to the project folder.

If you want to convert a non-shared module to a shared module then you need to manually move the module file to the shared modules folder and reload the project

9 Basic language

In computer programming, BASIC (an acronym which stands for **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) is a family of high-level programming languages designed to be easy to use.
The original Dartmouth BASIC was designed in 1964 by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College in New Hampshire, USA to provide computer access to non-science students. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to do. The language and its variants became widespread on microcomputers in the late 1970s and 1980s.
BASIC remains popular to this day in a handful of highly modified dialects and new languages influenced by BASIC such as Microsoft Visual Basic (source Wikipedia).

9.1 Program flow

This is a summary of the more detailed explanations in Process and Activity life cycle.

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form
End Sub

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

The program goes through following routines when starting from top to down:

9.1.1 Process_Globals routine

Dedicated to the declaration of variables and objects.
Variables declared with:

- `Private` are valid only in the module where they are declared.
- `Public` are valid during the whole life time of the process and accessible from everywhere in the program.

9.1.2 AppStart routine

Area to initialize, add nodes and to define node properties, if necessary.
When you run the IDE a default template is included.

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

9.2 Expressions

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

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

9.2.1 Mathematical expressions

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

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

Examples:

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

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

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

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

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

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

9.2.2 Relational expressions

In computer science in relational expressions an operator tests some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 >= 3).
In B4J these operators return **True** or **False**, depending on whether the conditional relationship between the two operands holds or not (source Wikipedia).

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

9.2.3 Boolean expressions

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

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

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

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

9.3 Conditional statements

Different conditional statements are available in Basic.

9.3.1 If – Then – End If

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

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

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

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

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

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

The simplest **If - Then** structure.

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

The same but in one line.

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

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

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

The same but in one line.

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

Note. Difference between:

B4J / B4i / B4A	VB
`Else If`	`ElseIf`

In B4A there is a blank character between `Else` and `If`.

Some users try to use this notation:

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

There is a big difference between B4J and VB that gives errors:
The above statements is equivalent to:

B4J / B4i / B4A	VB

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

The colon character ' : ' in the line above is treated in B4J like a CarriageReturn CR character.

9.3.2 Select – Case

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

General case:

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

TestExpression is the expression to test.

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

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

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

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

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

Examples:

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

The Value variable is a numeric value.

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

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

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

The **TestExpression** is a character at

Note. Differences between:

B4J / B4i / B4A	VB
`Select Value`	`Select Case Value`
`Case 1,2,3,4,8,9,10`	`Case 1 To 4 , 8 To 9`

In VB the keyword `Case` is added after the `Select` keyword.
VB accepts `Case 1 To 4` , this is not implemented in B4J.

9.4 Loop structures

Different loop structures are available in Basic.

9.4.1 For – Next

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

```
For i = n1 To n2 Step n3
   ' Specific code
Next
```

i incremental variable
n1 initial value
n2 final value
n3 step

The **For – Next** loop works as below:

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

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

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

is the same as

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

The step variable can be negative.

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

Next
```

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

```
For i = 0 To 10
   ' code
   If A = 5 Then Exit
   ' code
Next
```

In this example, if the variable a equals 5
Then exit the loop.

Note: Differences between

B4J / B4i / B4A	VB
`Next`	`Next i`
`Exit`	`Exit For`

In VB:

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

9.4.2 For - Each

It is a variant of the For - Next loop.

Example:

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

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

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

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

Example For - Each:

```
Public Numbers() As Int
Public Sum As Int

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

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

Same example but with a For - Next loop :

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

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

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

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

```
For Each lbl As Label In Page1.RootPanel
   lbl.Font = Font.CreateNew(20)
Next
```

Same example with a For - Next loop :

```
For i = 0 To Page1.RootPanel.NumberOfViews - 1
   Private v As View
   v = Page1.Rootctivity.GetView(i)
   If v Is Label Then
      Private lbl As Label
      lbl = v
      lbl.Font = Font.CreateNew(20)
   End If
Next
```

9.4.3 Do - Loop

Several configurations exist:

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

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

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

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

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

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

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

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

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

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

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

Examples:

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

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

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

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

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

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

tr.Close
```

Note : Difference between:

B4J / B4i / B4A	**VB**
`Exit`	`Exit Loop`

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

VB accepts also the following loops, which are not supported in B4J.
```
Do                          Do
  ' code                      ' code
Loop While test             Loop Until test
```

9.5 Subs

A Subroutine ("Sub") is a piece of code. It can be any length, has a distinctive name and a defined scope (in the means of variables scope discussed earlier). In Basci4i code, a subroutine is called "Sub", and is equivalent to procedures, functions, methods and subs in other programming languages. The lines of code inside a Sub are executed from first to last.
It is not recommended to have too long Subs, they get less readable.

9.5.1 Declaring

A Sub is declared in the following way:

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

It starts with the keyword `Private` or `Public`, depending on the scope, followed by the keyword `Sub`, followed by the Subs name, followed by a parameter list, followed by the return type and ends with the keywords `End Sub`.
Subs are always declared at the top level of the module, you cannot nest two Subs one inside the other.

9.5.2 Calling a Sub

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

For example:

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

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

9.5.3 Calling a Sub from another module

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

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

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

instead of:

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

9.5.4 Naming

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

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

9.5.5 Parameters

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

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

In B4J, the parameters are transmitted by value and not by reference.

It is possible to transmit nodes or objects to Subs, like:

```
Public Sub MySub(lbl As Label)
```

9.5.6 Returned value

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

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

9.6 Events

In Object-oriented programming, we have objects which can react on different user actions called events.
The number and the type of events an object can raise depend on the type of the object.
User interface objects are called 'Nodes' in Java.

Summary of the events for different nodes:

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

The most common events are:

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

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

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

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

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

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

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

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

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

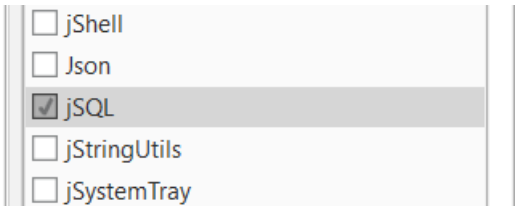
9.7 Libraries

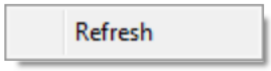
Libraries add more objects and functionalities to B4J.
Some of these libraries are shipped with B4J and are part of the standard development system.
Others, often developed by users, can be downloaded (by registered users only) to add supplementary functionalities to the B4J development environment.

All B4J specific libraries have "j" as a prefix.

When you need a library, you have to:

- Check in the Lib Tab, if you already have the library.
- For additional libraries, check if it's the latest version.
- If **yes**, then check the library in the list to select it.



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

and check the library in the list to select it.

9.7.1 Standard libraries

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

9.7.2 Additional libraries folder

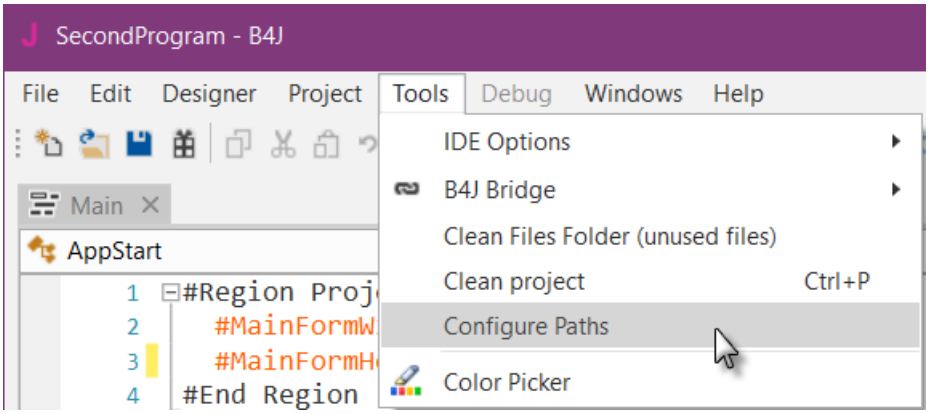
For the additional libraries it is useful to setup a special folder to save them somewhere else.
For example: C:\B4J\AdditionalLibraries

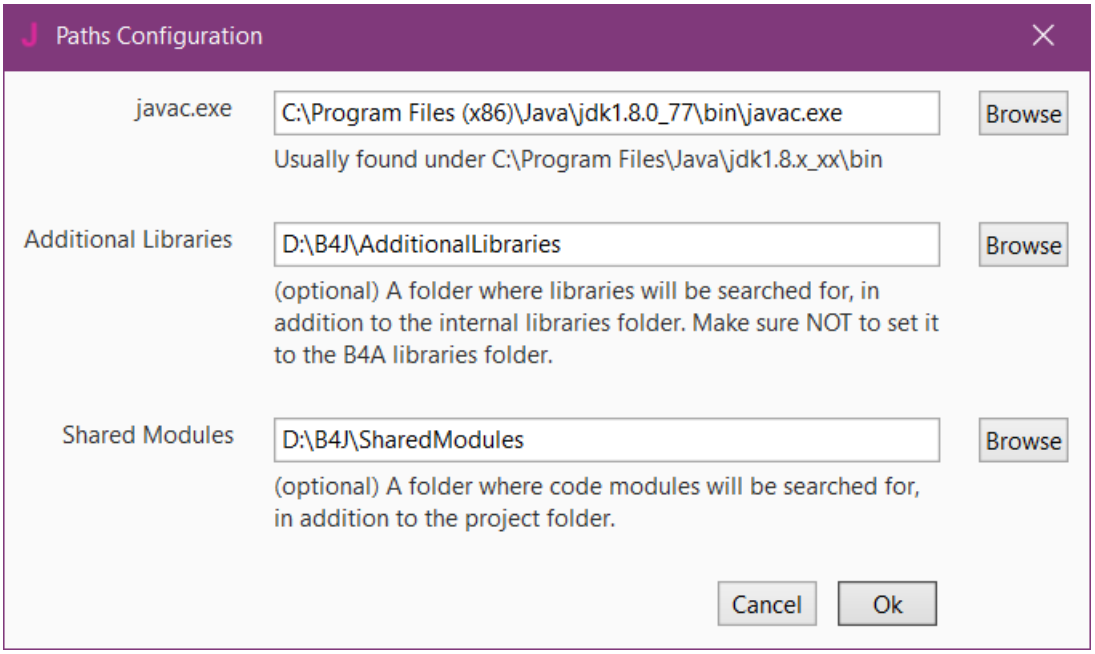
When you install a new version of B4J, all standard libraries are automatically updated, but the additional libraries are not included. The advantage of the special folder is that you don't need to care about them because this folder is not affected when you install the new version of B4J.
The additional libraries are not systematically updated with new version of B4J.

When the IDE starts, it looks first for the available libraries in the Libraries folder of B4J and then in the folder for the additional libraries.

If you setup a special additional libraries folder you must specify it in the IDE.

In the menu Tools / Configure Paths:





Enter the folder name and click on Ok.

9.7.3 Load and update a Library

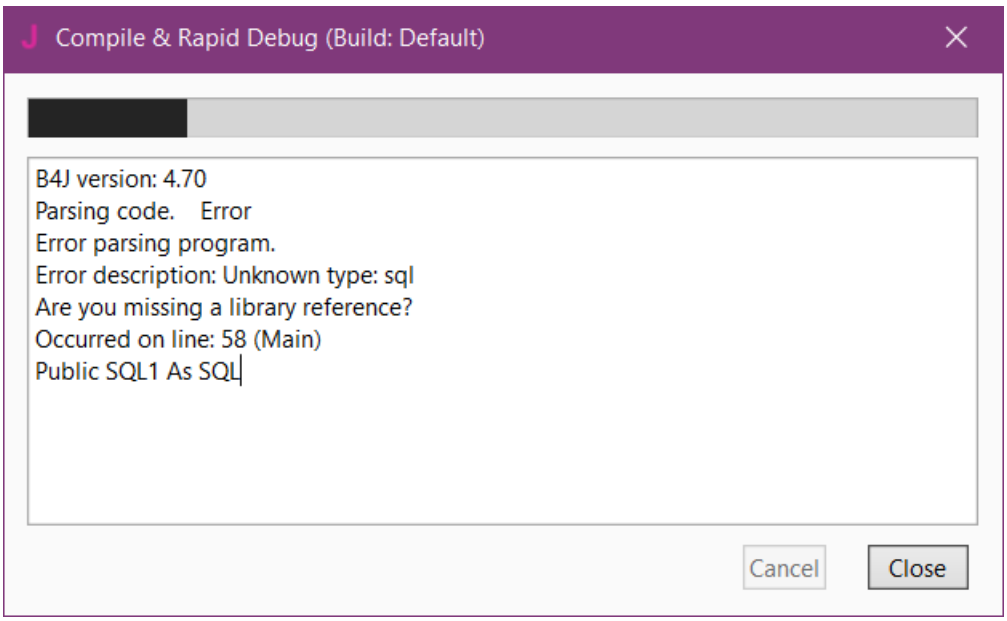
A list of the official and additional libraries with links to the relevant forum threads is shown in B4J Documentation.

To load or update a library follow the steps below:

- Download the library zip file somewhere.
- Unzip it.
- Copy the xxx.xml file to the
  - B4J Library folder for a standard B4J library
  - Additional libraries folder for an additional library.
- Right click in the Lib area and click on Refresh and check the library in the list to select it.

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

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



In the program line where the error occurs the unknown object is highlighted in red.

```
Private lblSelectedItem As Label
Public SQL1 As SQL
```

9.8 String manipulation

B4J allows string manipulations like other Basic languages but with some differences.

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

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

Result: `123; 234; 45; 23`

The different functions are:

- **CharAt(Index)** Returns the character at the given index.
- **CompareTo(Other)** Lexicographically compares the string with the Other string.
- **Contains(SearchFor)** Tests whether the string contains the given SearchFor string.
- **EndsWith(Suffix)** Returns True if the string ends with the given Suffix substring.
- **EqualsIgnoreCase(Other)** Returns True if both strings are equal ignoring their case.
- **GetBytes(Charset)** Encodes the Charset string into a new array of bytes.
- **IndexOf(SearchFor)** Returns the index of the first occurrence of SearchFor in the string.
- **IndexOf2(SearchFor, Index)** Returns the index of the first occurrence of SearchFor in the string. Starts searching from the given index.
- **LastIndexOf(SearchFor)** Returns the index of the first occurrence of SearchFor in the string. Starts searching from the end of the string.
- **LastIndexOf2(SearchFor, Index)** Returns the index of the first occurrence of SearchFor in the string. The search starts at the given index and advances to the beginning.
- **Length** Returns the length, number of characters, of the string.
- **Replace(Target, Replacement)** Returns a new string resulting from the replacement of all the occurrences of Target with Replacement.
- **StartsWith(Prefix)** Returns True if this string starts with the given Prefix.
- **Substring(BeginIndex)** Returns a new string which is a substring of the original string. The new string will include the character at BeginIndex and will extend to the end of the string.
- **Substring2(BeginIndex,EndIndex)** Returns a new string which is a substring of the original string. The new string will include the character at BeginIndex and will extend to the character at EndIndex, not including the last character.
- **ToLowerCase** Returns a new string which is the result of lower casing this string.
- **ToUpperCase** Returns a new string which is the result of upper casing this string.
- **Trim** Returns a copy of the original string without any leading or trailing white spaces.

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

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

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

9.9 Timers

A Timer object generates ticks events at specified intervals. Using a timer is a good alternative to a long loop, as it allows the UI thread to handle other events and messages.

A timer has:

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

  - **IsInitialized** Returns True if the Timer is initialized.
    Timer1.IsInitialized
    Ex: `Init = Timer1.IsInitialized`

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

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

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

The Timer must be declared in a Process_Global routine.

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

But it must be initialized in the AppStart routine in the module where the timer tick event routine is used.

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

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

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

9.10 Files

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

9.10.1 File keyword

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

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

File.DirAssets
Returns a reference to the files added to the Files tab.
These files are read-only in Release mode.

File.DirApp
Returns the application folder.

File.DirData
Returns the path to a folder that is suitable for writing files.
On Windows, folders under Program Files are read-only. Therefore File.DirApp will be read-only as well.
This method returns the same path as File.DirApp on non-Windows computers.
On Windows, it returns the path to the user data folder.
For example:
C:\Users\[user name]\AppData\Roaming\[AppName]

File.DirTemp
A temporary folder.

File.Exists (Dir As String, FileName As String)
Returns True if the file exists and False if not.

The File object includes several methods for writing to files and reading from files.

To be able to write to a file or to read from a file, it must be opened.

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

File.OpenInput (Dir As String, FileName As String)
- Opens an InputStream to the given file.

File.WriteString (Dir As String, FileName As String, Text As String)
- Writes the string to a new file with UTF-8 encoding.

File.ReadString (Dir As String, FileName As String) As String
- Reads a file and returns its content as a string with UTF-8 encoding.

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

File.ReadList (Dir As String, FileName As String) As List
- Reads the given text file and returns a list. Each line in the file is converted to a list item.

File.WriteMap (Dir As String, FileName As String, Map As Map)
- Creates a new file and writes the given map. Each key value pair is written as a single line.
All values are converted to strings.
See this link for more information about the actual format: Properties format at http://en.wikipedia.org/wiki/.properties.
You can use File.ReadMap to read this file.

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

File.ReadMap2 (Dir As String, FileName As String) As Map
- Similar to ReadMap. ReadMap2 adds the items to the given Map.
By using ReadMap2 with a populated map you can force the items order as needed.
Example:

```
Dim m As Map
m.Initialize
m.Put("Item #1", "")
m.Put("Item #2", "")
m = File.ReadMap2(File.DirApp, "settings.txt", m).
```

Some other useful functions:

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

File.Copy2 (In As InputStream, Out As OutputStream)
- Copies the input stream to the output stream. The input stream is closed at the end.

File.Delete (Dir As String, FileName As String) As Boolean
- Deletes the given file from the given directory. Returns True if the file was deleted.

File.ListFiles (Dir As String) As List
- Returns a read only list with all the files and directories which are stored in the specified path.
An uninitialized list will be returned if the folder is not accessible.
This method does not support the assets folder.
Example:
```
Dim List1 As List
List1 = File.ListFiles(File.DirDocuments)
```
List1 can be 'dimed' in Sub Process_Globals

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

File.Combine (Dir As String, FileName As String) As String
- Combines the Dir and FileName to a single string.
This method does not support files in the assets folder.

File.MakeDir (Parent As String, Dir As String)
- Creates a new folder under the Parent folder.

File.GetFileParent (FileName As String) As String
- Returns the path of the file or folder parent.

File.GetName (FilePath As String) As String
- Returns the file name from the full path (or the directory name in case of a directory).

File.GetUri (Dir As String, FileName As String) As String
- Returns a Uri string ("file://...") that points to the given file.

File.LastModified (Dir As String, FileName As String) As String
- Returns the last modified date of the specified file.
This method does not support files in the assets folder.
Example:
```
Dim d As Long
d = File.LastModified(File.DirApp, "1.txt")
Msgbox(DateTime.Date(d), "Last modified")
```

9.10.2 Filenames

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

Example : MyFile.txt

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

9.10.3 Subfolders

You can define subfolders in B4J with.

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

To access the subfolder you should add the subfolder name to the folder name with “\" in-between.

```
Image1.Initialize(File.DirApp & "\Picutres", "test.jpg")
```

Or add the subfolder name before the filename with "/" in-between.

```
Image1.Initialize(File.DirApp, "Picutres\test.jpg")
```

Both possibilities work.

9.10.4 TextWriter

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

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

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

Writer could be declared in Sub Globals.

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

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

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

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

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

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

TextWriter.Close
- Closes the stream.

Example:

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

9.10.5 TextReader

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

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

Example:
```
Private Reader  TextReader
Reader.Initialize(File.OpenInput(File.DirApp, "Test.txt"))
```

Reader could be declared in Sub Globals.

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

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

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

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

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

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

Example:

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

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

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

9.10.6 Text encoding

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

The 'standard' text encoding, in B4J, is UTF-8.

But in Windows the most common character sets are ASCII and ANSI.

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

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

iOS can use following character sets:

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

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

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

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

The symbol for the end of line is:

- Windows CRLF Chr(13) & Chr(10)
- B4J, B4A and B4i CRLF Chr(10)

9.11 Lists

Lists are similar to dynamic arrays.

Lists are often used and many examples can be found in code examples:

- jStringUtils LoadCSV, SaveCSV
- DBUtils module InsertMaps, UpdateRecord, ExecuteMemoryTable, ExecuteSpinner, ExecuteListView, ExecuteHtml, ExecuteJSON
- Charts module to hold different variables.

A list must be initialized before it can be used.

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

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

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

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

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

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

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

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

A list can hold any type of object.
B4J automatically converts regular arrays to lists. So when a List parameter is expected you can pass an array instead.

Get the size of a List:

- `List1.Size`

Use the Get method to get an item from the list with:

- Get(Index As Int)

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

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

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

Lists can be saved and loaded from files with:

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

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

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

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

A single item can be changed with:

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

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

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

- Sort(Ascending As Boolean)

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

- SortCaseInsensitive(Ascending As Boolean)

Clear a List with:

- `List1.Clear`

9.12 Maps

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

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

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

Maps are very useful for storing applications settings.

Maps are used in these example codes:

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

A list must be initialized before it can be used.

- **Initialize** Initializes an empty Map.
```
Dim Map1 As Map
Map1.Initialize
```

Add a new entry:

- **Put**(Key As Object, Value As Object)
```
Map1.Put("Language", "English")
```

Get an entry:

- **Get**(Key As Object)
```
Language = Map1.Get("Language")
```

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

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

Remove an entry:

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

Clear an entry, clears all items from the map:

- **Clear**
```
Map1.Clear
```

Maps can be saved and loaded with:

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

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

10 Differences B4J <> B4A

Even though B4J and B4A are similar, there are differences because of the different operating systems.

Only differences are explained in this chapter.

10.1 Nodes <> Views jFX library

User interface objects are called Nodes in B4J, against Views in B4A.
These objects are not in the Core library but in the jFX library, which is added by default to new projects.

10.2 CSSUtils

Certain node properties can only be changed with the additional CSSUtils library.
This library is not added by default to new projects you need to check it yourself.
The methods are explained in the CSSUtils chapter.

10.3 Screens Form <> Activity

B4J
The different screens are Forms. Several Forms can be displayed at the same time on the screen.
Forms can be resized by the user.

B4A
The different screens are manly managed with Activities in separate modules, or on Panels managed in the Main Activity or a mix of both.

10.4 Panel <> Pane

B4J
Panels are called Panes in B4J.

If a Pane, without event routines, covers other nodes the events are NOT submitted to underlying nodes. In B4A they are!
If you want to submit them, you must set the MouseTransparent property to True:

```
Private jo = cvsLayer2 As JavaObject
jo.RunMethod("setMouseTransparent", Array As Object(True))
```

Panes have MouseEvents instead of Touch events for B4A Panels.
Transparent panel: Background: Color.Transparent Alpha = 1
You cannot draw onto a Pane with a Canvas, Canvas is a Node in B4J.

B4A
Panels have a background bitmap.
It is also possible to draw onto a Panel with a Canvas.
If a Panel, without event routines, covers other nodes the events ARE submitted to the underlying nodes. In B4J they are NOT!
To avoid this, one solution is to add an empty event routine.

Transparent panel: Background: Color.Transparent Alpha = 0

10.5 Canvas

Canvas is a Node (View) on its own.
When you use a Canvas, you don't need to refresh the drawing to make it visible.

There are differences in the drawing routines, just one example below.
The drawing routines are explained in the Graphics chapter.

B4J
No need to Invalidate or refresh the drawing.

