B4x Booklets

CustomViews

Last update: 2017.02.15

 Edition 1.0

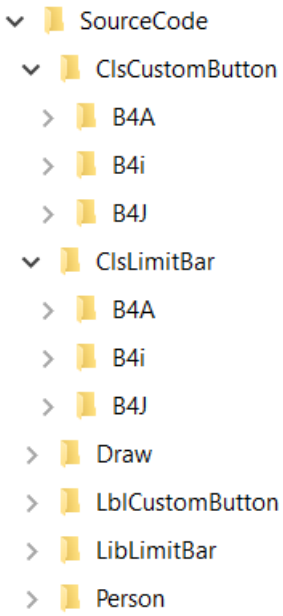
Main contributors: Klaus Christl (klaus) Erel Uziel (Erel).

1 General information

This guide is dedicated for more advanced users and treats the CustomView topic.

It covers B4A, B4i and B4J.

All the source code and files needed (layouts, images etc) for the example projects in this guide are included in the SourceCode folder.
For each project, there are three subfolders, one for each operating system.



2 Class modules

In B4x, you can use two types of Class Modules:

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

2.1 Getting started

Classes definition from Wikipedia:

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

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

Let's start with an example, the source code: *Person* in the / Person folder.

In the Person module

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

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

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

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

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

Main module.

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

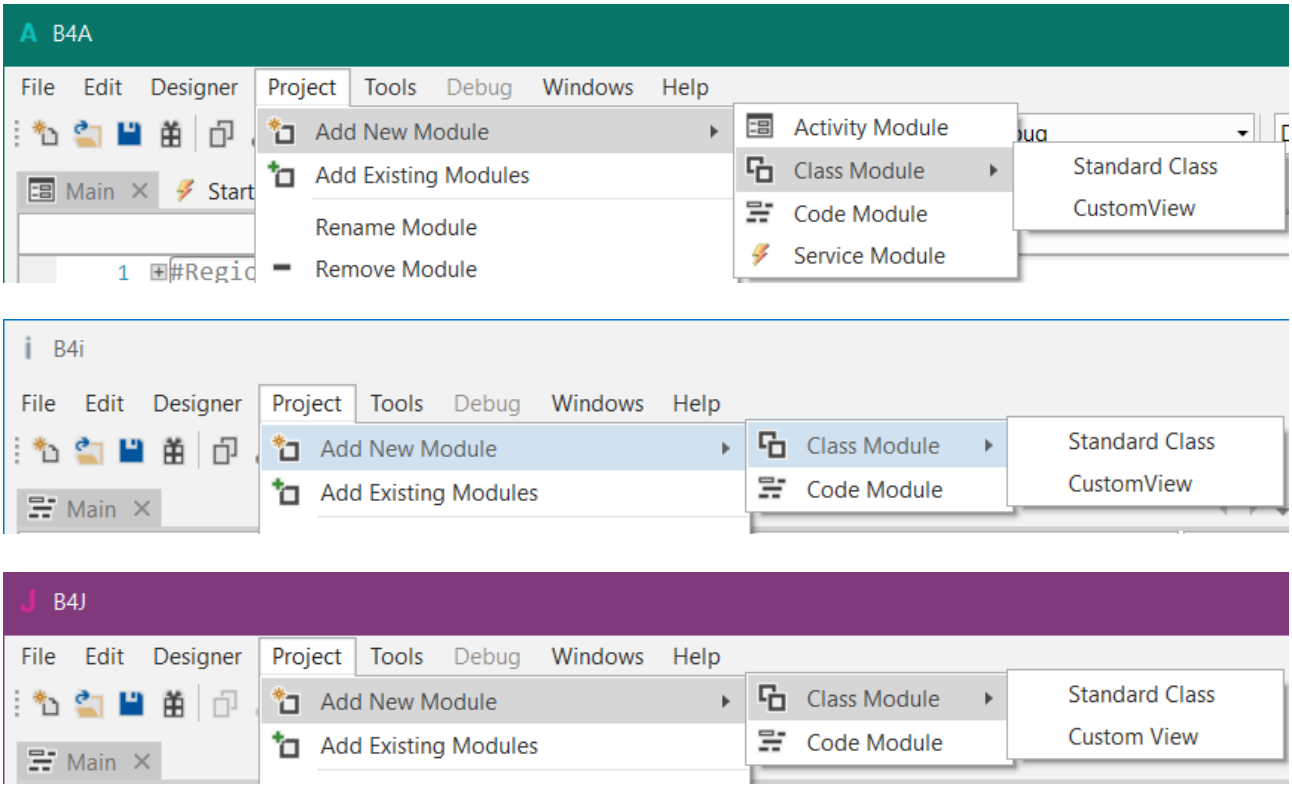
I will start by explaining the differences between classes, code modules and types.