```
Canvas1.DrawImage(Image As Image, x As Double, y As Double, Width As Double, Height As
Double)
```

B4A
You refresh the Canvas node like Panel1.Invalidate.
In B4A you can also refresh only a part of the Canvas node, limited by a rectangle Rect, with Invalidate2(Rect).

```
Canvas1.Initialize(Panel1)      'links Canvas1 to Panel1
Canvas1.DrawBitmap(Bitmap1 As Bitmap, SrcRect As Rect, DestRect As Rect)
Panel1.Invalidate               'refreshes the drawing on Panel1
```

10.6 Mouse Events <> Touch event

In B4J there are no Touch events!
Instead, there are several Mouse events.

B4J.

Mouse event:

- **MouseClicked** (EventData As MouseEvent)
  Raised when a mouse button was pressed and released.

- **MouseMoved** (EventData As MouseEvent)
  Raised when the mouse cursor moves above the node, no mouse button clicked.

- **MouseDragged** (EventData As MouseEvent)
  Raised when the mouse cursor moves above the node and one mouse button is pressed at the same time. Equivalent to ACTION_MOVE in B4A.

- **MousePressed** (EventData As MouseEvent)
  Raised when a mouse button was pressed. Equivalent to ACTION_DOWN in B4A.

- **MouseReleased** (EventData As MouseEvent)
  Raised when a mouse button was released. Equivalent to ACTION_UP in B4A.

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

- **MouseCursor**
  This is not an event but a property.
  Gets or sets the mouse cursor shape that will be used when the mouse is in the node's bounds.
  Example:
  ```
  cvsTest.MouseCursor = fx2.Cursors.E_RESIZE
  ```
  More in chapter fx.Cursors.

B4A.

In B4A the Touch event returns the action taken as a parameter in the same event, instead of several events.

```
Sub Panel1_Touch (Action As Int, X As Float, Y As Float)
   Select Action
   Case Panel1.ACTION_DOWN
   Case Panel1.ACTION_MOVE
   Case Panel1.ACTION_UP
   End Select
End Sub
```

ACTION_DOWN	equivalent to MousePressed.
ACTION_MOVE	equivalent to MouseDragged.
ACTION_UP	equivalent to MouseReleased.

10.7 Text views TextField / TextArea <> EditText

B4J
B4J has two nodes to enter text:

TextField	Single line.
TextArea	Multiline.

B4A
B4A has only one view to enter text, single line or multiline:

EditText	Can be single line or multiline.

10.8 ScrollPane / ScrollViews

B4J
ScrollViews are called ScrollPanes, they scroll in both directions.
It's equivalent to ScrollView2D in B4A.

The use of ScrollPane is different from B4A.
If you want to add nodes in the code, you need to load a layout onto the ScrollPane to get access to its internal pane.
The ScrollPane project shows an example.

```
Private ScrollPane1 As ScrollPane

Private innerPane As Pane
innerPane = ScrollPane1.InnerNode      'gets the innet Pane
ScrollPane1.SetVScrollVisibility("ALWAYS")   'shows the vertical scrollslider
ScrollPane1.SetHScrollVisibility("ALWAYS")   'shows the horizontal scrollslider
ScrollPane1.Pannable = True     'allows moving with the mouse
```

Change the internal Pane dimensions:
ScrollPane1.LoadLayout("ScrollPane1", 0, 0)

```
innerPane.PrefWidth = xx
innerPane.PrefHeight = yy
```

B4A

ScrollView	Scrolls only in vertical direction.
HorizontalScrollView	Scrolls only in horizontal direction.
ScrollView2D	Scrolls in both directions.

Change the internal panel size.

```
ScrollView1.Panel.Height
HorizontalScrollView1.Panel.Width
ScrollView2D1.Panel.Height
ScrollView2D1.Panel.Widht
```

10.9 TabPane <> TabHost

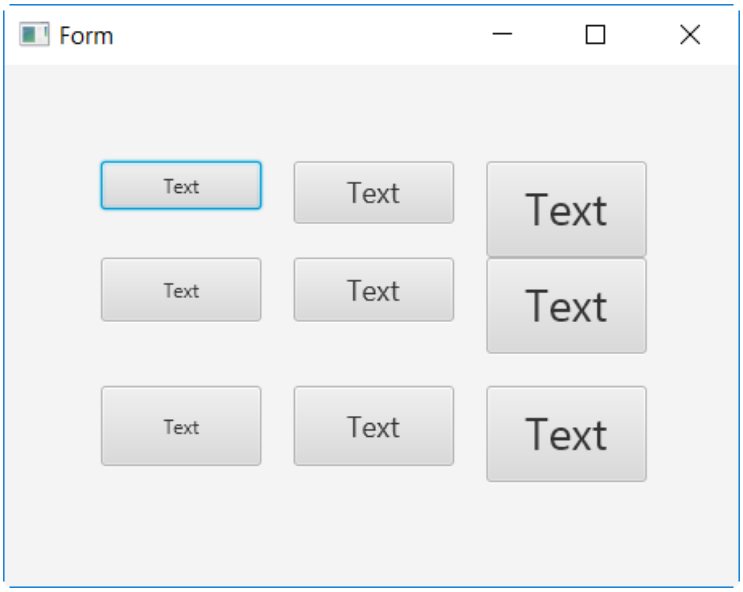
10.10 Button height and font size

B4J

The Button height automatically adjusts in relation with the font size.

In the image below we have:

Height of top buttons:	30 pixels.	Font size left column buttons:	12
Height of middle buttons:	40 pixels.	Font size middle column buttons:	18
Height of bottom buttons:	50 pixels.	Font size right column buttons:	28

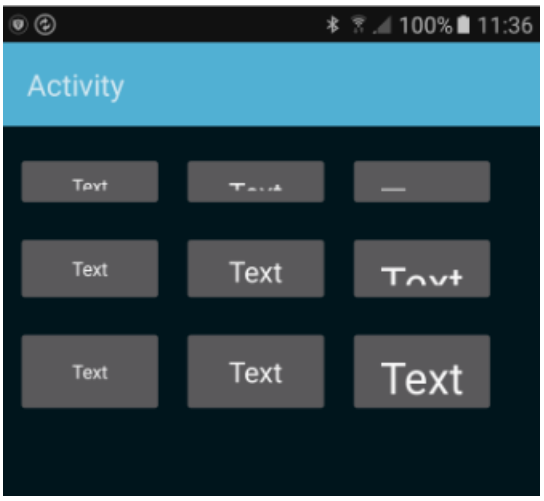


We see that:

- In the middle column, the height of the top button has increased.
- In the right column, the height of all button has increased.

B4A

The text is truncated.



10.11 RadioButton

B4J
You can group RadioButtons with:
GroupRadioButtons(RadioButtons As List)

Creates a group of RadioButtons which are stored in the list.
Only one of the group buttons can be selected at any time.
Note that RadioButtons added with the internal designer are grouped based on their parent like in B4A.

B4A
To group you need to put them onto a same Panel.

10.12 Slider / SeekBar

Similar nodes, only the name changes.

10.13 Node Background

B4J
Only color with border or image.
To set the background we need to add the CSSUtils library to the project!

```
SetBackgroundColor(Node As Node, Color As fx.Color)
SetBorder(Node As Node, Width As Double, Color As fx.Color, CornerRadius As Double)
SetBackgroundImage(Node As Node, Dir As String, FileName As String)
```

B4A
The Background is a Drawable.

- ColorDrawable same as SetBackgroundColor in B4J.
- GradientDrawable doesn't exist in B4J.
- BitmapDrawable doesn't exist in B4J.
- StateListDrawable doesn't exist in B4J.

10.14 Msgbox / Msgbox2

B4J
Similar to B4A, the difference, it needs the jFX library!
It needs an Owner object, more details in chapter fx.Msgbox / fx.Msgbox2.
Modal node.

```
fx.Msgbox(Qwner As Form, Message As String, Title As String)
fx.Msgbox2(Owner As Form, Message As String, Title As String, Positive As String,
Cancel As String, Negative As String, Style As Object)
```

B4A
Modal object.

```
Msgbox(Message As String, Title As String)
Answer = Msgbox2(Message As String, Title As String, Positive As String, Cancel As
String, Negative As String, Icon As Bitmap)
```

Example:

```
Answer = Msgbox2("MsgDelete", "Do you really want to delete the entry ?", "Delete
entry", "Yes", "", "No", Null))
```

10.15 SQLite ResultSet <> Cursor

B4J
The returned object is called ResultSet.
Code to go through the results

```
Private rs As ResultSet
rs = SQL1.ExecQuery2("SELECT * FROM table1 WHERE col1 = ?", Array(100))
Do While rs.NextRow
   ' your code
Loop
```

B4A

In B4A you can use the same code as in B4J.

In B4A you can use another object called Cursor, the Cursor object doesn't exist in B4J.
It is possible to position the Cursor at a given item with Cursor.Position = i
Code to go through the results

```
Private Curs As Cursor
Private i As Int
For i = 0 To Curs.RowCount - 1
   Curs.Position = i
   ' your code
Next
```

10.16 Font <> TextSize

B4J
Font is an object, to change the text size you need to create a new Font object

```
Label1.Font = fx.DefaultFont(14)
```

B4A
TextSize is a property which can be changed directly.

```
lblLabel1.TextSize = 20
```

10.17 File object and Folders

B4J
Predefined folders:

- **DirAssets**
  Same as B4A
- **DirApp**
  Returns the application folder.
- **DirData**
  Returns the path to a folder that is suitable for writing files.
  On Windows, folders under Program Files are read-only. Therefore File.DirApp will be read-only as well.
  This method returns the same path as File.DirApp on non-Windows computers.
  On Windows it returns the path to the user data folder. For example:
  C:\Users\[user name]\AppData\Roaming\[AppName]
- **DirTemp**
  Returns the temporary folder.

B4A
Predefined folders:

- **DirAssets**
  Same as B4J
- **DirInternal**
  Returns the folder in the device internal storage that is used to save application private data.
- **DirInternalCache**
  Returns the folder in the device internal storage that is used to save application cache data.
  This data will be deleted automatically when the device runs low on storage.
- **DirDefaultExternal**
  Returns the application default external folder which is based on the package name.
  The folder is created if needed.
- **DirRootExternal**
  Returns the root folder of the external storage media.
  If the device has an internal sdcard, then DirRooiExternal points to this one and not to an external sdcard.

10.18 Regex

B4J
Regex.Split allows “” (empty string) as the split string. Same as in B4A.

B4A
Regex.Split allows “” (empty string) as the split string

B4i
Regex.Split doesn’t allow “”(empty string) as the split string

11 CSSUtils and fx objects

A certain number of methods and properties for nodes can only be accessed with the CSSUtils and jFX library.

11.1 CSSUtils

Manly background methods.
More documentation can be found here: JavaFX CSS Reference Guide
The CSSUtils library is not added automatically to the project, you must check it in the Libraries Manager Tab.

11.1.1 CSSUtils.SetBackgroundColor

Sets the background color of the given node.

SetBackgroundColor(Node As ConcreteNodeWrapper, Color As PaintWrapper)

```
CSSUtils.SetBackgroundColor(MyLabel, fx.Colors.RGB(255, 235, 205))
```

11.1.2 CSSUtils.SetBackgroundImage

Sets the background image to the given node.

SetBackgroundImage(Node As ConcreteNodeWrapper, Dir As String, FileName As String)

```
CSSUtils.SetBackgroundImage(MyPane, File.DirAssets, "rose.jpg")
```

11.1.3 CSSUtils.SetBorder

Sets the border, width, color and corner radius to the given node.

SetBorder(Node As ConcreteNodeWrapper, Width As Double, Color As PaintWrapper, CornerRadius As Double)

```
CSSUtils.SetBorder(MyLabel, 2, fx.Colors.Blue, 5)
```

11.1.4 CSSUtils.SetStyleProperty

Sets a style property,

SetStyleProperty(Node As ConcreteNodeWrapper, Key As String, Value As String)

Key = style key. `"-fx-background-color"` in the example below.
Value = value of the style key. `"blanchedalmond"` in the example below.

The example to set the background color have done like below :

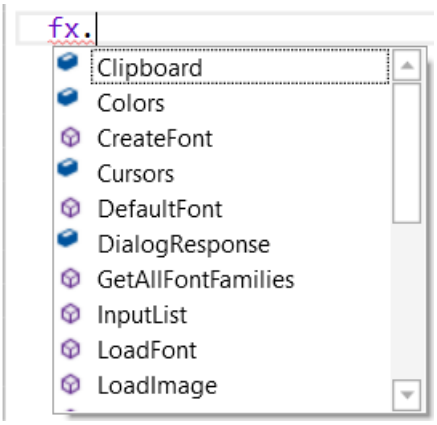
```
CSSUtils.SetStyleProperty(MyLabel, "-fx-background-color", "blanchedalmond")
CSSUtils.SetStyleProperty(MyLabel, "-fx-background-color", "rgb(255, 235, 205)")
CSSUtils.SetStyleProperty(MyLabel, "-fx-background-color", "#FFEBCD")
```

11.2 fx objects

fx objects are included in the jFX library which is added by default to any project.
There is no need to declare nor initialize something.

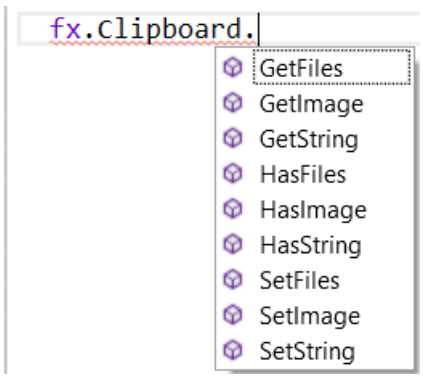
Examples in the jFX project in the SourceCode folder.

To call an fx object, just type `fx.` In the IDE and you get a popup window showing all objects.



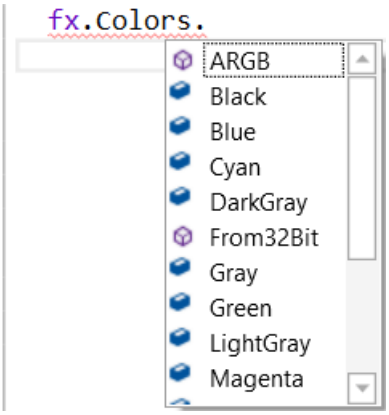
11.2.1 fx.Clipboard

To call the Clipboard, just type `fx.Clipboard.` in the IDE and you get a popup window showing all functions.



11.2.2 fx.Colors

This is a Paint object, it is used wherever you need a color:
To define a Color, just type `fx.Colors.` in the IDE and you get a popup window showing all the available colors.



You can either define a variable as a Paint object or use the color directly.

Examples:
Set background color, using the color directly:

```
CSSUtils.SetBackgroundColor(MyLabel, fx.Colors.RGB(255, 235, 205))
```

Draw a circle, with a Paint variable:

```
Private colBlue As Paint
colLine = fx.Colors.Blue
cvsTest.DrawCircle(50, 50, 40, colLine, True, 1)
```

11.2.3 fx.CreateFont

You can set different fonts.
You can either define a variable as a Font object or use the font directly.

fx.CreateFont(FamilyName As String, Size As Double, Bold As Boolean, Italic As Boolean)
FamilyName = name of the font, like “Arial”, “Times New Roman” etc.
Size = size of the font.
Bold = True or False
Italic = True or False

Examples:
Set a special Font to a Label, with a Font variable:

```
Private FontArial As Font
FontArial = fx.CreateFont("Arial", 20, False, True)
lblTestFont.Font = FontArial
```

Draw a text with the given font, using the object directly:

```
cvsLayer(2).DrawText("Rose", x1, y1, fx.CreateFont("Arial", 20, False, False),
fx.Colors.Red, "RIGHT")
```

11.2.4 fx.DefaultFont

You can set a new font size with fx.DefaultFont.

fx.DefaultFont(Size As Double)

Example:

```
lblTest.Font = fx.DefaultFont(10)
```

11.2.5 fx.LoadFont

You can load a custom font with LoadFont.

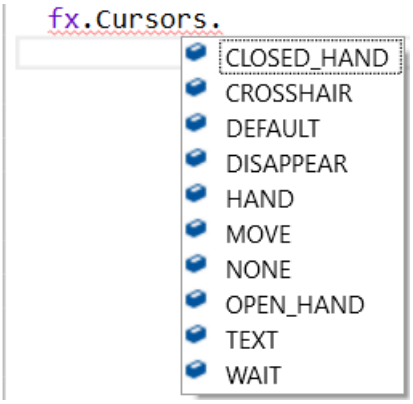
fx.LoadFont(Dir As String, FileName As String, Size As Double)

Loads a font resource from the given file. Returns an uninitialized font if loading has failed.
After the font was loaded you can call CreateFont with the font family name to create other variants of the given font.

```
Private MyFont As Font
MyFont = fx.LoadFont(File.DirAssets, "OLDENGL.TTF", 30)
lblTest2.Font = MyFont
```

11.2.6 fx.Curors

You can set different mouse cursors.
To define a Cursor, just type `fx.Cursor.` in the IDE and you get a popup window showing all the available cursors.



Example:

```
pneTest.MouseCursor = fx.Cursors.HAND
```

11.2.7 fx.LoadImage

Loads an image (similar to Image.Initialize), returns an Image object.

11.2.8 LoadImageSample

Loads a resized image (like Image.InitializeSample), returns an Image object.

LoadImageSample(Dir As String, FileName As String, Width As Double, Height As Double)

11.2.9 fx.MsgBox / fx.MsgBox2

Shows a modal message box, similar to the ones in B4A.
The program executes the `fx.Msgbox` or `fx.Msgbox2` call and stops at this line, waiting for an answer from the user.

11.2.9.1 fx.MsgBox

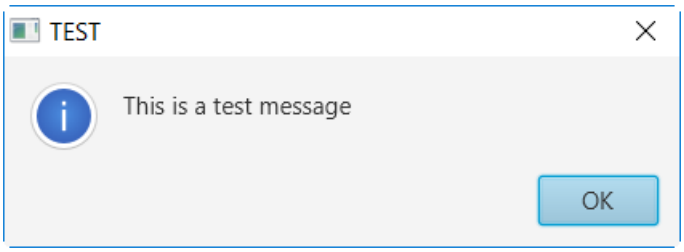
Shows a message box with an OK button, no return value.

Msgbox(Owner As Form, Message As String, Title As String)

Owner = Owner form, most of the time MainForm, or Null.

Example:

```
fx.Msgbox(MainForm, "This is a test message", "TEST")
```



11.2.9.2 fx.MsgBox2

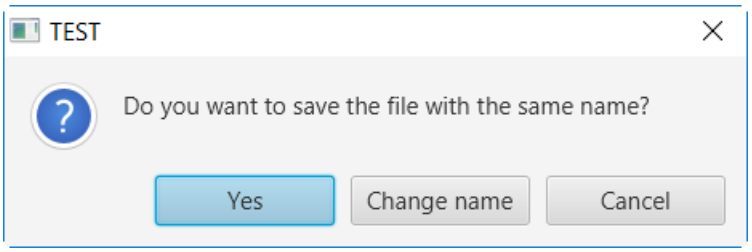
Shows a message box with up to three buttons, positive, negative and cancel.
It returns a value depending on the pressed button.
The return value can be checked with `fx.DialogResponse.POSITIVE`, `fx.DialogResponse.NEGATIVE`, `fx.DialogResponse.CANCEL`.

Msgbox2(Owner As Form, Message As String, Title As String, Positive As String, Cancel As String, Negative As String, Style As Object)
Owner = Owner form, most of the time MainForm.
Positive = Text for a positive answer.
Cancel = Text for a cancel answer.
Negative = Text for a negative answer.
Style = One of the style images.

- fx.MSGBOX_CONFIRMATION  .
- fx.MSGBOX_ERROR  .
- fx.MSGBOX_INFORMATION  .
- fx.MSGBOX_NONE .
- fx.MSGBOX_WARNING  .

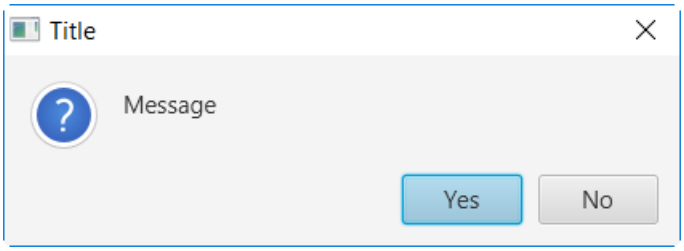
Example:

```
Private Answer As Int
Answer = fx.Msgbox2(MainForm, "Do you want to save the file with the same name?",
"TEST", "Yes", "Cancel", "Change name", fx.MSGBOX_CONFIRMATION)
If Answer = fx.DialogResponse.POSITIVE Then
   'save the file
Else If Answer = fx.DialogResponse.NEGATIVE Then
   'change the file name and save
End If
```



Note : The order of the button texts in the calling function can be different than in the displayed message box on some computers like mine.

Clicking on  is the same as clicking the Cancel button.



You can enter an empty string “” to hide a button.

```
Answer = fx.Msgbox2(MainForm, "Message", "Title", "Yes", "", "No", fx.MSGBOX_CONFIRMATION)
```

11.2.10 fx.InputList

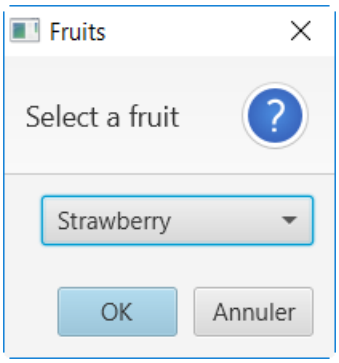
Shows an input list allowing the user to select predefined items.

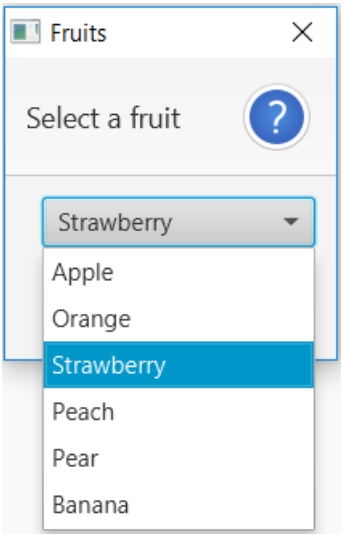
Returns the index of the selected item, or the Cancel index (-3)

InputList(Owner As Form, Items As List, Message As String, Title As String, DefaultItem As Int)
Owner = Owner form, most of the time MainForm.
Items = List of the items.
DefaultItem = Index of the default item.

Example:

```
Private lstFruits As List
Private Fruit As String
Private Answer As Int
lstFruits.Initialize
lstFruits.AddAll(Array As String("Apple", "Orange", "Strawberry", "Peach", "Pear",
"Banana"))
Answer = fx.InputList(MainForm, lstFruits, "Select a fruit", "Fruits", 2)
If Answer >= 0 Then  'checks if an item was selected
   Fruit = lstFruits.Get(Answer)
End If
```





11.2.11 fx. ShowExternalDocument

You can show external documents, files or internet sites.

Examples:

```
Show a pdf file from the PC:
Private FolderName As String
'you can chage the FolderName
'FolderName = "D:\B4J\BeginnersGuide\V1_0\SourceCode\jFX"
FolderName = File.DirAssets
fx.ShowExternalDocument(File.GetUri(FolderName, "Test.pdf"))
```

Show an internet site:

```
fx.ShowExternalDocument("http://www.basic4ppc.com")
```

Show an Excel file:

```
Private FolderName As String
'you can chage the FolderName
'FolderName = "D:\B4J\BeginnersGuide\V1_0\SourceCode\jFX"
FolderName = File.DirAssets
fx.ShowExternalDocument(File.GetUri(FolderName, "Test.xls"))
```

Show a B4J program:

```
Private FolderName As String
'you can chage the FolderName
'FolderName = "D:\B4J\BeginnersGuide\V1_0\SourceCode\Pong\Objects"
FolderName = File.DirAssets
fx.ShowExternalDocument(File.GetUri(FolderName, "Pong.jar"))
```

11.2.12 fx.PrimaryScreen

Returns the primary screen as a Screen object.

Example:

```
Private scrPrimary As Screen
scrPrimary = fx.PrimaryScreen
```

11.2.13 fx.Screens

Returns a List with all the screens. Each element in the list is a Screen object.

Example:

```
Private lstScreens As List
lstScreens.Initialize
lstScreens = fx.Screens
Private scrTest As Screen
scrTest = lstScreens.Get(0)
Log(scrTest.MaxX)
```

12 User Interface Nodes

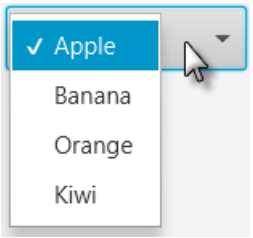
In this chapter, B4J specific User Interface Nodes are explained.

The test project UINodes is in the UINodes folder in SourceFolder.

12.1 ChoiceBox

ChoiceBox is a simplified version of ComboBox. It is optimized for a small number of options.
It can only show string items.

Click on the ChoiceBox to open it.



12.1.1 Specific properties

Items
Returns a List with the items. Read only.

SelectedIndex
Gets or sets the selected index. A value of -1 means that there is no selected item.

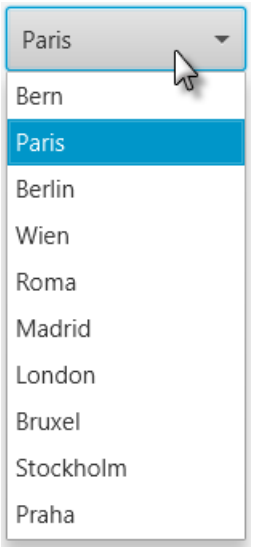
12.1.2 Specific events

SelectedIndexChanged (Index As Int, Value As Object)
Index = index of the selected item.
Value = value of the selected item.

12.2 ComboBox

ComboBox control. Allows the user to select a value from a dropdown list.
If the ComboBox is editable then the user can also write any value instead of selecting a predefined value.
SelectedIndexChanged event is raised when the selected index is changed. The Index will be -1 if there is no selection.
ValueChanged event is raised when the current value has changed. This event will be raised when the selected item has changed or
if the ComboBox is editable then it will be raised when the user has edited the value.

Click on the ComboBox to open it.

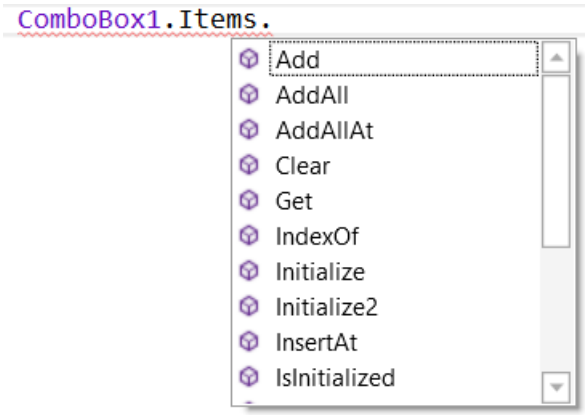


12.2.1 Specific properties

Editable
Gets or sets whether the ComboBox is editable. The default value is False.

Items
Returns a List with the items. Read only.

Items has all the properties and method as a List!



To add items use the Add, AddAll, or AddAllAt methods.
Example:

```
ComboBox1.Items.AddAll(Array As String("Bern", "Paris", "Berlin","Wien", "Roma",
"Madrid", "London", "Bruxel", "Stockholm", "Praha"))
```

SelectedIndex
Gets or sets the selected index. A value of -1 means that there is no selected item.

Value
Gets or sets the current value.

12.2.2 Specific events

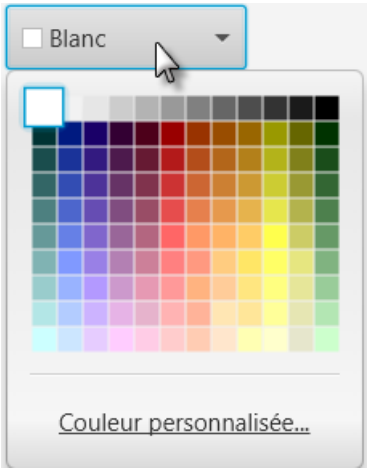
SelectedIndexChanged (Index As Int, Value As Object)
Index = index of the selected item.
Value = value of the selected item.

ValueChanged (Value As Object)

12.3 ColorPicker

You can use the ColorPicker to let the user select a color.

Click on the ColorPicker to open it.



12.3.1 Specific properties

SelectedColor
Gets or sets the current value, as a Paint object.

```
ColorPicker1.SelectedColor = fx.Colors.Red
```

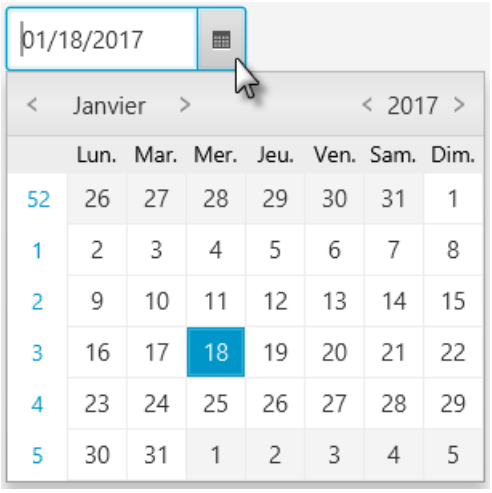
12.3.2 Specific events

ValueChanged (Value As Paint)
Gets or sets the current value, as a Paint object.

12.4 DatePicker

You can use the DatePicker to let the user select a date.

Click on  to open the DatePicker



12.4.1 Specific properties

DateFormat
Sets the date format. The default value is `"dd/MM/yyyy"`.

```
DatePicker1.DateFormat = "yyyy/MM/dd"
```

DateTicks
Gets or sets the date (as ticks). Pass 0 to clear the value.

```
DatePicker1.DateTicks = DateTime.DateParse("2017/04/25")
```

Editable
Gets or sets whether the field is editable. The default value is True.

```
DatePicker1.Editable = False
```

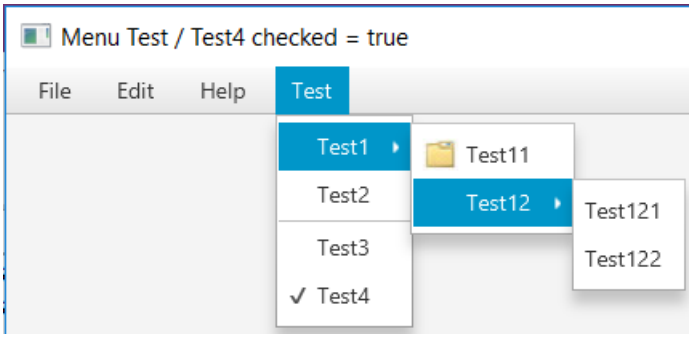
12.4.2 Specific events

ValueChanged (Value As Long)
Returns the selected date in Ticks.

12.5 MenuBar / Menu / MenuItem / CheckMenuItem

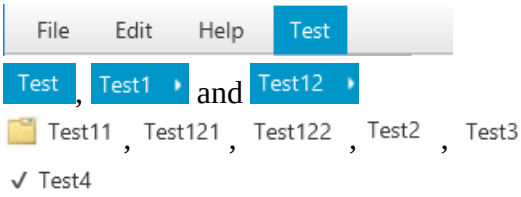
The menu management is different from the B4A menu management.

In the test program, the File / Edit / Help menus are entered in the Designer and the Test menu is added in the code.



Objects:

- MenuBar
- Menu
- MenuItem
- CheckMenuItem



Structure of the MenuBar above, only the open part is explained:

- MenuBar
  - Menu File
  - Menu Edit
  - Menu Help
  - Menu Test
    - Menu Test1
      - MenuItem Test11 with an image
      - Menu Test12
        - MenuItem Test121
        - MenuItem Test122
    - MenuItem Test2
    - MenuItem Test3
    - CheckMenuItem Test4

A **MenuBar** has only Menu objects as child objects.
A **Menu** can have Menu, MenuItem and CheckMenuItem objects as child objects.
A **MenuItem** and **CheckMenuItem** have no child objects.

Menus in the top bar must have at least one child.

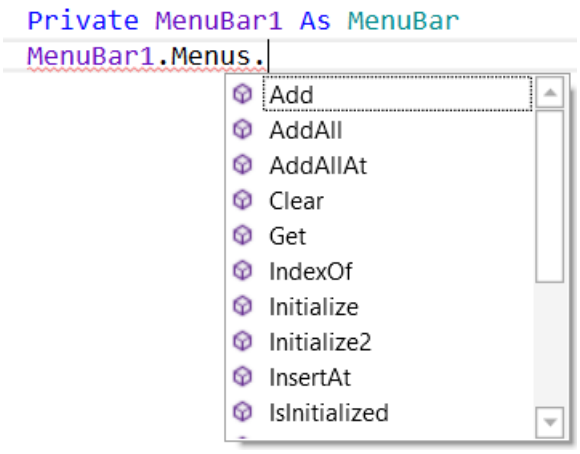
12.5.1 MenuBar

A MenuBar is a bar with menu items like: 

.

12.5.1.1 Properties

Menus
Returns a List with the Menu objects. Read only. You can modify this list and add new items.



MenuBar.Menus inherits the methods and properties of the List object.

12.5.1.2 Events

Action
Raised when the user clicks on a MenuItem with no specific event name.
You can check which MenuItem raised the event with the Sender object.

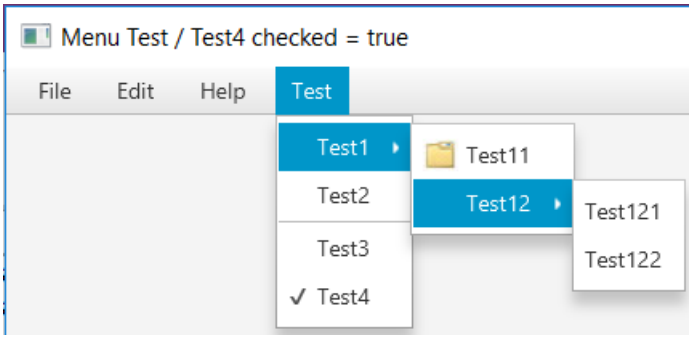
```
Private Sub MenuBar1_Action
   Private mnu As MenuItem

   mnu = Sender

   Select mnu.Text
   Case "_New"
      MainForm.Title = "Menu File / New"
   Case "_Save"
      MainForm.Title = "Menu File / Save"
   End Select
End Sub
```

SelectedChange (Selected As Boolean)
Raised when a selection in a CheckMenuItem has changed.

```
Private Sub MenuBar1_SelectedChange(Selected As Boolean)
   Private cmi As CheckMenuItem

   cmi = Sender

   Select cmi.Text
   Case "Checked Item"
       MainForm.Title = "Menu File / Checked Item selection" & cmi.Selected
   End Select
End Sub
```

12.5.2 Menu

The Menu object can hold Menu objects for sub menus or MenuItem or CheckMenuItems objects.



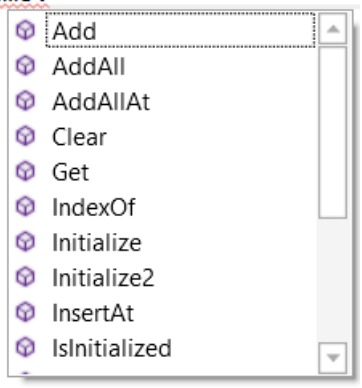
Test Menu object in the MenuBar.

Test1 and Test12
Menu objects in a parent Menu object.

12.5.2.1 Properties

MenuItems
Returns a List with the child items. You can modify this list and add new items.

```
Private mnuTest1 As Menu
mnuTest1.MenuItems.
```



Menu.MenuItems inherits the methods and properties of the List object.

MenuItems can hold following objects:

- Menu
- MenuItem
- CheckMenuItem
- SeparatorMenuItem

Tag
Gets or sets the tag object tied to this menu. Same as standard nodes.

Text
Gets or sets the menu text. Same as standard nodes.
An underscore "_" in the text like "_File" shows an underscored character in the menu File, activated with the Alt key.

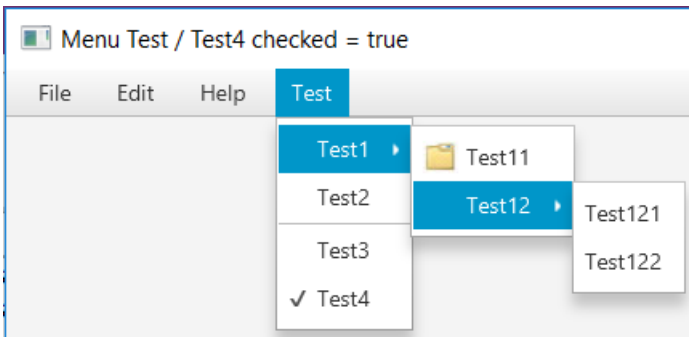
12.5.2.2 Methods

Initialize (Text As String, EventName As String)
Initializes the Menu object with the text to display and the event name. Same as standard nodes.

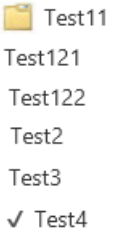
IsInitialized
Returns a Boolean. Same as standard nodes.

12.5.3 MenuItem

MenuItems are the objects for the menu selections,



like



12.5.3.1 Properties

Enabled
Gets or sets whether the menu item is enabled.

Image
Gets or sets the image that is displayed before the text.
The image size is typically 16x16.

Tag
Gets or sets the tag object tied to this menu. Same as standard nodes.

Text
Gets or sets the menu text. Same as standard nodes.
An underscore "_" in the text like "_File" shows an underscored character in the menu File, activated with the Alt key.

Visible
Gets or sets whether the menu item is visible.

12.5.3.2 Methods

Initialize(Text As String, EventName As String)
Initializes the Menu object with the text to display and the event name. Same as standard nodes.

IsInitialized
Returns a Boolean. Same as standard nodes.

12.5.3.3 Events

Action
Raised when the user clicks on the MenuItem.

12.5.4 CheckMenuItem

CheckMenuItems are the same as MenuItems but with a check box before the text.

They have one event more when the checkbox selection changes.

12.5.4.1 Events

SelectedChange (Selected As Boolean)
Raised when a selection has changed.

12.5.5 SeparatorMenuItem

SeparatorMenuItems can be defined in the Designer with a minus sign "-", in the JSON code.

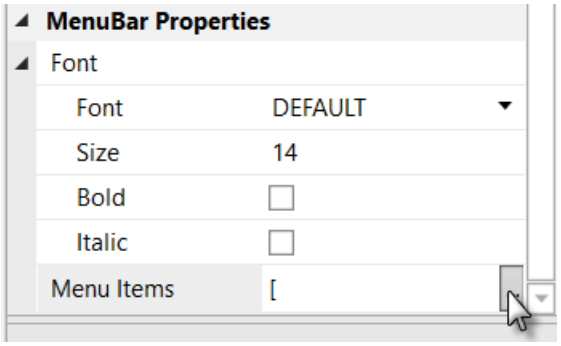
They are not supported directly in B4J, but if you want to add menus in the code you can use the JavaObject library.

```
Private mnuSeparation As JavaObject
mnuSeparation.InitializeNewInstance("javafx.scene.control.SeparatorMenuItem", Null)
mnuTest.MenuItems.Add(mnuSeparation)
```

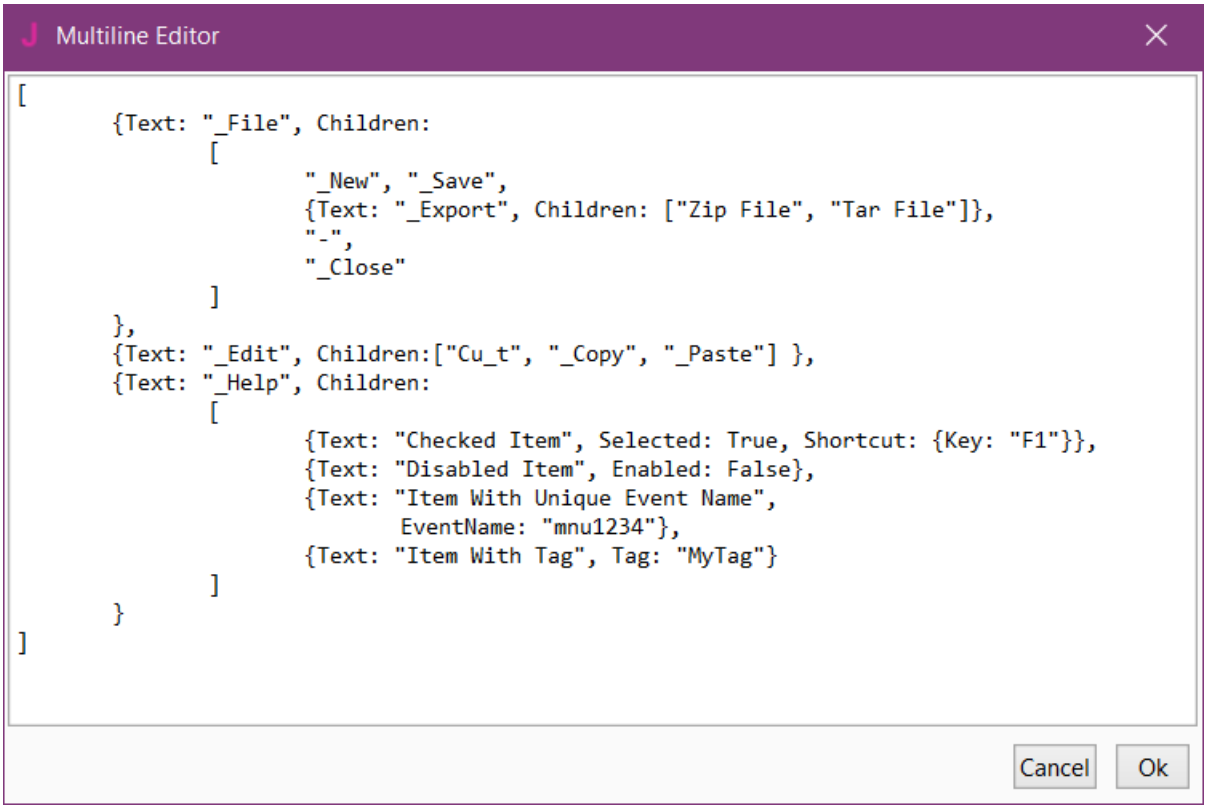
12.5.6 Menu added in the Designer

You can define menus in the Designer.

Add a MenuBar to the layout, position and resize it, and then in the Properties window, at the bottom, open the MenuEditor.



The File / Edit / Help menus are by default in the editor. In the Designer, menus are JSON strings.



The following fields are available (note that the field names are case sensitive):

Text - The item's text. This is the only required field. Underscore sets the mnemonic character.
EventName - Sets the sub that will handle the events. If not set, then the EventName will be the same as the control's event name property.
Tag - An arbitrary string that will be set as the item's tag.
Children - Holds a list of child menu items.
Selected - Sets whether the menu item is checked or not.
Image - Image file name.
ShortCut - Adds a shortcut at the end of the text.

Examples:
A single Menu with three MenuItems:

The entire file has square brackets [ ] at the beginning and the end.
A Menu object is between brackets {}.
A Children list is between square brackets [ ].

On the top line:

```
[
    {Text: "_File", Children:
        [
            "_New", "_Save",
            "-",
            "_Close"
        ]
    }
]
```

The short way:

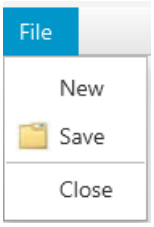
```
[ {Text: "_File", Children: ["_New", "_Save", "-", "_Close"] } ]
```

`{Text: "_File"` adds a Menu object with the text `"_File"`.

And `Children: ["_New", "_Save", "-", "_Close"]` adds the children New, Save, Separator and Close.

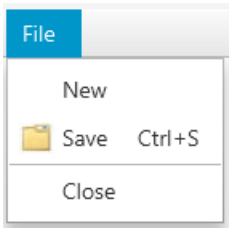
Now we add an image to Save

```
[
    {Text: "_File", Children:
        [
            "_New",
            {Text: "_Save",
            Image: "FolderClosed.jpg"},
            "-",
            "_Close"
        ]
    }
]
```



We add an event name, a Tag and a shortcut to Save:

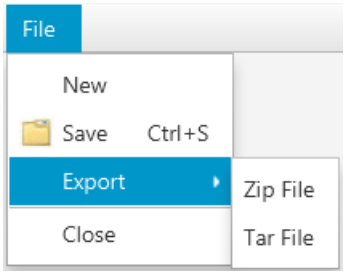
```
[
    {Text: "_File", Children:
        [
             "_New",
             {Text: "_Save",
             EventName: "mnuSave",
             Image: "FolderClosed.jpg",
             Tag: "mnuSave",
             Shortcut: {Key: "S", Modifier: "CONTROL"}
             },
             "-",
             "_Close"
        ]
    }
]
```



Shortcuts `Shortcut: {Key: "S", Modifier: "CONTROL"}`:
Key = any key on the keyboard, like “S”, “F1” etc.
Modifier = an additional control key (ALT, CONTROL, SHIFT).

We add an Export menu item with children:

```
[
    {Text: "_File", Children:
        [
           "_New",
           {Text: "_Save",
           EventName: "mnuSave",
           Image: "FolderClosed.jpg",
           Tag: "mnuSave",
           Shortcut: {Key: "S", Modifier: "CONTROL"}
           },
           {Text: "_Export", Children: ["Zip File", "Tar File"]},
           "-",
           "_Close"
        ]
    }
]
```



Adding Selected is like adding the others, example in the next page.

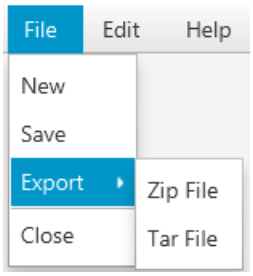
The entire code of the default Menu in the Designer:

```
[
    {Text: "_File", Children:
        [
            "_New", "_Save",
            {Text: "_Export", Children: ["Zip File", "Tar File"]},
            "-",
            "_Close"
        ]
    },
    {Text: "_Edit", Children:["Cu_t", "_Copy", "_Paste"] },
    {Text: "_Help", Children:
        [
            {Text: "Checked Item", Selected: True, Shortcut: {Key: "F1"}},
            {Text: "Disabled Item", Enabled: False},
            {Text: "Item With Unique Event Name",
                EventName: "mnu1234"},
            {Text: "Item With Tag", Tag: "MyTag"}
        ]
    }
]
```

We have:

```
    {Text: "_File", Children:
        [
            "_New", "_Save",
            {Text: "_Export", Children: ["Zip File", "Tar File"]},
            "-",
            "_Close"
        ]
    },
```

This is the code for:



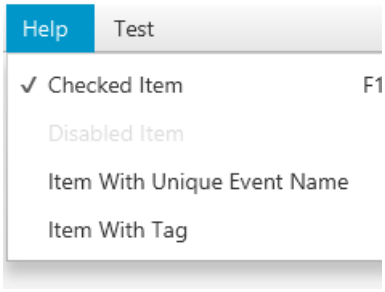
- `Text: "_File"`, is the File Menu.
- `[ "_New", "_Save",` are the *New* and *Save* MenuItems.
- `{Text: "_Export", Children: ["Zip File", "Tar File"]},` is the *Export* Menu and the two *Zip File* and *Tar File* MenuItems.
- `"-",` is the Separator line, just below Export.
- `"_Close"` is the *Close* MenuItem.

The *Edit* Menu and its three children *Cut*, *Copy*, *Paste*

```
{Text: "_Edit", Children:["Cu_t", "_Copy", "_Paste"] }
```

And the *Help* Menu.