Similar to types, classes are templates. From this template, you can instantiate any number of objects.
The type fields are similar to the classes global variables. However, unlike types which only define the data structure, classes also define the behaviour. The behaviour is defined in the classes' subs.

Unlike classes which are a template for objects, code modules are collections of subs. Another important difference between code modules and classes is that code modules always run in the context of the calling sub. The code module doesn't hold a reference to any context. For that reason it is impossible to handle events or use CallSub with code modules.
Classes store a reference to the context of the module that called the Initialize sub. This means that classes objects share the same life cycle as the module that initialized them.

2.1.1 Adding a class module

Adding a new or existing class module is done by choosing Project > Add New Module > Class module or Add Existing module.
Like other modules, classes are saved as files with *bas* extension.



There are two class module types:
Standard Class
CustomView

2.1.2 Polymorphism

Polymorphism allows you to treat different types of objects that adhere to the same interface in the same way.
B4x polymorphism is similar to the Duck typing concept.

As an example we will create two classes named: Square and Circle.
Each class has a sub named Draw that draws the object to a canvas:
Source code *Draw* in the Draw folder.

The code below is the B4A code.

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

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

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


'Class Circle module
Sub Class_Globals
   Private mx, my, mRadius As Int
End Sub

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

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

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

```
Sub Process_Globals
   Public Shapes As List
End Sub

Sub Globals
   Private cvs As Canvas
End Sub

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

   DrawAllShapes
End Sub

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

As you can see, we do not know the specific type of each object in the list. We just assume that it has a Draw method that expects a single Canvas argument. Later we can easily add more types of shapes.
You can use the `SubExists` keyword to check whether an object includes a specific sub.

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

2.1.3 Self-reference

The `Me` keyword returns a reference to the current object. `Me` keyword can only be used inside a class module.
Consider the above example. We have passed the Shapes list to the Initialize sub and then add each object to the list from the Initialize sub:

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

2.1.4 Activity object B4A only

This point is related to the Android Activities special life cycle.
Make sure to first read the activities and processes life-cycle tutorial.

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

3 Standard class

3.1 Standard class structure

Default template of a standard class:

B4A and B4i

```
Sub Class_Globals

End Sub

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

End Sub
```

B4J

```
Sub Class_Globals
   Private fx As JFX
End Sub

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

End Sub
```

Only two routines are predefined:

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

`Sub` **`Initialize`** - A class object must be initialized before you can call any other sub. Initializing an object is done by calling the Initialize sub. When you call Initialize you set the object's context (the parent object or service).
Note that you can modify this sub signature and add arguments as needed.

Example: Person class module
The source codes are in the Person folder.

The code is the same for all three B4x platforms (B4A. B4i, B4J).

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

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

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

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

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

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

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

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

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

4 CustomViews

With custom view classes, you can create your own custom views which can be based on other standard or custom views, with more functions.

4.1 CustomView class structure

Several declarations and routines are predefined:

Default template of a CustomView class:

```
'Custom View class
#Event: ExampleEvent (Value As Int)
#DesignerProperty: Key: BooleanExample, DisplayName: Boolean Example, FieldType:
Boolean, DefaultValue: True, Description: Example of a boolean property.
#DesignerProperty: Key: IntExample, DisplayName: Int Example, FieldType: Int,
DefaultValue: 10, MinRange: 0, MaxRange: 100, Description: Note that MinRange and
MaxRange are optional.
#DesignerProperty: Key: StringWithListExample, DisplayName: String With List,
FieldType: String, DefaultValue: Sunday, List:
Sunday|Monday|Tuesday|Wednesday|Thursday|Friday|Saturday
#DesignerProperty: Key: StringExample, DisplayName: String Example, FieldType: String,
DefaultValue: Text
#DesignerProperty: Key: ColorExample, DisplayName: Color Example, FieldType: Color,
DefaultValue: 0xFFCFDCDC, Description: You can use the built-in color picker to find
the color values.
#DesignerProperty: Key: DefaultColorExample, DisplayName: Default Color Example,
FieldType: Color, DefaultValue: Null, Description: Setting the default value to Null
means that a nullable field will be displayed.
Sub Class_Globals
   Private mEventName As String 'ignore
   Private mCallBack As Object 'ignore
   Private mBase As Panel
   Private Const DefaultColorConstant As Int = -984833 'ignore
End Sub