```
{Text: "_Help", Children:
    [
        {Text: "Checked Item", Selected: True, Shortcut: {Key: "F1"}},
        {Text: "Disabled Item", Enabled: False},
        {Text: "Item With Unique Event Name",
            EventName: "mnu1234"},
             {Text: "Item With Tag", Tag: "MyTag"}
```

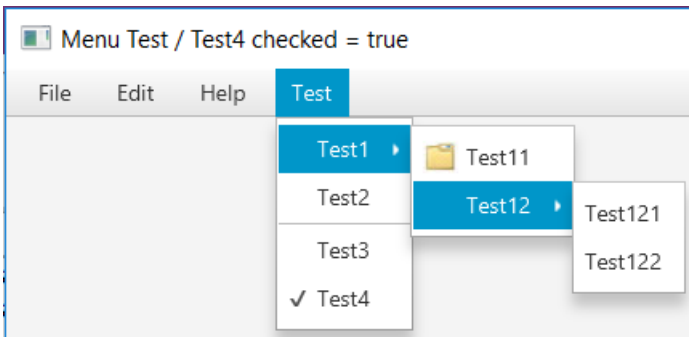


- `{Text: "_Help", Children:` is the *Help* Menu whith children.
- `{Text: "Checked Item", Selected: True, Shortcut: {Key: "F1"}},`
  is the *Checked Item* CheckMenuItem which is selected by default and has the shortcut *F1*.
- is the *Disabled Item* MenuItem, disabled by default.
- `{Text: "Item With Unique Event Name",`
  `EventName: "mnu1234"},`
  is the *Item With Unique Event Name* MenuItem with `"mnu1234"` as the EventName.
- `{Text: "Item With Tag", Tag: "MyTag"}` is the *Item With* Tag MenuItem.

12.5.7 Menu added in the code

The top object is a MenuBar, it can be added in the code or in the Designer.

In the UINodes example program, the MenuBar is added in the Designer.
And we add the Test menu in the code.



We:

- Define the Test menu `mnuTest`.

```
Private Sub InitMenuBar
   Private mnuTest As Menu
   mnuTest.Initialize("Test", "")
```

- Define the Test1 Menu object `mnuTest1` and add it to the Test menu.

```
   Private mnuTest1 As Menu
   mnuTest1.Initialize("Test1", "")
   mnuTest.MenuItems.Add(mnuTest1)
```

- Define the Test11 MenuItem `mnuTest11` with an image, and add it to the Test1 menu.

```
   Private mnuTest11 As MenuItem
   mnuTest11.Initialize("Test11", "mnuTest")
   mnuTest11.Tag = "Test11"
   mnuTest11.Image = imgFolderClosed
   mnuTest1.MenuItems.Add(mnuTest11)
```

- Define the Test12 MenuItem `mnuTest12` and add it to the Test1 menu.

```
   Private mnuTest12 As Menu
   mnuTest12.Initialize("Test12", "")
   mnuTest12.Tag = "Test12"
   mnuTest1.MenuItems.Add(mnuTest12)
```

- Define the Test121 MenuItem `mnuTest121` and add it to the Test12 menu.

```
   Private mnuTest121 As MenuItem
   mnuTest121.Initialize("Test121", "mnuTest")
   mnuTest121.Tag = "Test121"
   mnuTest12.MenuItems.Add(mnuTest121)
```

- Define the Test122 MenuItem `mnuTest122` and add it to the Test12 menu.

```
Private mnuTest122 As MenuItem
mnuTest122.Initialize("Test122", "mnuTest")
mnuTest122.Tag = "Test122"
mnuTest12.MenuItems.Add(mnuTest122)
```

- Define the Test2 MenuItem `mnuTest2` and add it to the Test menu.

```
Private mnuTest2 As MenuItem
mnuTest2.Initialize("Test2", "mnuTest")
mnuTest2.Tag = "Test2"
mnuTest.MenuItems.Add(mnuTest2)
```

- Define SeparatorMenuItem `mnuSeparation` and add it to the Test menu.
  For this we use the JavaObject library.

```
Private mnuSeparation As JavaObject
mnuSeparation.InitializeNewInstance("javafx.scene.control.SeparatorMenuItem", Null)
mnuTest.MenuItems.Add(mnuSeparation)
```

- Define the Test3 MenuItem `mnuTest3` and add it to the Test menu.

```
Private mnuTest3 As MenuItem
mnuTest3.Initialize("Test3", "mnuTest")
mnuTest3.Tag = "Test3"
mnuTest.MenuItems.Add(mnuTest3)
```

- Define the Test4 CheckMenuItem `mnuTest4` and add it to the Test menu.

```
Private mnuTest4 As CheckMenuItem
mnuTest4.Initialize("Test4", "mnuTest4")
mnuTest4.Tag = "Test4"
mnuTest4.Selected = True
mnuTest.MenuItems.Add(mnuTest4)
```

- And finally add the Test menu to the MenuBar.

```
MenuBar1.Menus.Add(mnuTest)
```

We used the same event name `"mnuTest"` for all MenuItems and a specific one for the CheckMenuItem `"mnuTest4"`.

12.5.8 Menu events

Here we have look at the Menu events in the UINodes program.

MenuBar1_Action routine.
We get the MenuItem which raised the event and check its Text property.

```
'This event handles all menu items without a specific event name
Private Sub MenuBar1_Action
   Private mnu As MenuItem

   mnu = Sender  'get the MenuItem which raised the event

   Select mnu.Text
   Case "_New"
      MainForm.Title = "Menu File / New"
   Case "_Save"
      MainForm.Title = "Menu File / Save"
   Case "Checked Item"
      MainForm.Title = "Menu File / Checked Item"
   Case "_Close"
      MainForm.Title = "Menu File / Close"
   Case "Cu_t"
      MainForm.Title = "Menu Edit / Cut"
   Case "_Copy"
      MainForm.Title = "Menu Edit / Copy"
   Case "_Paste"
      MainForm.Title = "Menu Edit / Paste"
   Case "Item With Tag"
      MainForm.Title = "Menu Help / Item With Tag"
   End Select
End Sub
```

MenuBar1_SelectedChange routine.

The second routine is MenuBar1_SelectedChange.

```
Private Sub MenuBar1_SelectedChange(Selected As Boolean)
   Private cmi As CheckMenuItem

   cmi = Sender

   Select cmi.Text
   Case "Checked Item"
      MainForm.Title = "Menu File / Checked Item selection" & cmi.Selected
   End Select
End Sub
```

mnuTest_Action routine.

The event routine for the Test menu.
This routine is like the MenuBar1_Action routine, the difference is that here we check the Tag property instead of the Text property.

```
Private Sub mnuTest_Action
   Private mnu As MenuItem

   mnu = Sender

   Select mnu.Tag
   Case ("Test11")
       MainForm.Title = "Menu Test / Test11"
   Case ("Test121")
       MainForm.Title = "Menu Test / Test121"
   Case ("Test122")
       MainForm.Title = "Menu Test / Test122"
   Case ("Test2")
       MainForm.Title = "Menu Test / Test2"
   Case ("Test3")
       MainForm.Title = "Menu Test / Test3"
   End Select
End Sub
```

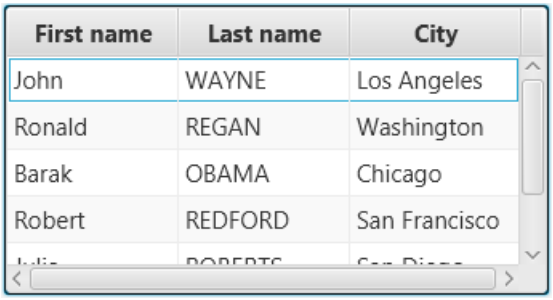
mnuTest4_SelectedChange routine.

The Test4 CheckMenuItem SelectedChange routine is a specific routine because we defined a specific event name for it.

```
Private Sub mnuTest4_SelectedChange (Selected As Boolean)
   MainForm.Title = "Menu Test / Test4 checked = " & Selected
End Sub
```

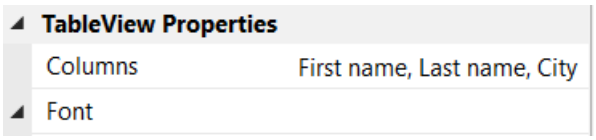
12.6 TableView

A control that shows data in a table.
The table data is stored in a List. Each item in the list (which represents a row) is an array of objects. One object for each column.
Changing the data in the list will change the data in the table.
Set SingleCellSelection to True if you want to allow selection of single cells instead of rows.



The column headers can be changed in the Designer.

Default



changed.

12.6.1 Specific methods

GetColumnHeader (Index As Int)
Gets the column header. Index - Column index (first column index is 0).

GetColumnVisible (Index As Int)
Gets whether the column is visible, returns a Boolean.

GetColumnWidth (Index As Int)
Gets the column width. Index - Column index (first column index is 0).

ScrollTo (Index As Int)
Scrolls to the given row index.

SelectCell (Row As Int, Column As Int)
Selects a single cell. Make sure to first set SingleCellSelection to True.

SetColumnHeader (Index As Int, Header As String)
Sets the column header. Index - Column index (first column index is 0).

SetColumns (Columns As List)
Sets the table columns. Columns - A List (or array) with the columns titles.

SetColumnSortable (Index As Int, Sortable As Boolean)
Sets whether the column is sortable, True by default.
Sortable means that the user can click on the header to sort it.

SetColumnVisible (Index As Int, Visible As Boolean)
Sets whether the column is visible.

SetColumnWidth (Index As Int, Width As Double)
Sets the column width. Index - Column index (first column index is 0).

12.6.2 Specific properties

ColumnsCount
Returns the number of columns, read only.

SelectedRow
Gets or sets the index of the selected row.

SelectedRowValues
Gets the values of the selected row or sets the selected row based on the given values, as an array of objects.

SingleCellSelection
Gets or sets whether single cell selection is enabled.

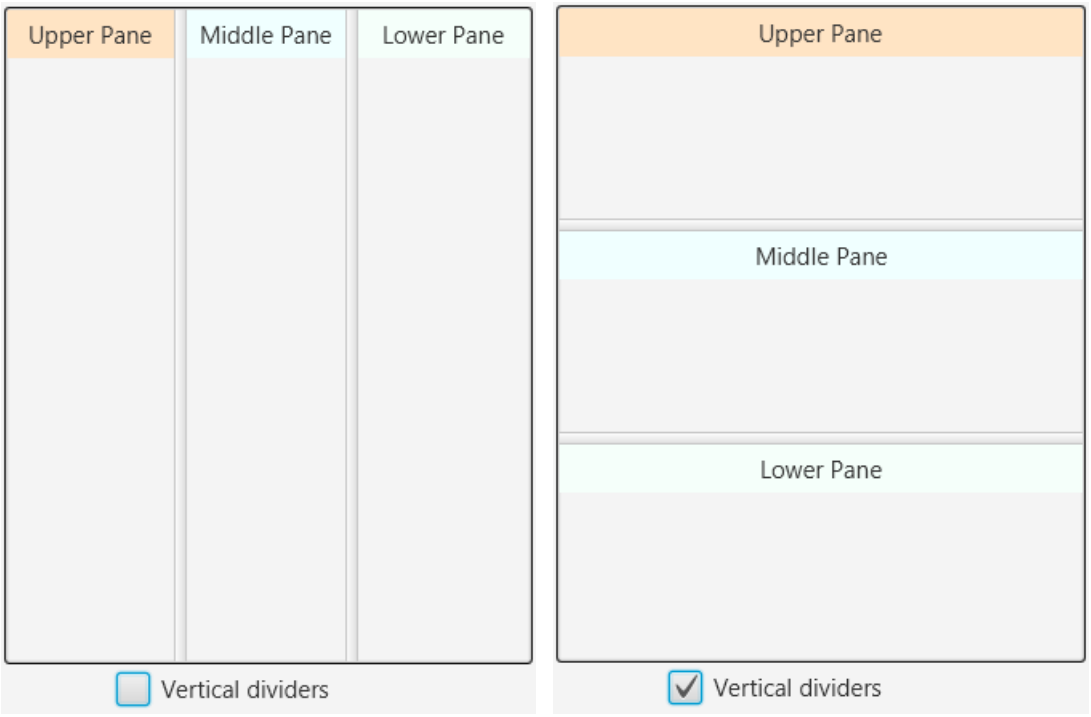
12.6.3 Specific events

SelectedCellChanged (RowIndex As Int, ColIndex As Int, Cell As Object)
Returns the row index, the column index and the cell object.
With the SingleCellSelection property set to True.

SelectedRowChanged (Index As Int, Row() As Object)
Returns the row index and an array of the row objects.
With the SingleCellSelection property set to False.

12.7 SplitPane

Pane with sizable parts, the orientation can be either horizontal or vertical.
The number of parts is defined by the number of loaded layouts, 3 in the example.



12.7.1 Specific methods

LoadLayout (LayoutFile As String)
Load a layout file, the number of loaded layouts defines the number of parts.

SetSizeLimits (LayoutIndex As Int, MinSize As Double, MaxSize As Double)
Sets the layout minimum and maximum sizes.
MinSize - The layout minimum size in pixels (width for horizontal orientation and height for vertical orientation).
MaxSize - The layout maximum size in pixels. Pass 0 for no limit.

```
SplitPane1.SetSizeLimits(0, 100, 0)
```

12.7.2 Specific properties

DividerPositions
Gets or sets the dividers positions. The positions are stored in an array of doubles.
Each value should be between 0 to 1.

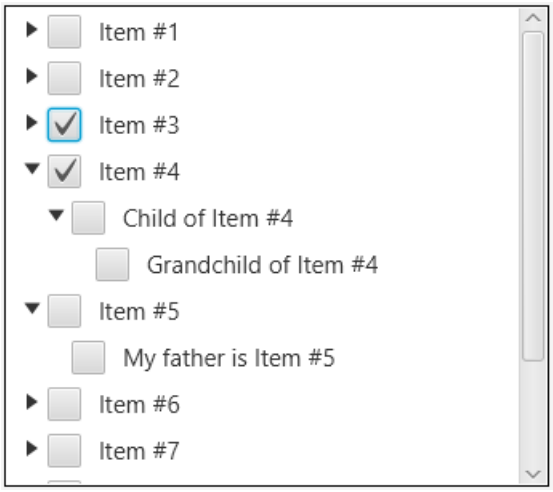
```
SplitPane1.DividerPositions = Array As Double(0.33, 0.66)
```

Vertical
Gets or sets the divider orientations.

12.8 TreeView / TreeItem

Displays a hierarchical collection of labeled items, each represented by a TreeItem.
TreeItems can have images or CheckBoxes before the text.

Standard with images



With CheckBoxes

12.8.1 TreeView

Displays a hierarchical collection of labeled items, each represented by a TreeItem.

12.8.1.1 Specific methods

SetCheckBoxesMode
Shows a checkbox before each item in the TreeView.
Items should be CheckboxTreeItems.

```
TreeView2.SetCheckBoxesMode
```

12.8.1.2 Specific events

SelectedItemChanged (SelectedItem As TreeItem)

12.8.2 TreeItem / CheckboxTreeItem

TreeItems and CheckboxTreeItem are the objects in the TreeView node.
CheckboxTreeItem have a checkbox before the text.

12.8.2.1 Specific properties

Children
Returns a list with the TreeItem children.

Expanded
Gets or sets whether the tree item is expanded.

Image
Gets or sets the image that is displayed before the text.

Parent
Returns the TreeItem parent. Will return an uninitialized TreeItem if this is a root item.

Root
Tests whether this TreeItem is a root item.

Text
Gets or sets the TreeItem text.

12.8.2.2 Specific events

ExpandedChanged (Expanded As Boolean)
Raised when a TreeItem or a CheckboxTreeItem expansion changes.

CheckedChange (Checked As Boolean)
Raised when a CheckboxTreeItem check changes.

13 Debugging

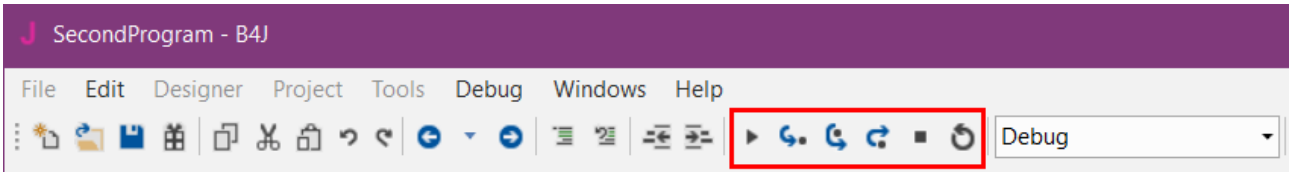
Debugging is an important part when developing.

The two major utilities for debugging are:

Breakpoints - You can mark lines of codes as breakpoints. This is done by pressing on the grey area left of the line.
The program will pause when it reaches a breakpoint and will allow you to inspect the current state.

Logging - The Logs tab at the right pane is very useful. It shows messages related to the components life cycle and it can also show messages that are printed with the Log keyword. You should press on the Connect button to connect to the device logs. Note that there is a Filter checkbox. When it is checked you will only see messages related to your program. When it is unchecked you will see all the messages running in the system. If you are encountering an error and do not see any relevant message in the log, it is worth unchecking the filter option and looking for an error message.
Note that the log is maintained by the device. When you connect to a device you will also see previous messages.

13.1 Debug Toolbar



Debug Toolbar:

▶	Run the program	F5	Runs the program, no action in Debug (rapid)
	Step In	F8	Executes the next statement.
	Step Over	F9	Executes a routine without jumping in it.
	Step Out	F10	Finishes executing the rest of a routine.
■	Stop		Stops the program.
	Restart	F11	Restarts the program.

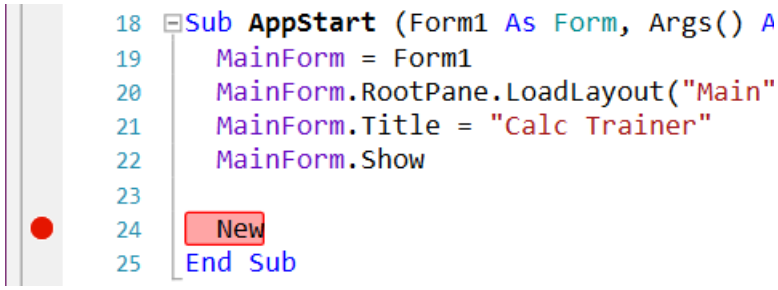
The examples below use the SecondProgram project.

13.1.1 Run ▶ F5

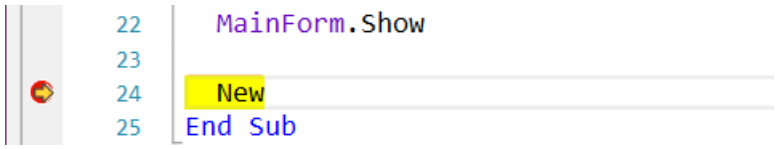
Runs the program,
If the program is stopped at a breakpoint the program runs until the next breakpoint or completes running.

13.1.2 Step In F8

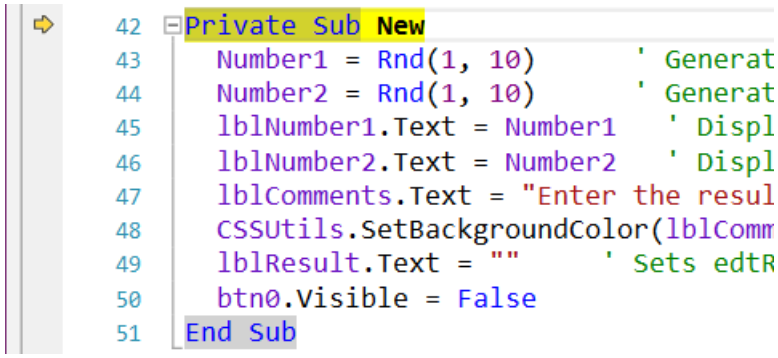
The debugger executes the code step by step.



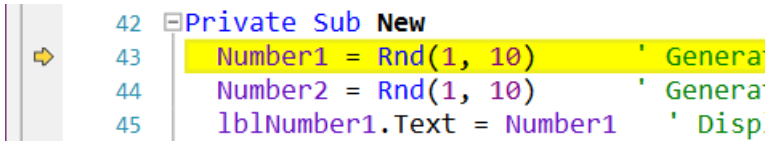
In the SecondProgram project we set a Breakpoint at line 24 `New`.



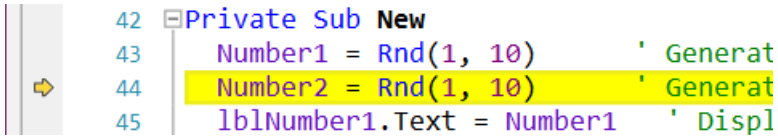
We run the program, it will stop executing at line 24 `New`.



Click on .
The debugger executes the next line, `Sub New` in this case.



Click once more on .
The debugger executes the next line, `Number1 =...`



Click once more on .
The debugger executes the next line, `Number2 =...`

13.1.3 Step Over F9

If the current line is a sub calling line the debugger executes the code in this subroutine and jumps to the line after the calling line.

```
18 Sub AppStart (Form1 As Form, Args() A
19   MainForm = Form1
20   MainForm.RootPane.LoadLayout("Main"
21   MainForm.Title = "Calc Trainer"
22   MainForm.Show
23
24   New
25 End Sub
```

In the SecondProgram project we set a Breakpoint at line 24 `New`.

```
22   MainForm.Show
23
24   New
25 End Sub
```

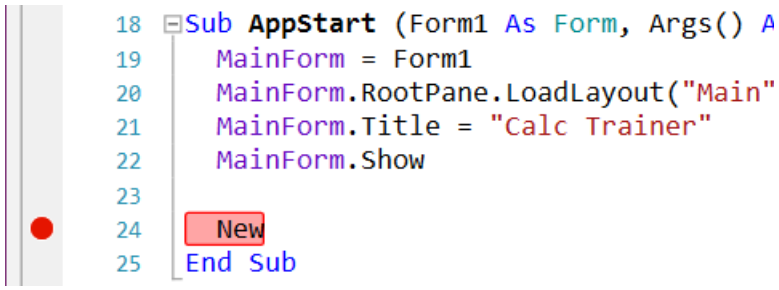
We run the program, it will stop executing at line 24 `New`.

```
21   MainForm.Title = "Calc Trainer"
22   MainForm.Show
23
24   New
25 End Sub
```

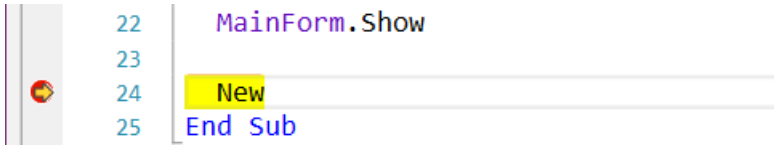
Click on .
The debugger executes the code in `New` and jumps directly to the next line which is
`End Sub` of `AppStart`.

13.1.4 Step Out F10

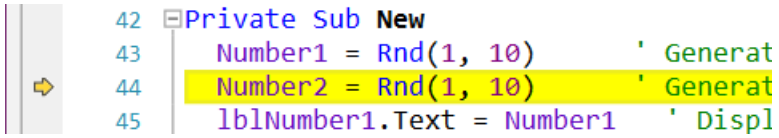
If the current line is a sub calling line the debugger executes the code in this subroutine and jumps to the line after the calling line.



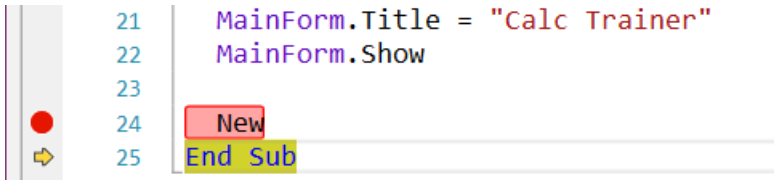
In the SecondProgram project we set a Breakpoint at line 24 `New`.



We run the program, it will stop executing at line 24 `New`.



We go step by step with to a line in the subroutine.



Click on .
The debugger executes the rest of the code in the subroutine and jumps to the next line which is `End Sub` of `AppStart`.

13.1.5 Stop

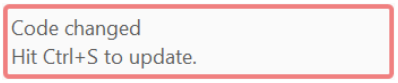
Stops the program and leaves the Rapid Debugger.

13.1.6 Restart F11

Restarts the program remaining in the Rapid Debugger.
Executes Process_Globals, AppStart and reloads the layout.

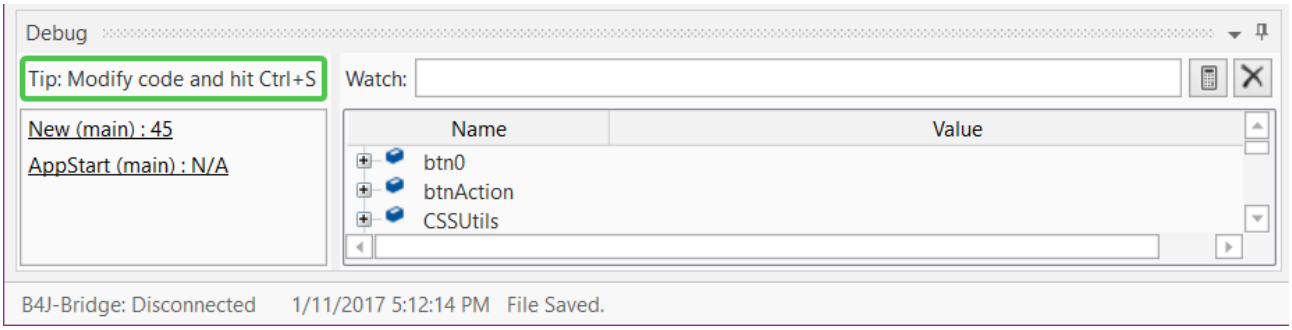
This is useful if you changed a layout file.

It is different from



explained in the next chapter.

13.2 Debug window



In the debug window we have (example with the SecondProgram, and a breakpoint in line 45:

13.2.1 The status button

Tip: Modify code and hit Ctrl+S

Shows that the program is running, the button border is green.

Code changed
Hit Ctrl+S to update.

When you change the code the button border changes to red.
To update the code click on the button or hit Ctrl + S.

13.2.2 The breakpoint window

New (main) : 45
AppStart (main) : N/A

The breakpoint window shows where the program has stopped.

```
42  Private Sub New
43    Number1 = Rnd(1, 10)       ' Generates
44    Number2 = Rnd(1, 10)       ' Generates
45    lblNumber1.Text = Number1    ' Displays
46    lblNumber2.Text = Number2    ' Displays
47    lblComments.Text = "Enter the result"
48    CSSUtils.SetBackgroundColor(lblComment
49    lblResult.Text = ""      ' Sets edtResu
50    btn0.Visible = False
51  End Sub
```

New (main) : 45

The program stopped in line 45, in routine New in the main module.

```
18  Sub AppStart (Form1 As Form, Args() A
19    MainForm = Form1
20    MainForm.RootPane.LoadLayout("Main"
21    MainForm.Title = "Calc Trainer"
22    MainForm.Show
23
24    New
25  End Sub
```

AppStart (main) : N/A

The calling routine is AppStart, and the calling line is not shown.

New (main) : 46
AppStart (main) : N/A

When you click on one of the lines the cursor jumps to that line.

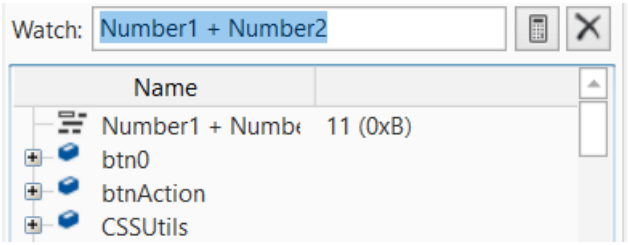
13.2.3 The Watch window

The Watch window allows to check more complex functions for testing and debugging.

```
42  Private Sub New
43      Number1 = Rnd(1, 10)
44      Log(Number1)
45      Number2 = Rnd(1, 10)
46      Log(Number2)
47      lblNumber1.Text = Number1
48      lblNumber2.Text = Number2
49      lblComments.Text = "Enter
```

In the SecondProgram code add two Log lines and set a breakpoint in line 47.

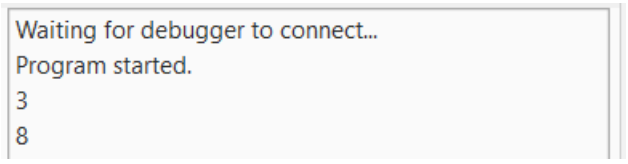
Run the program.



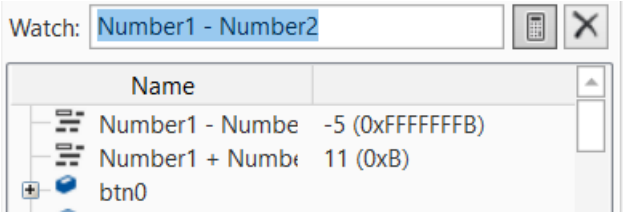
In the Add Watch field enter:

`Number1 + Number2`

Click on  to show the result on top of the list.



As we left the two Log lines in the code we still see the values of Number1 and Number2.

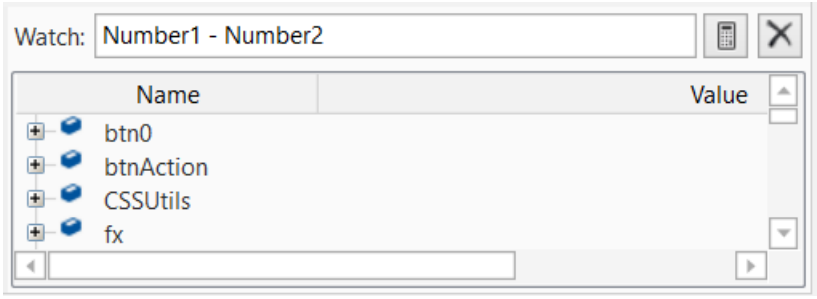


You can enter a new watch line
`Number1 + Number2`
and show it.

Click on  to remove the watch functions. This removes all the functions.

We could, of course, also have done this test with a Log.

13.2.4 The object window



Shows all variables and objects in the list ordered by alphabetical order.

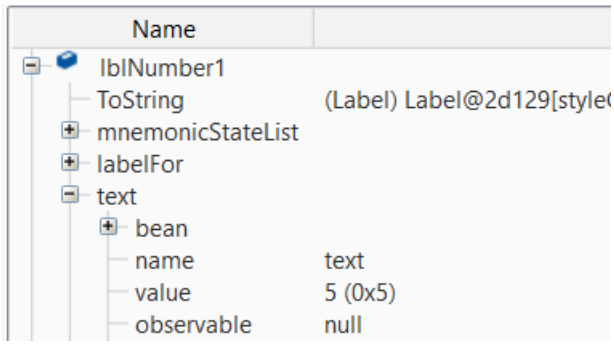
Click on ⊞ to show the details of the object:

Examples:

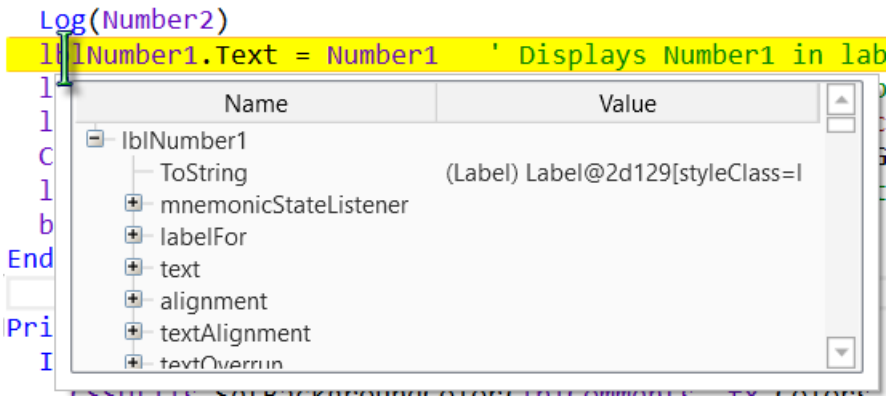
- Number1

Shows the current value (3).

- lblNumber1

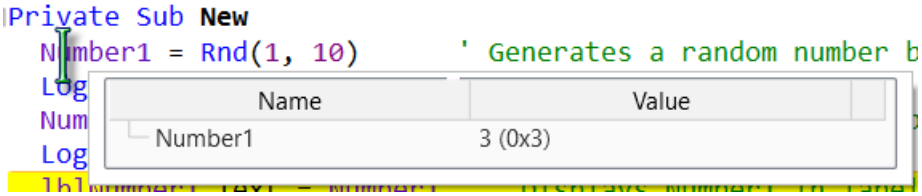


Shows all properties of the object, a Label in the example.



You get the same information when you hover over the object in the code:

lblNumber1

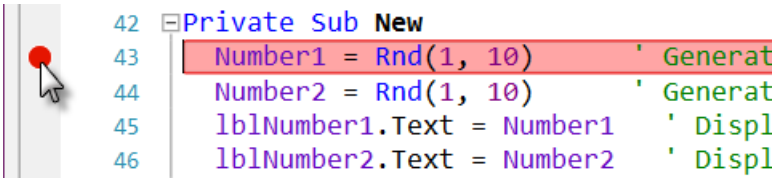


Number1

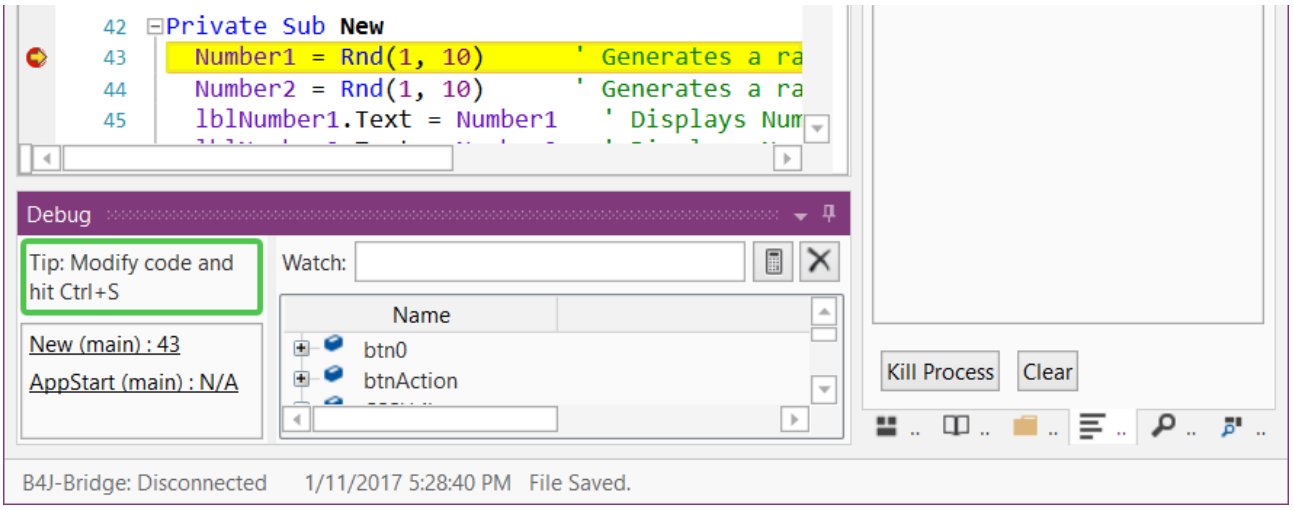
13.3 Breakpoints

One important feature to make debugging easier are breakpoints. You can set breakpoint almost wherever you want in the code.
Breakpoints, in Process_Globals are ignored.

Clicking on a line in the left margin adds a breakpoint. When the program is running it stops at the first encountered breakpoint.



Run the program, the program stops at the breakpoint and the IDE looks like below. The breakpoint line is highlighted in yellow.

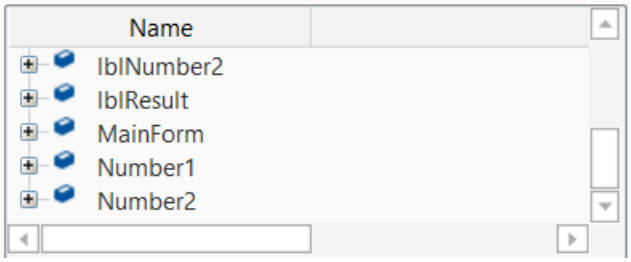


On the bottom of the window you see the debug window.

```
40  End Sub
41
42  Private Sub New
43      Number1 = Rnd(1, 10)
44      Number2 = Rnd(1, 10)
45      lblNumber1.Text = Number1
```

Example with the SecondProgram:

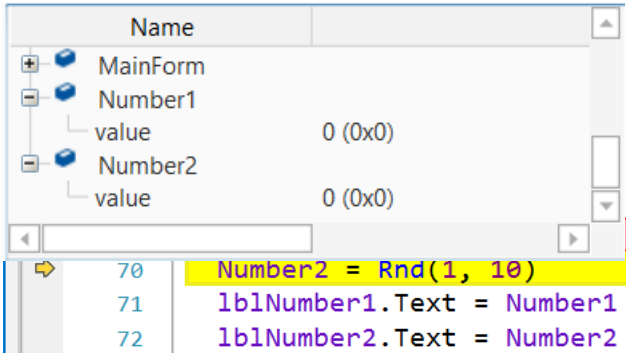
Set a breakpoint in line 43 and run the program.



In the variable window look at Number1 and Number2:

```
42  Private Sub New
43      Number1 = Rnd(1, 10)
44      Number2 = Rnd(1, 10)
```

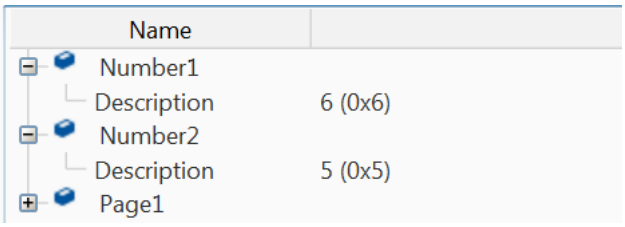
The values are 0 for both.
If you see this ⊞ at the left side of Number1 or Number2 click on it to show the details.



Click on  .
The program jumps to the next line.

```
70      Number2 = Rnd(1, 10)
71      lblNumber1.Text = Number1
72      lblNumber2.Text = Number2
```

Click on ⊞.
You see that the value of Number1 has changed.



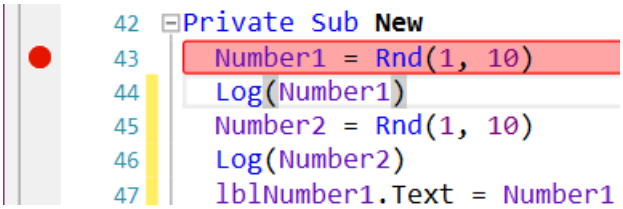
Click on  again.
The program jumps to the next line.

Click on ⊞.
You see that the value of Number2 has changed.

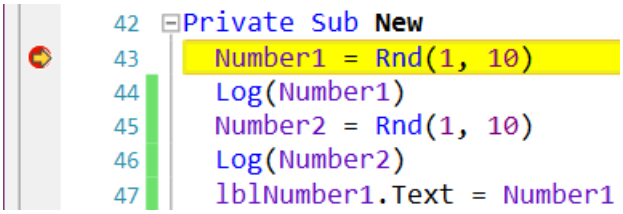
The best way to learn debugging is testing, testing and testing!

13.4 With Logs

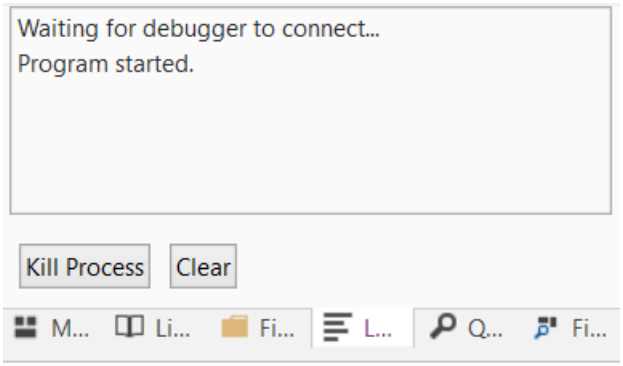
Example with the SecondProgram.



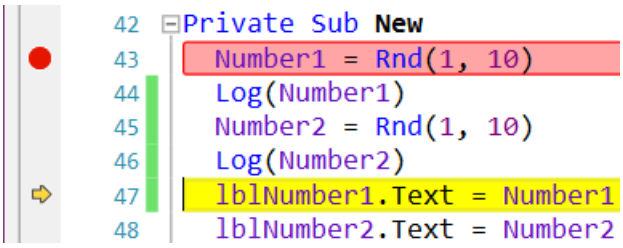
We add the two lines with the Log keyword to display the two numbers in the Log Tab.
We and add a breakpoint in line 69 to watch what happens.



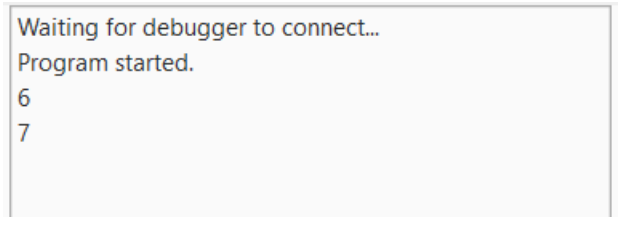
Run the program, it stops at line 69.



In the Log Tab we see at the moment only `Waiting for debugger to connect...` and `Program started` telling that the program has started.

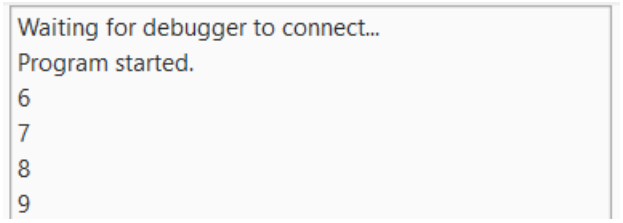


Click four times on  till the program reaches line 47.



In the Log Tab we see the values of the two variables.

Click on  to run to the end.
Nothing new is displayed

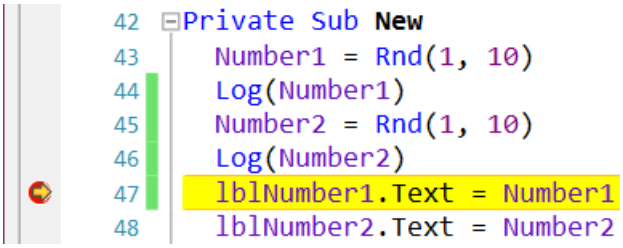


When you are using the program the two new values will be shown every time the program runs the New routine.

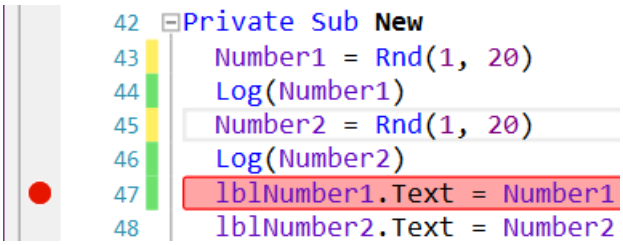
13.5 Modifying code in the Debugger

It is possible to change the code in the Debugger and see the new behavior without restarting the program.

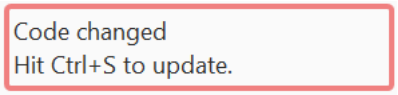
Still with SecondProgram and the two Logs and the breakpoint in line 47.



Run the program till it stops at the breakpoint.



We change the two numbers 10 to 20.



The status button color has changed confirming a code change. To rerun the program click on Ctrl + S.

Using the program we see now that the numbers can be between 1 and 19.

14 Example programs

In this chapter, several example programs are explained with code examples.
All projects are saved in the SourceCode folder shipped with the guide.

14.1 Pong

This is a small vintage pong game. The ball moves and the player must return it with the racket.

The source code is in the SourceCode\Pong folder.

My grandson, 12 years old, dreamed once to become a game programmer, so I proposed him to write a very simple one together to show him a little bit what's behind the scene.

I thought that it could be interesting for beginners.

It shows the use of a Timer for the ball movements, some drawing on a Canvas, the use of Sliders and some other objects. The racket is moved with the mouse.

It needs the CSSUtils library and the CallSubPlus class.

15 Graphics / Drawing

15.1 Overview

To draw graphics, we need to use a Canvas object.

A Canvas is a special Node that can be drawn on.
The Canvas node will not be resized automatically when its parent is resized.

In Java, the drawing is directly done on the Canvas node, you don't need any other node like in B4A or B4i.

A Canvas can be added in the Designer like any other node, therefore no need to initialize it in the code.

Coordinate variables are Double variables and not Float like in B4A.
There is no Rect (rectangle) object in B4J, Rectangles are given with Left, Top, Width, Height.

In the following methods, you will find several common parameters.

- Image1 as Image — an image, like a bitmap
- x, y, x1, y1, x2, y2 As Double — are coordinates, Double variables.
- fx.Color as Color — are color variables. Int variables
- Filled As Boolean — flag if the surface is filled (True) or not (False)

The most common drawing functions are:

- **DrawImage** (Image1 As Image, x As Double, y As Double, Width As Double, Height As Double).
  Draws the given image.
  x and y = coordinates of the top left corner.
  Width = The width of the destination rectangle.
  Height = The height of the destination rectangle.
  Do draw with the same size both rectangles must have same width and same height.

- **DrawImage2** (Image1 As Image, SourceX As Double, SourceY As Double, SourceWidth As Double, SourceHeight As Double, DestX As Double, DestY As Double, DestWidth As Double, DestHeight As Double).
  Draws the given image or only a part of it.
  SourceX and SourceY = coordinates of the top left corner of the source rectangle.
  SourceWidth = The width of the source rectangle.
  SourceHeight = The height of the source rectangle.
  DestX and DestY = coordinates of the top left corner of the destination rectangle.
  DestWidth = The width of the destination rectangle.
  DestHeight = The height of the destination rectangle.
  To draw with the same size both rectangles must have same width and same height.

- **DrawImageRotated** (Image 1 As Image, x As Double, y As Double, Width As Double, Height As Double, Degree As Double)
  Same function as DrawImage, but with a rotation of the given angle in Degrees around the center of the bitmap.

- **DrawCircle** (x As Double, y As Double, Radius As Double, Paint As Paint, Filled As Boolean, StrokeWidth As Double)
  Draws a circle.
  x and y are the center coordinates of the circle and Radius the circles radius.

- **DrawLine** (x1 As Double, y1 As Double, x2 As Double, y2 As Double, Paint As Paint, StrokeWidth As Double)
  Draws a straight line between two points.

- **DrawRect** (x As Double, y As Double, Width As Double, Height As Double, Paint As Paint, Filled As Boolean, StrokeWidth As Double)
  Draws a rectangle with given size, color, filled or not and line width.

- **DrawRectRotated** (x As Double, y As Double, Width As Double, Height As Double, Paint As Paint, Filled As Boolean, StrokeWidth As Double, Degree As Double)
  Draws a rotated rectangle with given size, color, filled or not, line widths and rotation angle in degrees.

- **DrawText** (Text As String, x As Double, y As Double, Font As Font, Paint As Paint, Alignment As TextAlignment Enum).
  Draws the given text.
  Font = Font object
  Alignment can be either : LEFT, CENTER or RIGHT

- **DrawText2** (Text As String, x As Double, y As Double, Font As Font, Paint As Paint, Alignment As TextAlignment Enum, MaxWidth As Double).
  Draws the given text.
  Font = Font object
  Alignment can be either : LEFT, CENTER or RIGHT
  Similar to DrawText. MaxWidth defines the text bounds. The text will be wrapped if it is longer.

- **DrawTextRotated** (Text As String, x As Double, y As Double, Font As Font, Paint As Paint, Alignment As TextAlignment Enum, Degree As Double)
  Similar to DrawText. Draws rotated text.
  Font = Font object

15.2 Coordinates

Coordinate values are different in B4J from those in B4A and B4i.

The only coordinate values are Pixels.

15.3 Transparency

There is a big difference between B4J and B4A with transparency.

In B4A, drawing with any Draw method and Transparent color sets the pixels to transparent, this means that the underlying parts are shown.
In Java (B4J), drawing with any Draw method and Transparent color does nothing!
It draws ‘transparent’ pixels onto the canvas which means, do nothing.

The only way to get parts of the canvas transparent is the **ClearRect** method.

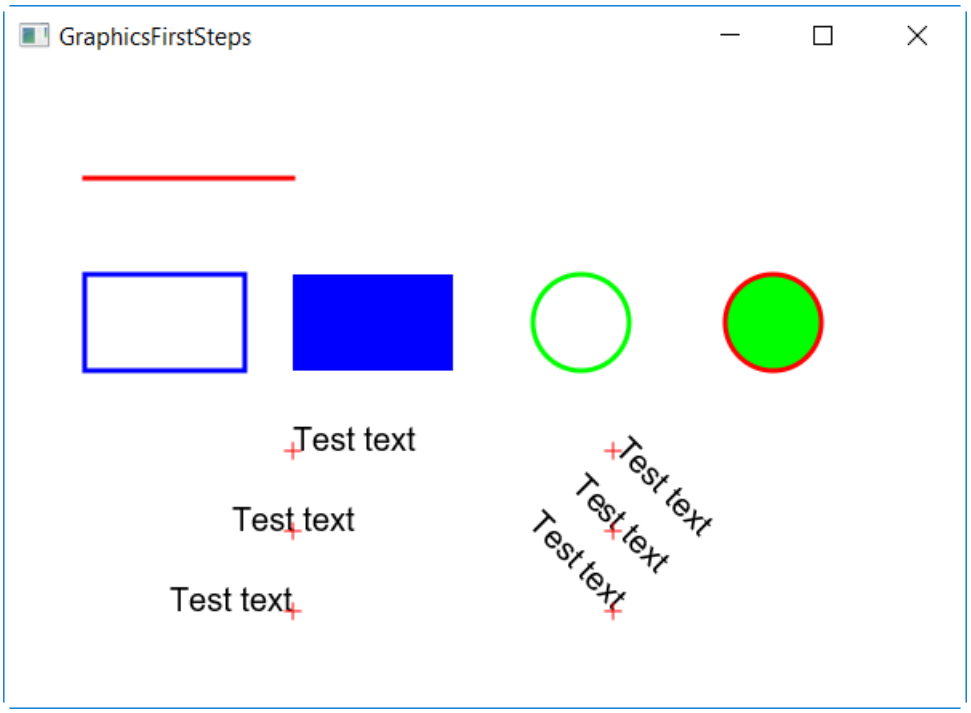
15.4 Example programs

15.4.1 First steps Example program

The project is in: SourceCode\Graphics\GraphicsFirstSteps.b4j

It is not possible to draw directly onto a Form but you can draw on different Canvases at the same time.

In the example program we'll use several drawing functions and draw onto a Canvas `cvsGraph` defined in the 'Main' layout file.



15.4.1.1 Start - declaration

First, we must declare the Canvas in **Process_Globals**:

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form

   Private cvsGraph As Canvas
End Sub
```