Public Sub Initialize (Callback As Object, EventName As String)
   mEventName = EventName
   mCallBack = Callback
End Sub

Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
   mBase = Base

End Sub

Public Sub GetBase As Panel
   Return mBase
End Sub
```

Additional routine in B4i and B4J:

```
Private Sub Base_Resize (Width As Double, Height As Double)

End Sub
```

This event routine is raised every time a resize occurs, device rotation in B4i or Form resize in B4J.

4.1.1 Event declarations

You should add Event declarations if you compile the custom view into a library.
If the event routine has parameters, these must also be declared.

```
#Event: ExampleEvent (Value As Int)
#RaisesSynchronousEvents: ExampleEvent
```

important for intellisense.
important for library compilation.

4.1.2 Designer properties declarations

```
#DesignerProperty: Key: BooleanExample, DisplayName: Boolean Example, FieldType:
Boolean, DefaultValue: True, Description: Example of a boolean property.
```

You can add custom properties for the Designer.

More details in the chapter Custom view in the Designer.

4.1.3 Global variable declarations

In this routine, you should declare all global variables used in the class.

The variables below are mandatory.

```
Sub Class_Globals
   Private EventName As String 'ignore
   Private CallBack As Object  'ignore
   Private mBase As Panel
End Sub
```

`EventName`	Event name used for the events in the code, same as for standard views.
`CallBack`	Module where the class is declared, used for event calls.
`mBase`	Main panel of the custom view.

You can, if you want, change the name of the base panel.

What is this for `'ignore` ?
It avoids a warning of the compiler that these variables are unused.

Variables only used in the class should be declared as `Private`.
If you want to have access to variables from other modules you must declare them as `Public`.

4.1.4 Initialization routine

The initialize routine initiates a new instance of the custom view.

```
Public Sub Initialize (Callback As Object, EventName As String)
   mCallBack = Callback
   mEventName = EventName
End Sub
```

The two variables will be used to call event routines in the module where the custom view is initialized.

You should not modify its signature.

Example:

```
' if a callback routine exists in the calling module we call it
If SubExists(mCallback, mEventName & "_ValuesChanged") Then
   CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
End If
```

4.1.5 Designer support routine

This routine assures the support for the Designer, it is called directly after the Initialize routine of the custom view class.

You should not modify its signature.

```
Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
   mBase = Base
End Sub
```

`Base` Is the base panel defined in the Designer, it holds the Left, Top, Width, Height and Parent properties of the custom view. The Base panel can be used or not.

`Lbl` Is a Label which holds all the text properties defined in the Designer.
This Label can be used or not.

`Props` Is a Map holding additional properties.
The ones you defined yourself in the designer properties definition.

4.1.6 Routine to get the base Panel

You can use this routine if you want to access the base panel / pane from other modules.

B4A / B4J

```
Public Sub GetBase As Panel
   Return mBase
End Sub
```

B4J

```
Public Sub GetBase As Pane
   Return mBase
End Sub
```

In the calling module:

```
Private pnlClass As Panel
pnlClass = clsTest.GetBase
```

```
Private pnlClass As Pane
pnlClass = clsTest.GetBase
```

4.2 Adding a custom view by code

To offer the possibility to add the custom view by code you must add a routine in the class which adds the custom view onto a parent view which can be either for:

B4A	an Activity or a Panel.	`Public Sub AddToParent(Parent As Activity,`
B4i	a Panel.	`Public Sub AddToParent(Parent As Panel,`
B4J	a Pane.	`Public Sub AddToParent(Parent As Pane,`

Example:

```
Public Sub AddToParent(Parent As Activity, Left As Int, Top As Int, Width As Int,
Height As Int)
   mBase.Initialize("mBase")
   Parent.AddView(mBase, Left, Top, Width, Height)
End Sub
```

`Parent`	is the parent view which can be an Activity, Panel or a Pane.
`Left`	is the `Left` property.
`Top`	is the `Top` property.
`Width`	is the `Width` property.
`Height`	is the `Height` property.

You can add other parameters or properties to the routine if necessary.

And in the calling module:

B4A / B4i

```
Private clsTest2 As ClsCustomView

clsTest2.Initialize(Me, "clsTest2")
clsTest2.AddToParent(MyPanel, 10dip, 10dip, 200dip, 50dip)
```

B4J Pane instead of Panel and no dip values.