Then we load the layout file, show the form and call the Drawing routine.

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

   Drawing
End Sub
```

15.4.1.2 Draw a line

Then in the Drawing routine we draw a horizontal line onto `cvsGraph`:

The function is:
DrawLine (x1 As Double, y1 As Double, x2 As Double, y2 As Double, Paint As Paint, StrokeWidth As Double)
Where:

- x1, y1 are the coordinates of the start point in pixels
- x2, y2 are the coordinates of the end point in pixels
- Paint is the line color
- StrokeWidth the line thickness in pixels

And the code:

```
' draw a horizontal line
cvsGraph.DrawLine(20, 20, 150, 20, fx.Colors.Red, 3)
```

15.4.1.3 Draw a rectangle

Then we draw an empty rectangle: 

The function is:
DrawRect (x As Double, y As Double, Width As Double, Height As Double, Paint As Paint, Filled As Boolean, StrokeWidth As Double)
Where:

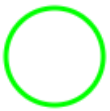
- y and y = the coordinates of the top left corner.
- Width and Height = the dimensions of the rectangle.
- Paint is the border or rectangle color
- Filled: False = only the border is drawn True = the rectangle is filled
- StrokeWidth is the line thickness of the border, not relevant when Filled = True

```
' draw an empty rectangle
cvsGraph.DrawRect(20, 80, 100, 60, fx.Colors.Blue, False, 3)
```

Then we draw a filled rectangle: 

```
' draw a filled rectangle
cvsGraph.DrawRect(150, 80, 100, 60, fx.Colors.Blue, True, 3)
```

15.4.1.4 Draw a circle

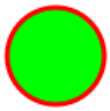
Then we draw an empty circle: 

The function is:
DrawCircle (x As Double, y As Double, Radius As Double, Paint As Paint, Filled As Boolean, StrokeWidth As Double)
Where:

- x, y are the coordinates of the center in pixels.
- Radius is the radius in pixels.
- Paint is the border or circle color
- Filled: False = only the border is drawn True = the circle is filled
- StrokeWidth is the line thickness of the border, not relevant when Filled = True

And the code:

```
' draw an empty circle
cvsGraph.DrawCircle(330, 110, 30, fx.Colors.Green, False, 3)
```

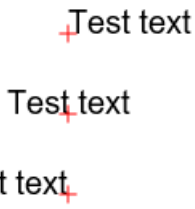
Then we draw a filled circle with a border with a different color. 

There is no direct function to draw a filled circle with a border with a different colors.
So we first draw the filled circle and then the circle border in two steps.
Instead of using fixed values like 220 we can also use variables like in the code below.
When a same value is used several times it's better to use variables because if you need to change the value you change only once the value of the variable all the rest is changed automatically by the variable.

```
' draw a filled circle with a boarder
Private centerX, centerY, radius As Double
centerX = 450
centerY = 110
radius = 30
cvsGraph.DrawCircle(centerX, centerY, radius, fx.Colors.Green, True, 3)
cvsGraph.DrawCircle(centerX, centerY, radius, fx.Colors.Red, False, 3)
```

15.4.1.5 Draw a text

Then we draw texts with the three alignments.



The function is:
DrawText (Text As String, x As Double, y As Double, Font As Font, Paint As Paint, Alignment As TextAlignment Enum)
Where:

- Text is the text to draw
- x, y are the coordinates of the reference point in pixels.
  The reference point is on the texts baseline.
- Font1 is the Font object
- Paint is the text color
- Alignemt is the alignment of the text according to the reference point.
  Possible values: "LEFT", "CENTER", "RIGHT". Note: must be upper case!

We use variables for the coordinates and draw a cross at the reference point to show how the text is aligned.
`refX` = x coordinate of the reference point.
`refY` = y coordinate of the reference point.
`hl` = half length of the cross lines.

And the code:

```
Private refX, refY, hl As Double
refX = 150
refY = 190
hl = 5
' draw texts with the three alignments
Private Font1 As Font
Font1 = fx.CreateFont("Arial", 20, False, False)
cvsGraph.DrawText("Test text", refX, refY, Font1, fx.Colors.Black, "LEFT")
DrawCross(refX, refY, hl) ' draw a cross on the reference point

refY = 240
cvsGraph.DrawText("Test text", refX, refY, Font1, fx.Colors.Black, "CENTER")
DrawCross(refX, refY, hl)

refY = 290
cvsGraph.DrawText("Test text", refX, refY, Font1, fx.Colors.Black, "RIGHT")
DrawCross(refX, refY, hl)
```

Routine to draw the cross :

```
'Draw a cross on the reference point
Private Sub DrawCross(x As Double, y As Double, l As Double)
   cvsGraph.DrawLine(x - l, y, x + l, y, fx.Colors.Red, 1)
   cvsGraph.DrawLine(x, y - l, x, y +l, fx.Colors.Red, 1)
End Sub
```

Then we draw rotated texts with the three alignments.

The function is DrawTextRotated, it's the same as DrawText but with an additional parameter Degrees, the rotation angle.

```
refX = 350
refY = 190
' draw rotated texts  with the three alignments
cvsGraph.DrawTextRotated("Test text", refX, refY, Font1, fx.Colors.Black, "LEFT", 45)
DrawCross(refX, refY, hl) ' draw a cross on the reference point

refY = 240
cvsGraph.DrawTextRotated("Test text", refX, refY, Font1, fx.Colors.Black, "CENTER", 45)
DrawCross(refX, refY, hl)

refY = 290
cvsGraph.DrawTextRotated("Test text", refX, refY, Font1, fx.Colors.Black, "RIGHT", 45)
DrawCross(refX, refY, hl)
```

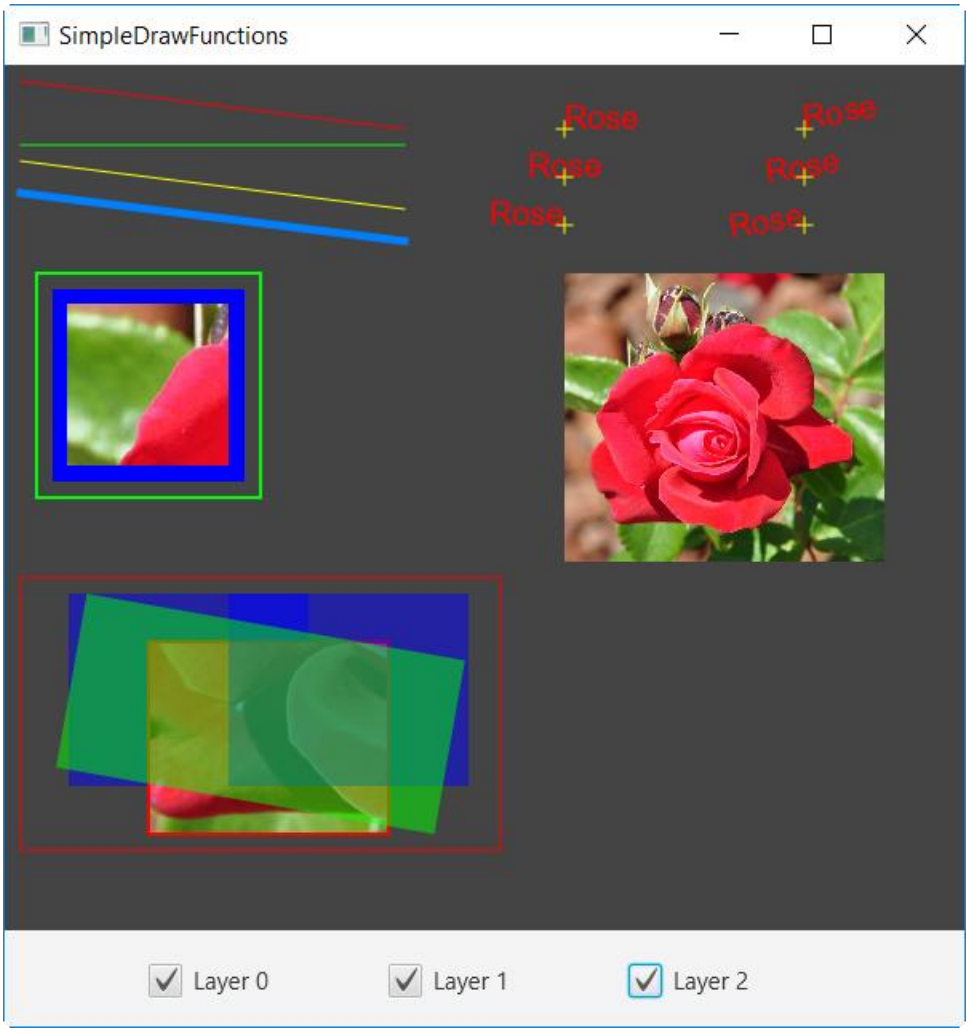
15.4.2 Simple draw functions Example program

The project is in: SourceCode\Graphics\SimpleDrawFunctions\SimpleDrawFunctions.b4j

In the second drawing program, SimpleDrawFunctions, we use the other common drawing functions.

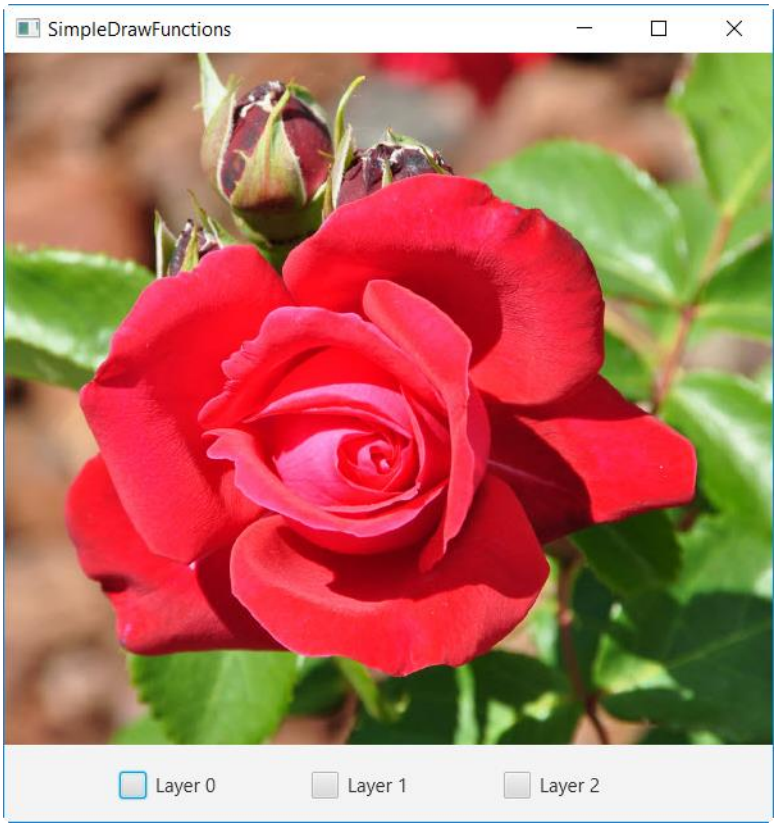
The program has no other purpose than show what can be done with drawings.

The program has four Panels which we use as layers, one for the background and three layers with three Switches allowing to show or hide the different layers.
The background layer has an image, Layer(0) has a grey background and the two other layers have a transparent background.



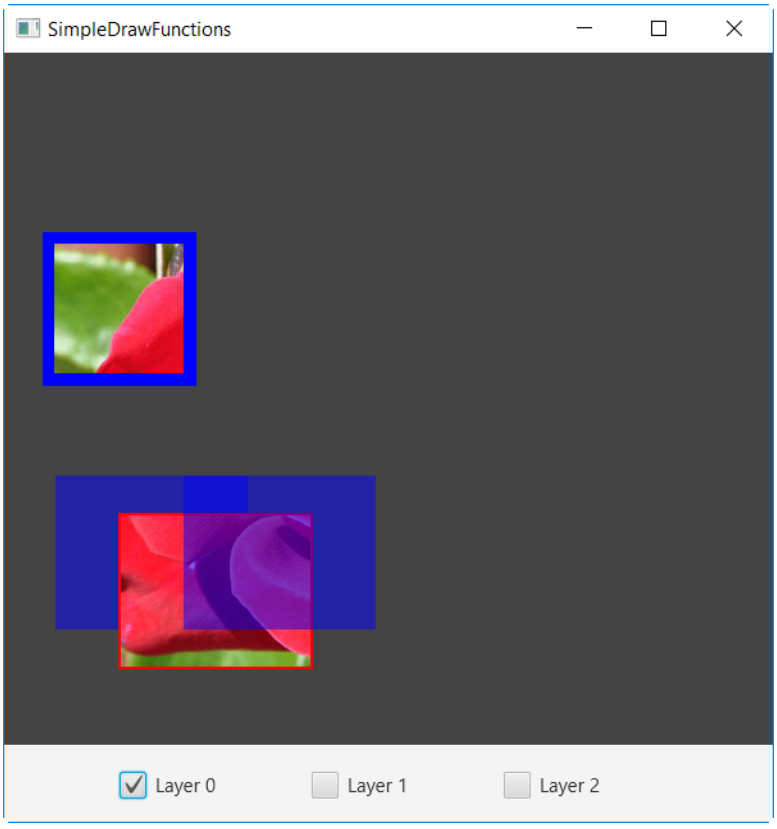
You can play with the switches to observe the different combinations of visible and hidden layers.

The layout is defined in the Designer.



In this screenshot we solely see the background image of the background layer.

We use Checkboxes to either show or hide the different layers.

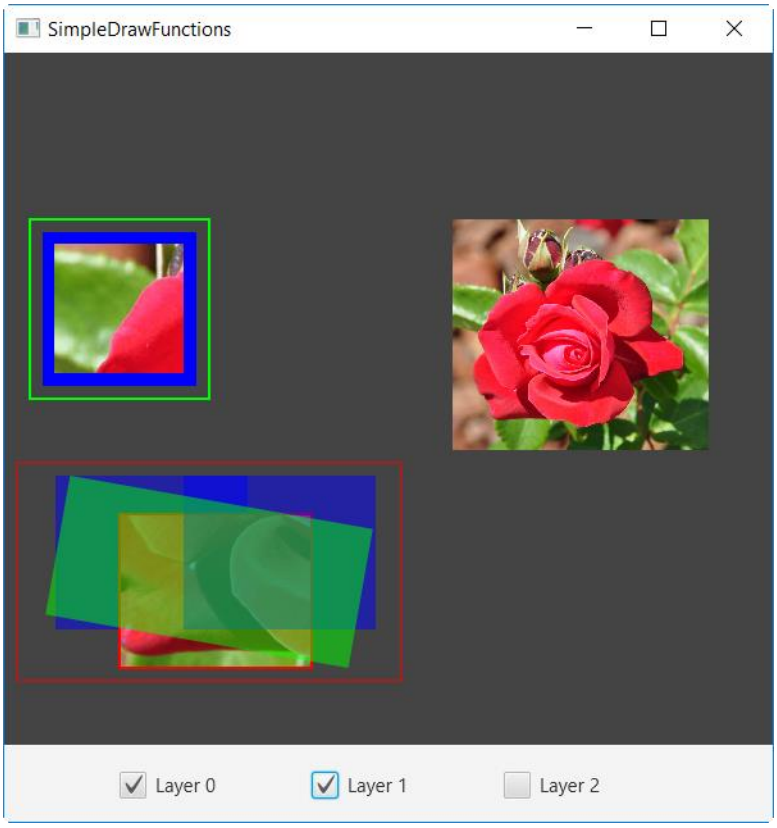


Here we show layer(0).

The panel has a dark gray background with:

- a light blue square.
- a transparent square, the background image appears inside this square.
- two blue, semi-transparent rectangles, the left one is drawn before the transparent rectangle and the second one is drawn after the transparent rectangle.
- a transparent rectangle, the background image appears inside this rectangle.

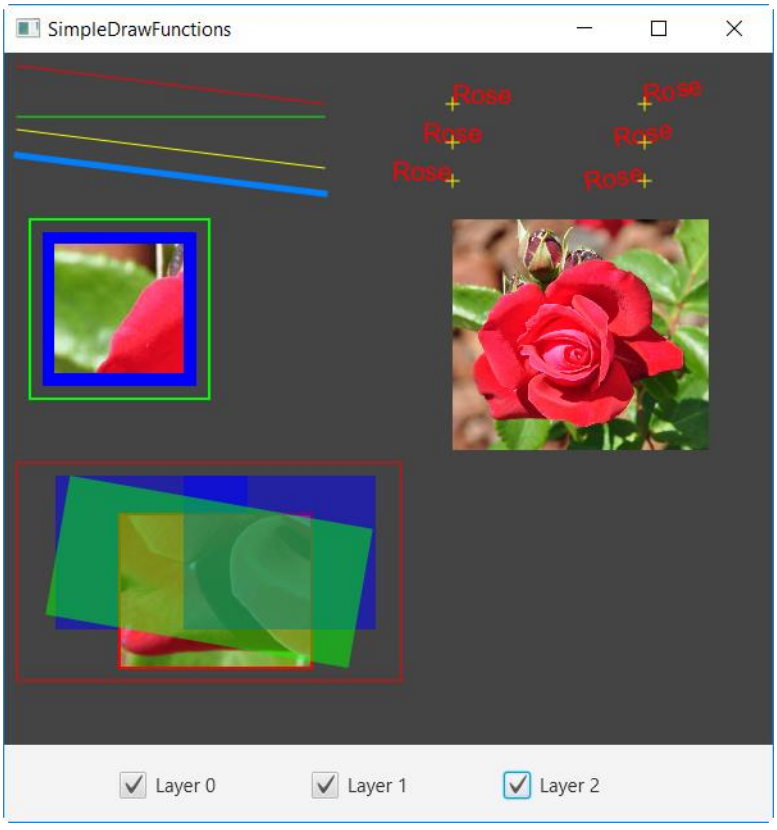
Touching the screen and moving the finger moves the blue and transparent circles on layer(0).



Here we show layer(1) on top of layer(0).

The panel has a transparent background with:

- a green square.
- a small copy of the background image.

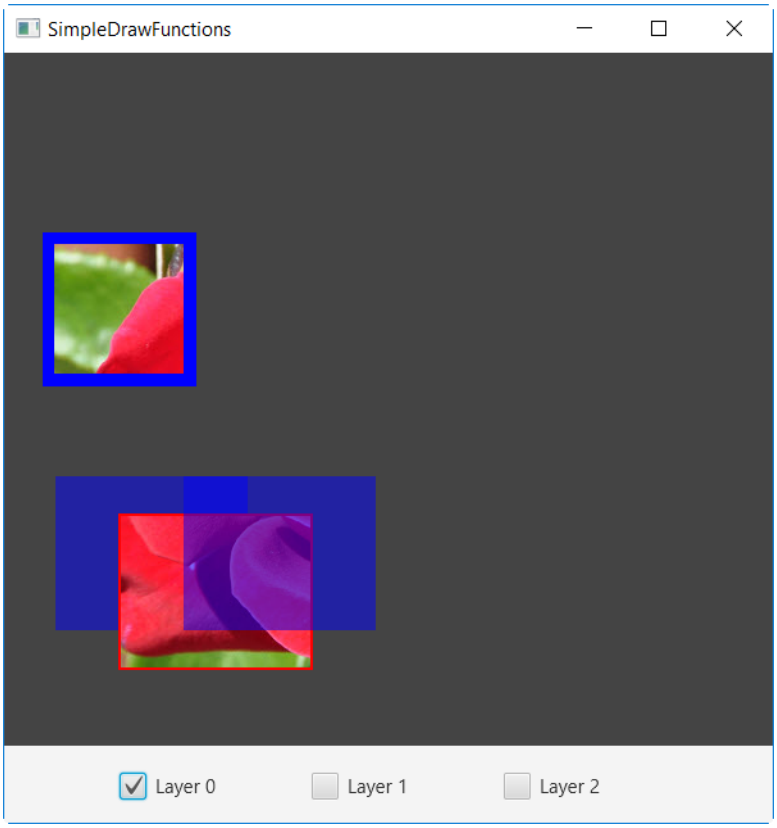


Here we show layer(2) on top of layer(0) and layer(1).

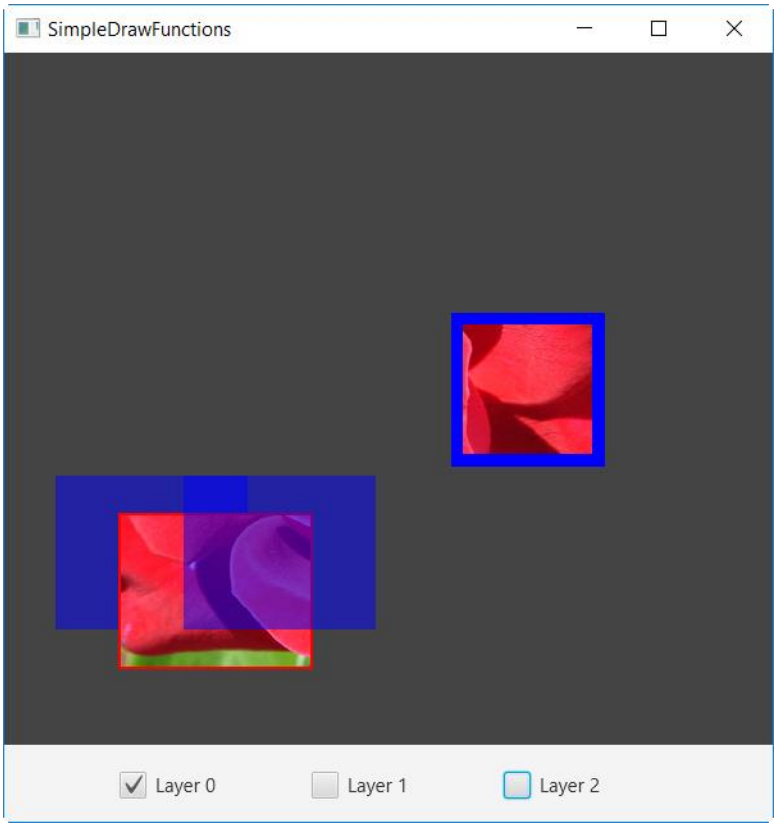
The panel has a transparent background with:

- 4 lines on top.
- 3 horizontal texts with the three different alignments.
- 3 rotated texts with the three different alignments.
- a cross for each text showing the position of the text reference point.

You can play with the buttons to show the different combinations of visible and hidden layers.



Touching the screen with the finger and moving it, moves the blue and transparent squares.



On each move, the background image of the activity appears.

Analysis of the code:

In the Sub Process_Globals routine we have:

- 3 application variables.
- 4 Panels, one background and 3 layers.
- 4 Canvases, one for each panel.
- 2 Rects, one for the background and the other is used to draw rectangles.
- 1 Bitmap, holding the background image
- different variables used for the drawings.

Note that we use arrays of nodes for the three layer panels and the canvases.

```
Sub Process_Globals
   Private fx As JFX
   Private MainForm As Form

   Private cvsBackground, cvsLayer0, cvsLayer1, cvsLayer2, cvsLayer(3) As Canvas
   Private bmpBackground As Image

   Private xc, yc, x1, y1, x2, y2, w1, w2 As Double
End Sub
```

In Sub AppStart we:

- Set Form1 to the variable MainForm.
- Load the layout.
- Show MainForm.
- Set the layers array.
- Set `setMouseTransparent` to `True` to transfer the events to the underlying node. This property is not exposed to B4J so we use a JavaObject.
- Initialize the background image `rose2.jpg` of the form.
- Call the Drawing routine

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

   'define an Array of Canvases
   cvsLayer = Array As Canvas(cvsLayer0, cvsLayer1, cvsLayer2)

   'transfer the events to the underlying node
   Private jo = cvsLayer2 As JavaObject
   jo.RunMethod("setMouseTransparent", Array As Object(True))
   Private jo = cvsLayer1 As JavaObject
   jo.RunMethod("setMouseTransparent", Array As Object(True))

   'initialize the background image
   bmpBackground.Initialize(File.DirAssets, "rose.jpg")

   Drawing
End Sub
```

In the Sub Drawing routine we:

- Initialize the Font for the text.
- Draw the background image.
- Draw the layout(0) background dark gray.
- The background of layout(1) and layout(2) are set to transparent in the Designer.

```
Sub Drawing
   Private Font1 As Font
   Font1 = fx.CreateFont("Arial", 20, False, False)

   'draw the background image
   cvsBackground.DrawImage(bmpBackground, 0, 0, cvsBackground.Width,
cvsBackground.Height)

   cvsLayer(0).DrawRect(0, 0, cvsLayer(0).Width, cvsLayer(0).Height, fx.Colors.DarkGray,
True, 1)
```

- Draw the two squares on layer(0) and layer(1)

```
   ' draw the three squares on layer(0) and layer(1)
   xc = 90
   yc = 200
   w1 = 140
   cvsLayer(1).DrawRect(xc - w1 / 2, yc - w1 / 2, w1, w1, fx.Colors.Green, False, 2)
   w1 = 120
   cvsLayer(0).DrawRect(xc - w1 / 2, yc - w1 / 2, w1, w1, fx.Colors.Blue, True, 1)
   w2 = 100
   cvsLayer(0).ClearRect(xc - w2 / 2, yc - w2 / 2, w2, w2)
```

- Draw the rectangles on layer(0) and layer(1)

```
   ' draw the rectangles on layer(0) and layer(1)
   cvsLayer(1).DrawRect(10, 320, 300, 170, fx.Colors.Red, False, 1)
   cvsLayer(0).DrawRect(40, 330, 150, 120, fx.Colors.ARGB(128, 0, 0, 255), True, 1)
   cvsLayer(1).DrawRectRotated(40, 350, 240, 110, fx.Colors.ARGB(128, 0, 255, 0), True,
1, 10)
   cvsLayer(0).ClearRect(90, 360, 150, 120)
   cvsLayer(0).DrawRect(90, 360, 150, 120, fx.Colors.Red, False, 2)
   cvsLayer(0).DrawRect(140, 330, 150, 120, fx.Colors.ARGB(128, 0, 0, 255), True, 1)

   'draw the background image smaller
   cvsLayer(1).DrawImage(bmpBackground, 350, 130, 200, 180)
```

- Draw four lines onto layer(2).

```
' draw the four lines on layer(2)
x1 = 10
y1 = 10
x2 = 250
y2 = 40
cvsLayer(2).DrawLine(x1, y1, x2, y2, fx.Colors.Red, 1)
y1 = 50
y2 = 50
cvsLayer(2).DrawLine(x1, y1, x2, y2, fx.Colors.Green, 1)
y1 = 60
y2 = 90
cvsLayer(2).DrawLine(x1, y1, x2, y2, fx.Colors.Yellow, 1)
y1 = 80
y2 = 110
cvsLayer(2).DrawLine(x1, y1, x2, y2, fx.Colors.RGB(0, 128, 255), 5)
```

- Draw the three horizontal texts.
  DrawCross draws the reference point of the text.

```
' draw the three horizontal texts
x1 = 350
y1 = 40
cvsLayer(2).DrawText("Rose", x1, y1, Font1, fx.Colors.Red, "LEFT")
DrawCross(x1, y1, fx.Colors.Yellow)
y1 = 70
cvsLayer(2).DrawText("Rose", x1, y1, Font1, fx.Colors.Red, "CENTER")
DrawCross(x1, y1, fx.Colors.Yellow)
y1 = 100
cvsLayer(2).DrawText("Rose", x1, y1, Font1, fx.Colors.Red, "RIGHT")
DrawCross(x1, y1, fx.Colors.Yellow)
```

- Draw the three rotated texts.
  Similar to the code above, but rotated with the rotation angle as the last parameter-.

```
cvsLayer(2).DrawTextRotated("Rose", x1, y1, Font1, fx.Colors.Red, "LEFT", -10)
```

Looking closer on the displayed texts we see yellow crosses which are the reference point for each text.

LEFT alignment.
CENTER alignment.
RIGHT alignment.

These are the x1 and y1 coordinates used to display the texts.

```
cvsLayer(2).DrawText("Rose", x1, y1, Font1, fx.Colors.Red, "RIGHT")
DrawCross(x1, y1, fx.Colors.Yellow)
```

In the Sub ckbLayer_CheckedChange routine we:

- Dim ckb as a local CheckBox to get the node that raised the event.
- Set ckb to Sender, which is the node that raised the event.
- Get the index of the CheckBox from the Tag property, the Tag property is set in the Designer.
- Change the Visible property from True to False or from False to True.

```
' CheckBox ValueChanged event routine, all three CheckBoxes call this routine
Private Sub ckbLayer_CheckedChange(Checked  As Boolean)
   Dim ckb As CheckBox
   Dim index As Int

   ckb = Sender
   index = ckb.Tag
   cvsLayer(index).Visible = Checked
End Sub
```

In the Sub cvsLayer0_MouseDragged routine we:

- Draw a dark gray square to erase the previous blue and transparent square.
- Set xc and yc to the new coordinates of the square centers.
- Draw a blue square on layer(1).
- Clear the transparent square by setting its background to transparent with `ClearRect`.

```
Private Sub cvsLayer0_MouseDragged (EventData As MouseEvent)
   cvsLayer(0).DrawRect(xc - w1 / 2 - 1, yc - w1 / 2 - 1, w1 + 2, w1 + 2,
fx.Colors.DarkGray, True, 1)
   xc = EventData.X
   yc = EventData.Y
   cvsLayer(0).DrawRect(xc - w1 / 2, yc - w1 / 2, w1, w1, fx.Colors.Blue, True, 1)
   cvsLayer(0).ClearRect(xc - w2 / 2, yc - w2 / 2, w2, w2)
End Sub
```

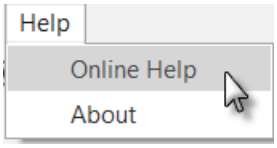
In the DrawCross routine we draw two lines.

```
'Draws a cross at the given coordinates with the given color
'x any y = coordinates in pixels
'Color = color of the two lines
'Code example: <code>
'DrawCross(20, 50, fx.Colors.Red)</code>
Private Sub DrawCross(x As Int, y As Int, Paint As Paint)
   Private d = 5 As Int

   cvsLayer(2).DrawLine(x - d, y, x + d, y, Paint, 1)
   cvsLayer(2).DrawLine(x, y - d, x, y + d, Paint, 1)
End Sub
```

16 Help Tools

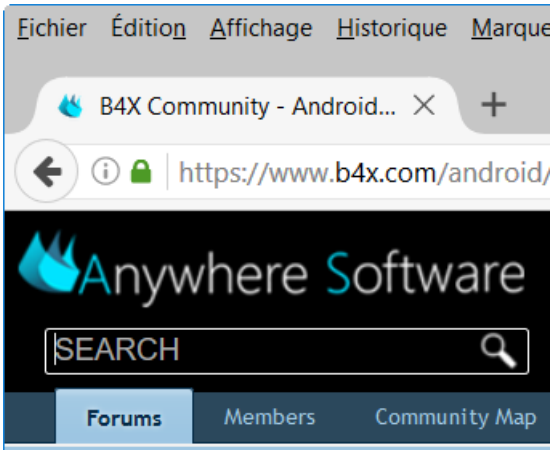
To find answers to questions about B4J the following tools are very useful.

16.1 Online Help link in the IDE



In the IDE Help menu click on Online Help.

16.2 Search function in the forum

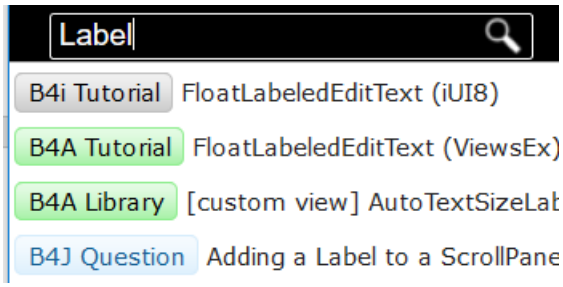


In the upper left or upper right corner you find the search box for the forum.

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

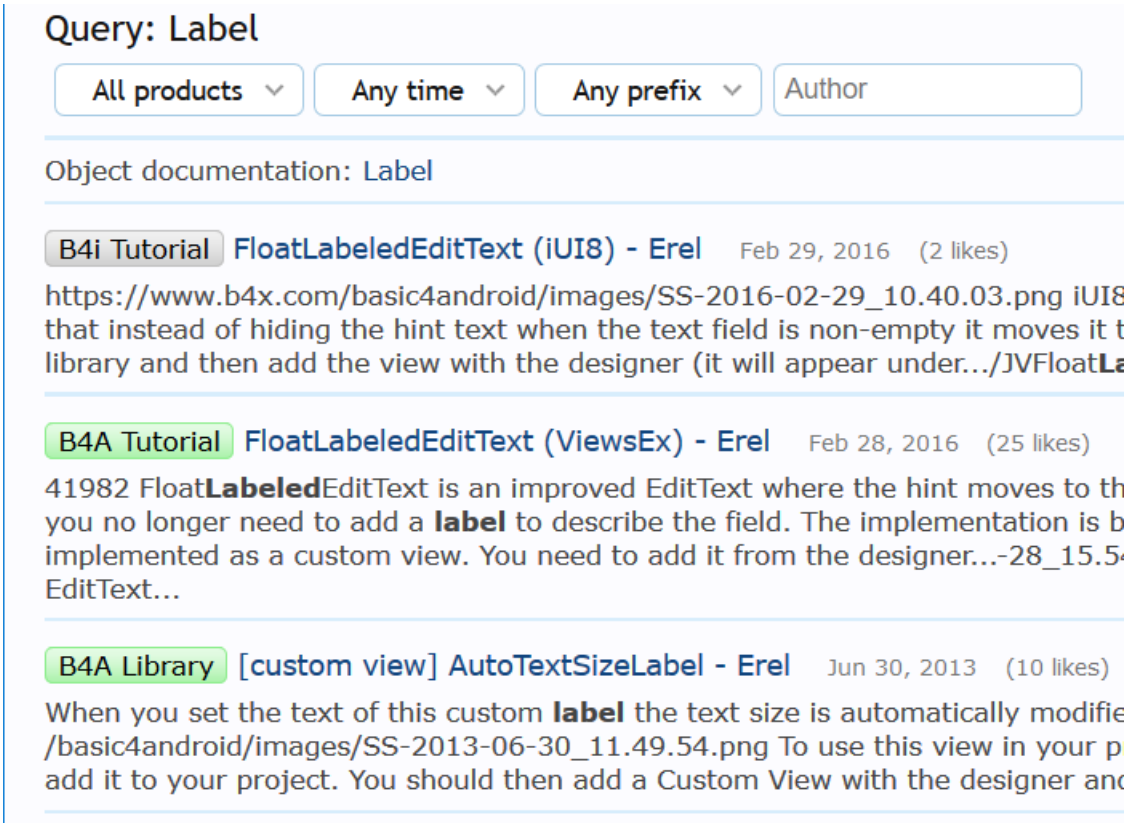
The function shows you the posts that match your request.

Example: Enter the keyword Label:



While you type, you get suggestions just below the search field.

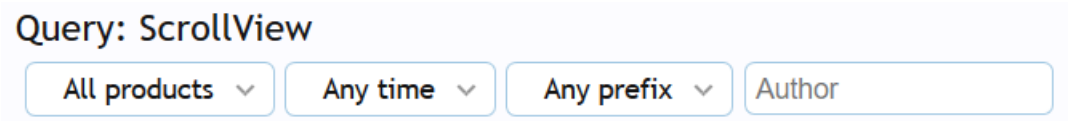
And the result:



Click on the title to show the selected post.

We see that there is no result for B4J on top of the list.

On top of the screen you see comboboxes allowing to filter the result.



You can filter the search with different options.

Example: Click on B4J in the first combobox:

The result may look like this:

We would have got the same result with a direct product filter.

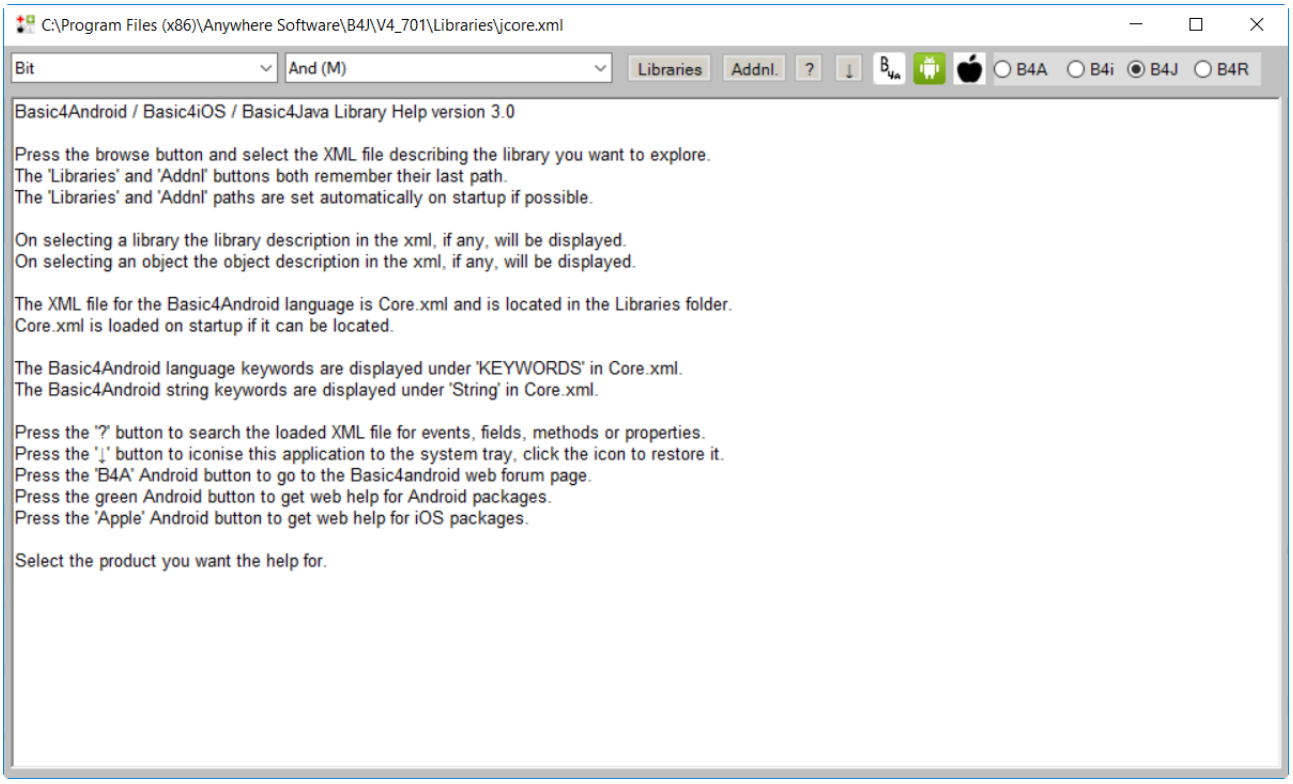
Instead of enter **Label** in the search field we could entered **B4J Label**.

Adding the product name, B4J, B4A, B4i or B4r, before the search value filters the results directly.

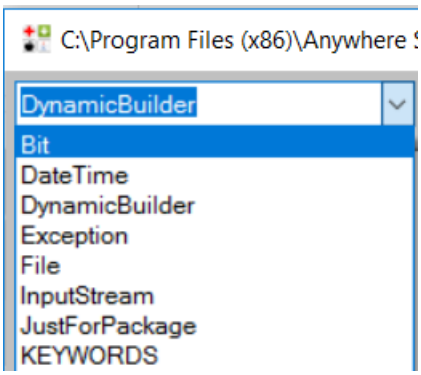
16.3 B4x Help Viewer

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

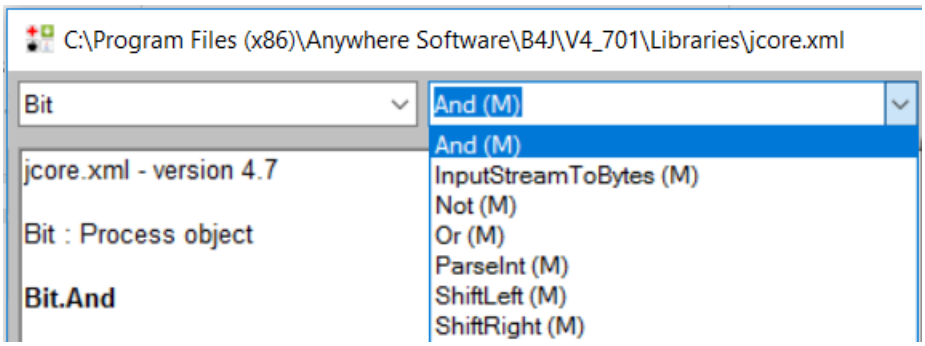
The program can be downloaded from the forum.



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

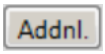
- methods(M)
- events(E)
- properties(P)
- fields(F) constants

for the selected object.

Select an object.

Select a property.

Select a standard library.



Select an additional library.

Search for a given text.

Closes B4xHelp

Link to the B4x forum.

Link to Android documentation.

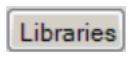
Link to iOS documentation.

B4A help files.

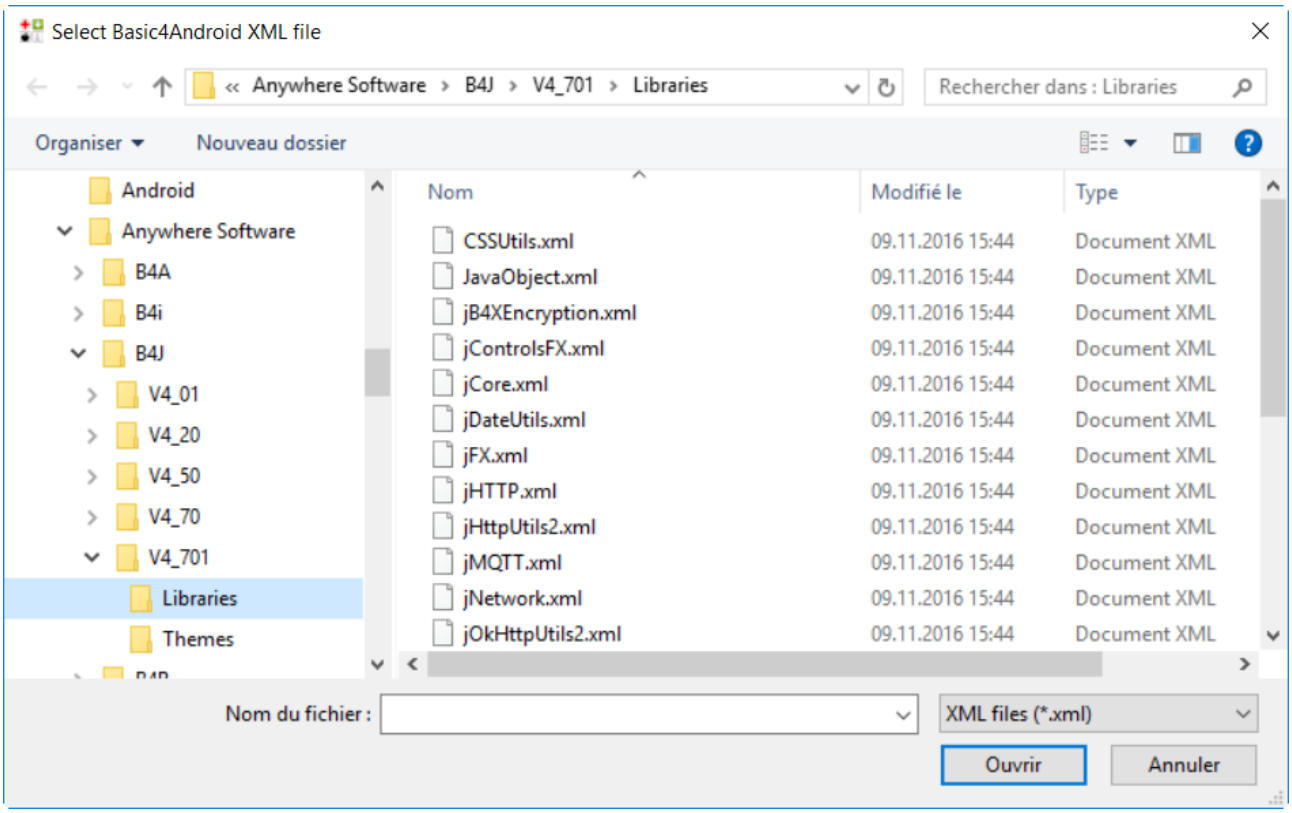
B4i help files.

B4J help files.

B4R help files.



Standard libraries



Select the library to display and click on 

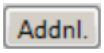
(Open).

Here 

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

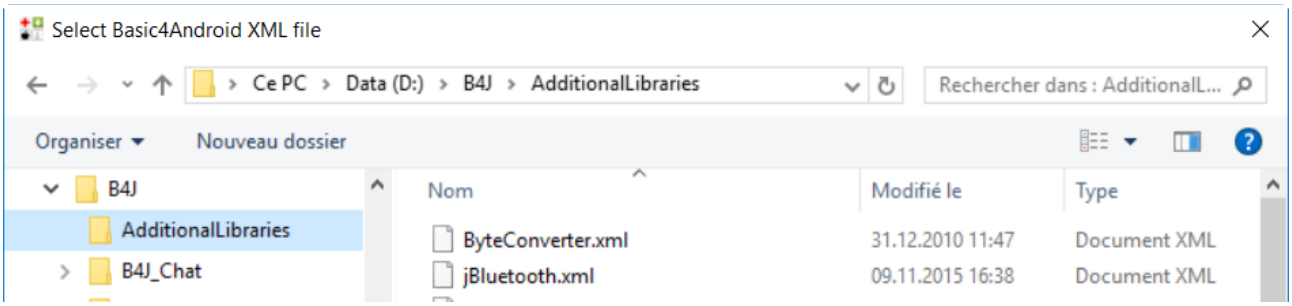
Once selected the directory is saved for the next start of the program.

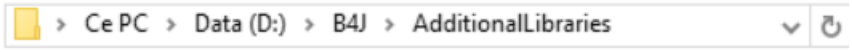
Note: All the nodes are in the *jFX.xml* library and not in the *jCore.xml* library.



Additional libraries.

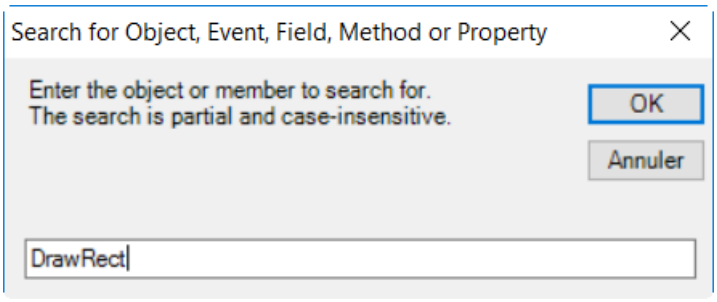
The same also for the additional libraries.



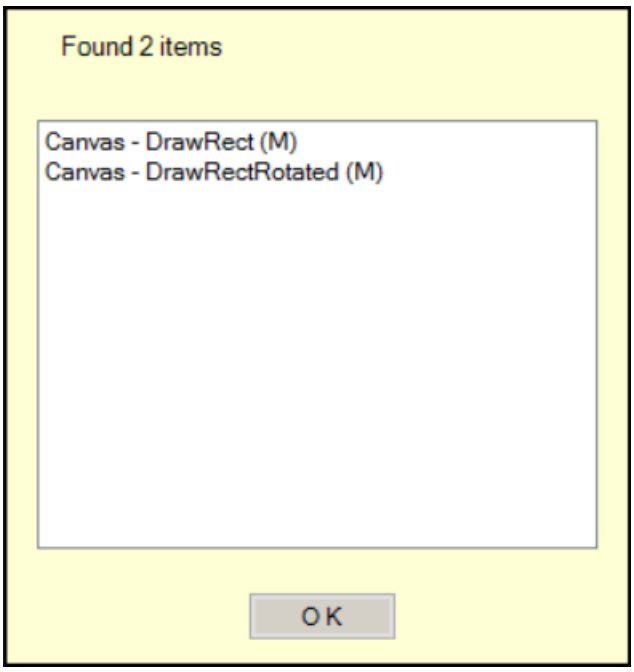
Here 

you can select the directory for the additional libraries.

Search engine for the selected library.

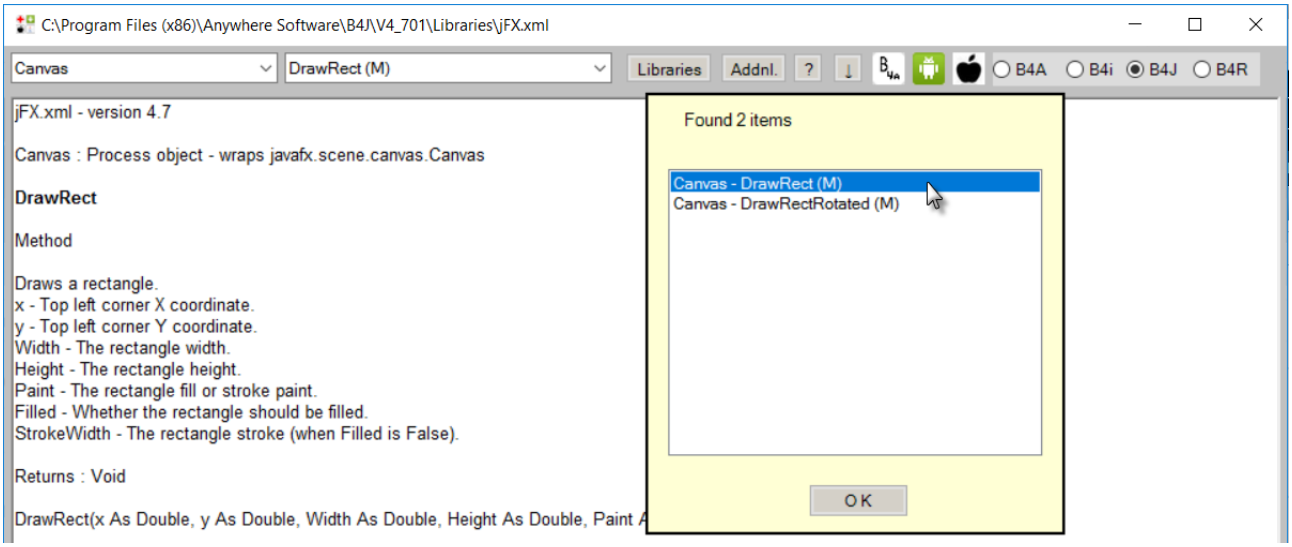


Example:
Selected library: jFX
Enter *DrawRect*

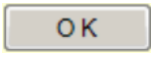


And the result.

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.

16.4 Asking a question in the forum

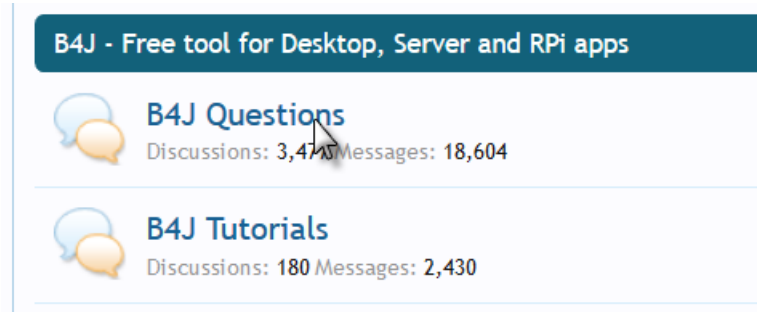
If you cannot find the answer with the previous tools you can ask the question in the forum.

Guidelines to ask a question:

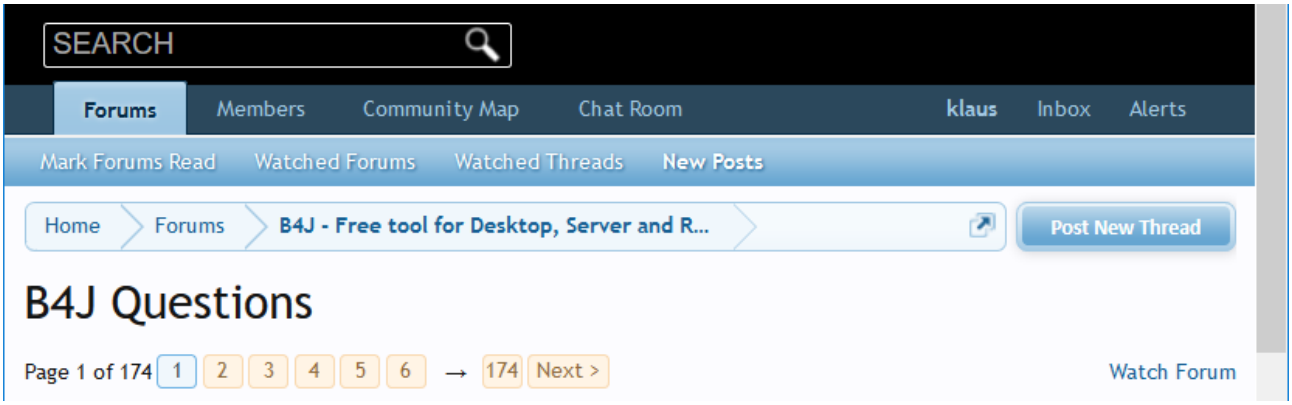
- Open a new thread for each problem with a meaningful title.
  This will make researches easier for other users.
- Explain in detail your problem.
  - What you want to do.
  - What you have done and how.
  - What is not working as expected?
- Post the error message in the Log if you get one.
- Post the relevant code or better a small project showing the problem.
- Click on Like in the answer, or post a confirmation when the problem is solved.

16.5 Creating a new thread

Example:

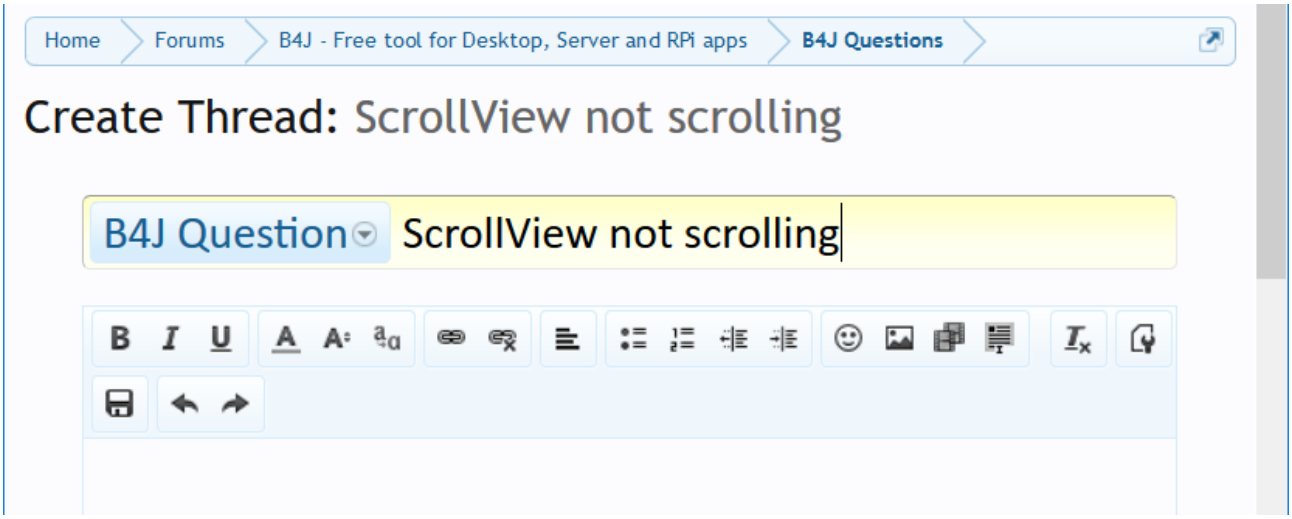


In the B4x Community page select the correct forum, depending on the product.



Then click on 

.



Enter a meaningful title.

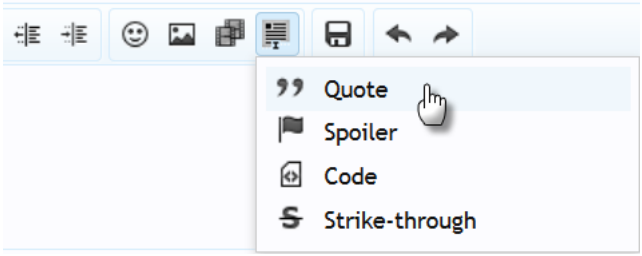
16.5.1 Editing a new thread or post

The editor:

Enter your text in the editor.

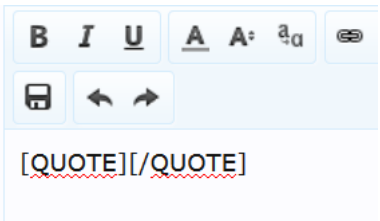
There are some useful features to edit the text:

16.5.2 QUOTE tags



Click on the Insert button and

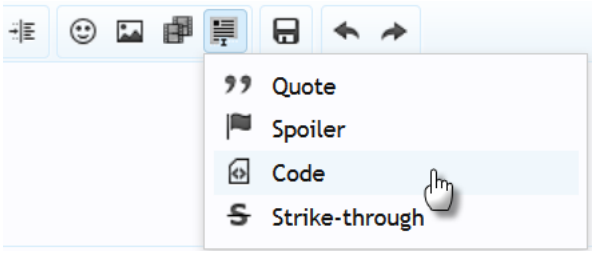
select Quote.



You will see [QUOTE][/ QUOTE] in the editor

Enter your text between][, for example [QUOTE]Test text for quote.[/QUOTE]

16.5.3 CODE tags

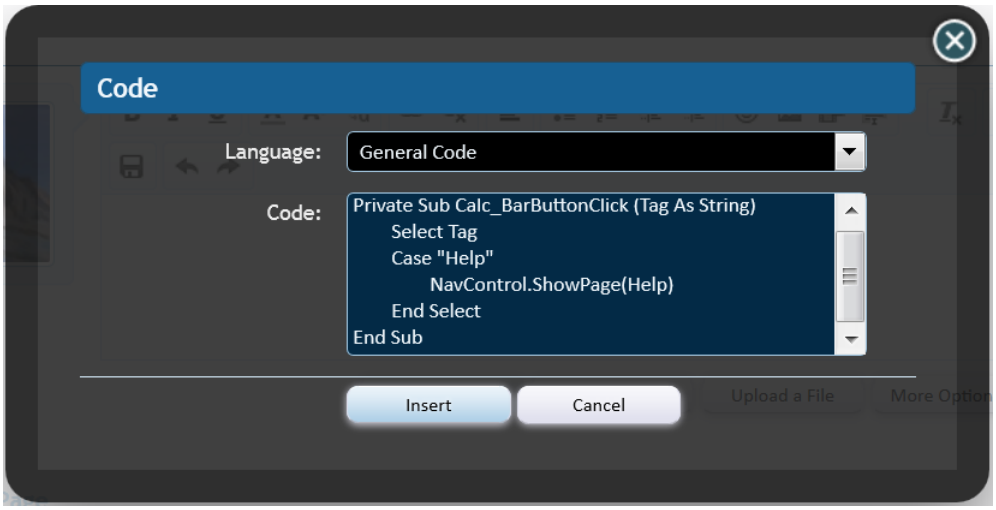


Click on the Insert button and

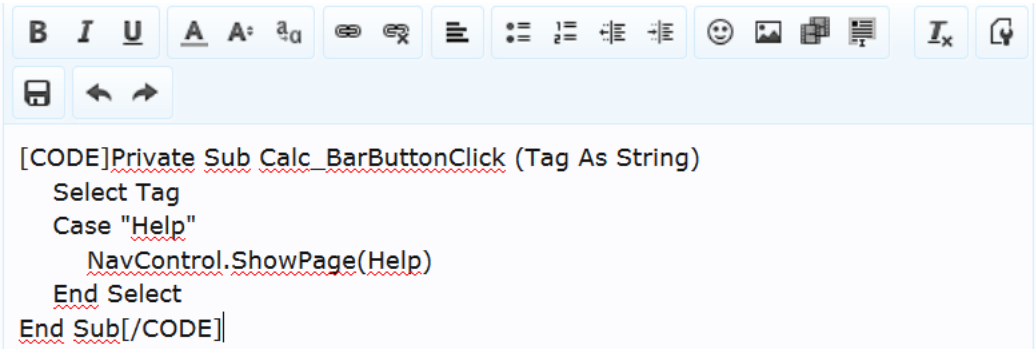
select Code.

You will get this window.

Copy the code from the IDE, or enter it, and click on Insert.



You will see this in the editor:



The code is added between the CODE tags [CODE][/CODE] and formatted.
You can also add the CODE tags and the code directly in the editor.

And the result in the forum.

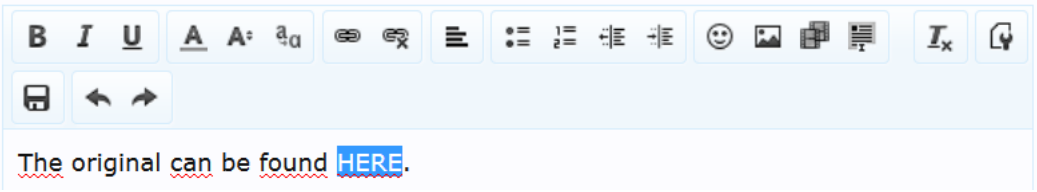
Code:

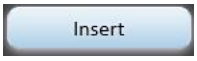
```
Private Sub Calc_BarButtonClick (Tag As String)
    Select Tag
    Case "Help"
        NavControl.ShowPage(Help)
    End Select
End Sub
```

16.5.4 Links

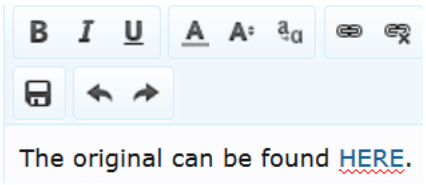
You can add links in the editor.

Select the text and click on  .



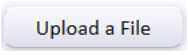
Enter or copy the link and click on 

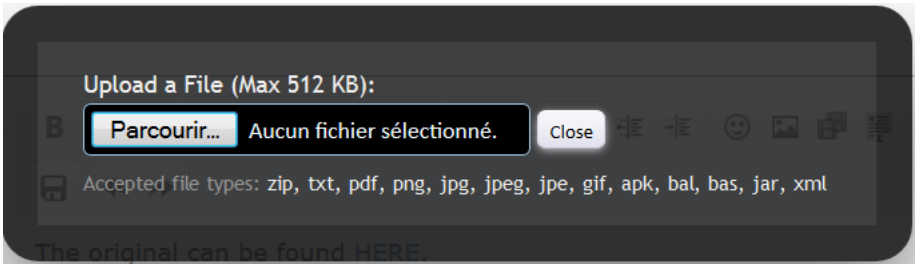
.



And the result

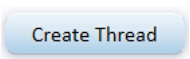
16.5.5 Add files.

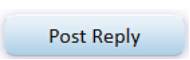
To add files to the thread or post click on 



Select the files in the files explorer.

16.5.6 Send the thread or post

Click on 

to create the thread or on 

to send the post.

17 Code snippets

17.1 Submit events to underlaying nodes.

Sometimes it is necessary to submit events to underlaying nodes, especially with Panes. This is not set by default like in B4A.
If you want to consume events in B4A you must add an empty event routine.

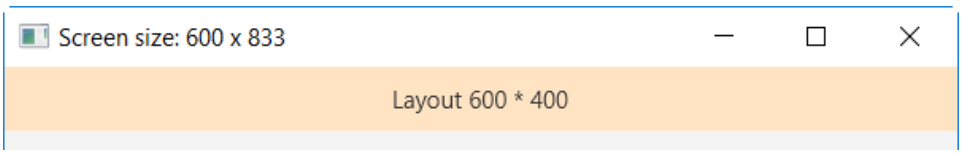
The code below does it, this is like `Panel.UserInteractionEnabled` in B4i.
Needs the JavaObject library.

```
Private jo = Pane As JavaObject
jo.RunMethod("setMouseTransparent", Array As Object(True))
```

17.2 Form styles, only close button or no title bar

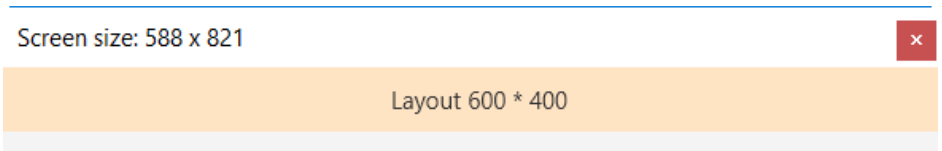
Sometimes it can be useful to remove the resize buttons in the title bar or even to remove it.

Standard:



This code shows only the close button:

```
MainForm.SetFormStyle("UTILITY")
MainForm.Show
```



This code removes the title bar:

```
MainForm.SetFormStyle("UNDECORATED")
MainForm.Show
```