```
Private clsTest2 As ClsCustomView

clsTest2.Initialize(Me, "clsTest2")
clsTest2.AddToParent(MyPane, 10, 10, 200, 50)
```

4.3 Add properties

Property routines can be added which work like any property of the standard views.

These properties can be read and or set.

To read a property you must add a routine beginning with `get`, **lower case** and the property name.
Examples:
Get the *Left* Property.

```
'gets the Left property
Public Sub getLeft As Int
   Return ltbPanelBack.Left
End Sub
```

Get the custom *Max* property.

```
'gets the Max value
Public Sub getMax As Int
   Return MaxValue
End Sub
```

To set a property you must add a routine beginning with `set`, **lower case** and the property name.
Examples:
Set the *Left* Property.

```
'sets the Left property
Public Sub setLeft(Left As Int)
   ltbPanelBack.Left = Left
End Sub
```

Set the custom *Max* property.

```
'sets the Max value
Public Sub setMax(MaxValue As Int)
   mMaxValue = MaxValue
   Scale = (x1 - x0) / mMaxValue
End Sub
```

If you define only a `get` routine the property is read only.
If you define only a `set` routine the property is write only.
If you define both a `set` and a `get` routine, the property is write and read.

Note:
`Public Sub setMax` and `Public Sub SetMax` are not the same!

`Public Sub setMax` is considered as a Property.

`Public Sub SetMax` is considered as a Public Subroutine.

4.4 Custom view and custom properties in the Designer

You can add code to make custom properties visible in the Designer.
The images below are from the DefaultLayout project in the SourceCode\ DefaultLayout folder.
Only the B4A version.

On the top of the code you must include declaration lines. The default layout of a custom view class includes these example declarations:

```
#DesignerProperty: Key: BooleanExample, DisplayName: Boolean Example, FieldType:
Boolean, DefaultValue: True, Description: Example of a boolean property.
#DesignerProperty: Key: IntExample, DisplayName: Int Example, FieldType: Int,
DefaultValue: 10, MinRange: 0, MaxRange: 100, Description: Note that MinRange and
MaxRange are optional.
#DesignerProperty: Key: StringWithListExample, DisplayName: String With List,
FieldType: String, DefaultValue: Sunday, List:
Sunday|Monday|Tuesday|Wednesday|Thursday|Friday|Saturday
#DesignerProperty: Key: StringExample, DisplayName: String Example, FieldType: String,
DefaultValue: Text
#DesignerProperty: Key: ColorExample, DisplayName: Color Example, FieldType: Color,
DefaultValue: 0xFFCFDCDC, Description: You can use the built-in color picker to find
the color values.
#DesignerProperty: Key: DefaultColorExample, DisplayName: Default Color Example,
FieldType: Color, DefaultValue: Null, Description: Setting the default value to Null
means that a nullable field will be displayed.
```

Each property declaration is made of several fields, the following fields are required:

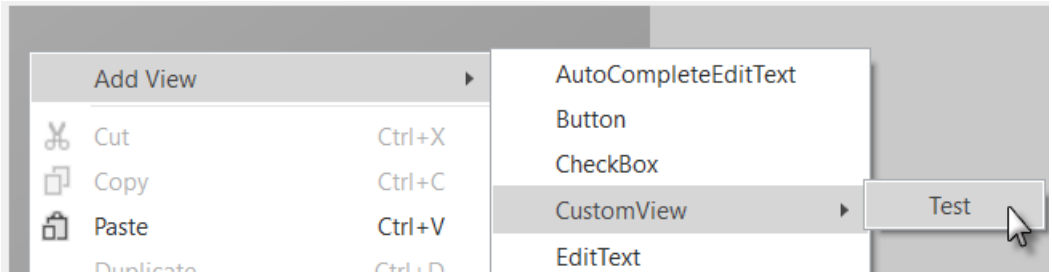
`Key`	Is the key value for the Map. This will be used to get the value from the Props map.
`DisplayName`	Is the name displayed in the Designer property grid.
`FieldType`	Is the type of the field. Possible values: String, Int, Double, Boolean or Color.
`DefaultValue`	Is the default value which is set in the Designer.

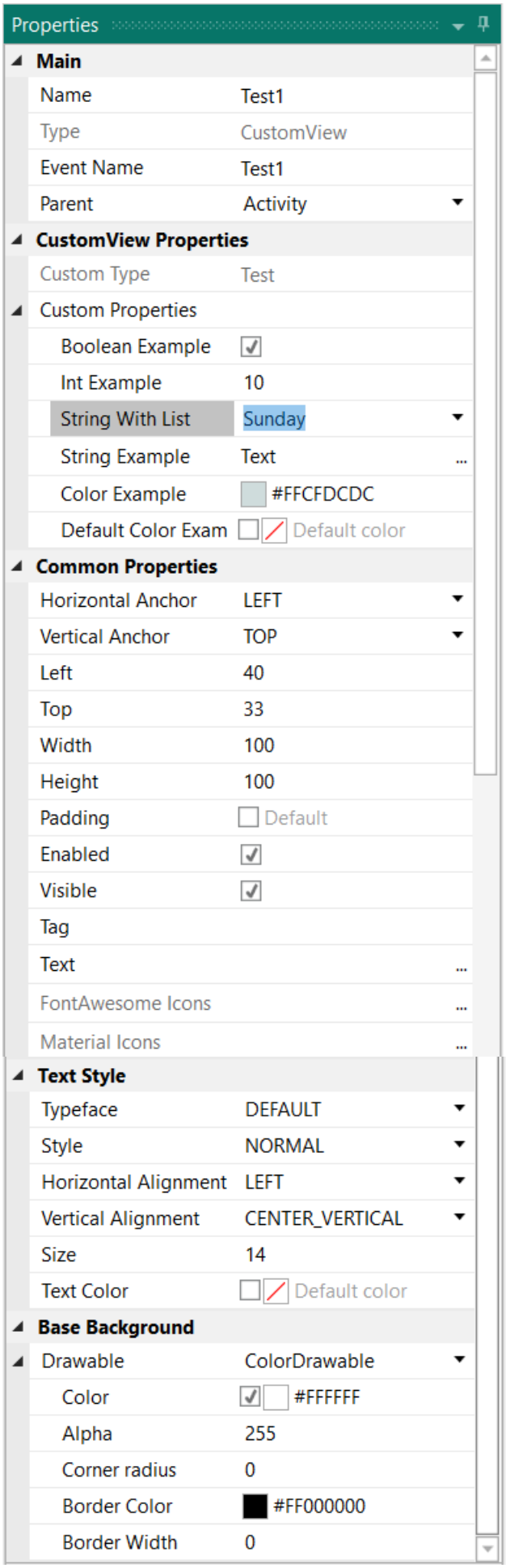
Optional fields:

`Description`	Is the explanation text displayed in the Designer.
`MinRange` / `MaxRange`	Minimum and maximum numeric values allowed.
`List`	A pipe (|) separated list of items from which the developer can choose (should be used with string fields).

In the Designer, you can add a CustomView like this:

Right click in the screen area, select Add View and select CustomView.
Select the custom from the list of available custom views *ClsCustomView* in the example.



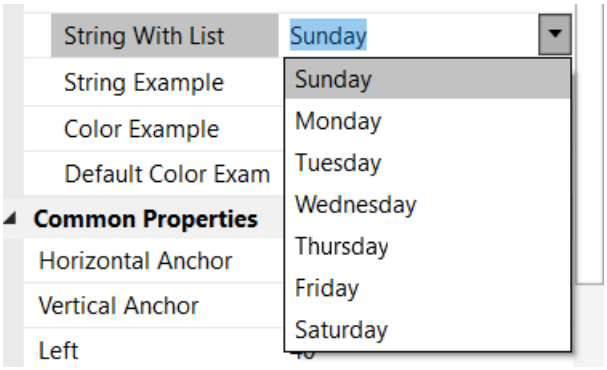


In the Properties window, you find all the properties for the selected custom view.
Imges B4A.

Custom properties:

Here we see the six custom properties declared on top of the Class code.

Example with the String With List property.



Common Properties:
The common properties like any view.

Text Style:
The properties are set to the `Lbl` Label of the class.

Base Background:
Background of the base panel `Base`.

To access the custom properties you must use the Props Map in the **DesignerCreateView** routine.

Variable declaration:
```
Private BooleanTest As Boolean
Private IntTest As Int
Private Day As String
Private StringTest As String
Private ColorTest As Int
Private DefaultColorTest As Int
```

And the DesignerCreateView routine:

```
Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
   mBase = Base

   BooleanTest = Props.Get("BooleanExample")
   IntTest = Props.Get("IntExample")
   Day = Props.Get("StringWithListExample")
   StringTest = Props.Get("StringExample")
   ColorTest = Props.Get("ColorExample")
   DefaultColorTest = Props.Get("DefaultColorExample")
End Sub
```

B4J the Base declaration is different, `Pane` instead of `Panel`.
```
Public Sub DesignerCreateView (Base As Pane, Lbl As Label, Props As Map)
```

You can get properties of the Base Panel / Pane like:

B4A / B4i / B4J
```
Private mWidth As Int
mWidth = Base.Width

Private mHeight As Int
mHeight = Base.Height
```

You can get text properties from the Lbl Label like:

B4A
```
Private mText As String
mText = Lbl.Text

Private mTextColor As Int
mTextColor = Lbl.TextColor

Private mTextSize As Float
mTextSize = Lbl.TextSize
```

B4i
```
Private mText As String
mText = Lbl.Text

Private mTextColor As Int
mTextColor = Lbl.TextColor

Private fnt As Font
Private mTextSize As Float
fnt = Lbl.Font
mTextSize = fnt.Size
```

B4J
```
Private mText As String
mText = Lbl.Text

Private mTextColor As Paint
mTextColor = Lbl.TextColor

Private mTextSize As Double
mTextSize = Lbl.TextSize
```

4.5 Complie to a library

In B4A, B4i and B4J you can compile your project, or part of it to a regular library.

Why should we compile a library?

- Break large projects into several smaller (more maintainable) projects.
- Build reusable components and use them from any number of projects.
- Share components with other developers without sharing the source code.
- Create different versions of your application (free, pro...) by referencing the same "core" library.

The output of library compilation is:

- Two files for B4A and B4J:
  A jar file with the compiled code and a xml file that includes the metadata that is required by the IDE.
  These two files are automatically saved in the additional libraries folders.
- Three files for B4i:
  The xml file like above which is copied to the additional libraries folders.
  And, .a and .h files are created in the Mac Libs folder.

You can exclude other modules as well with the ExcludeFromLibrary attribute.

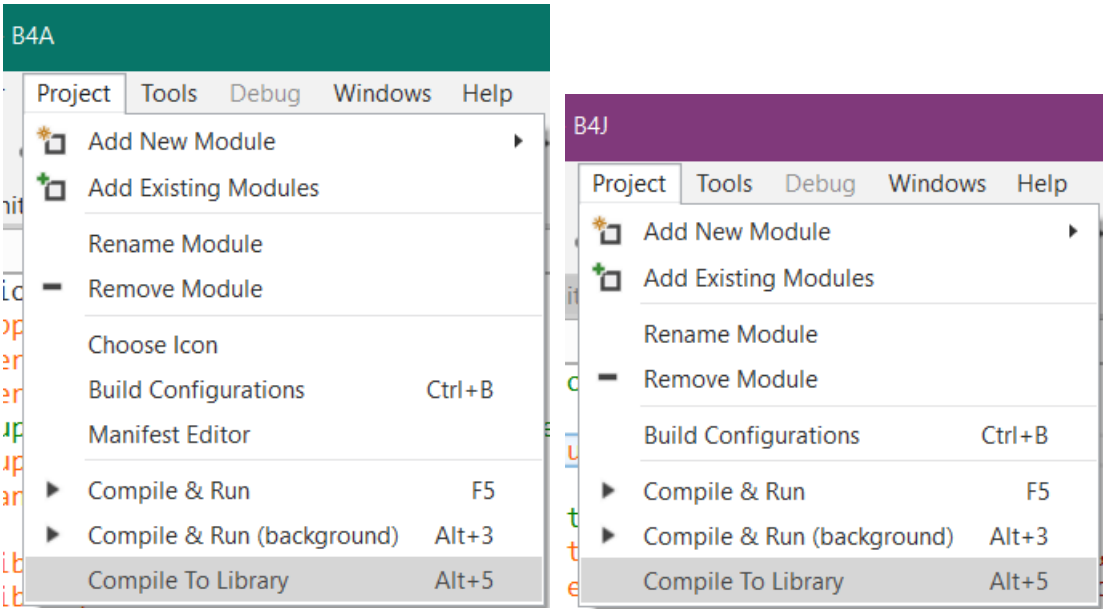
```
#ExcludeFromLibrary: True
```

The Main module and the other excluded modules can be used to test the library.

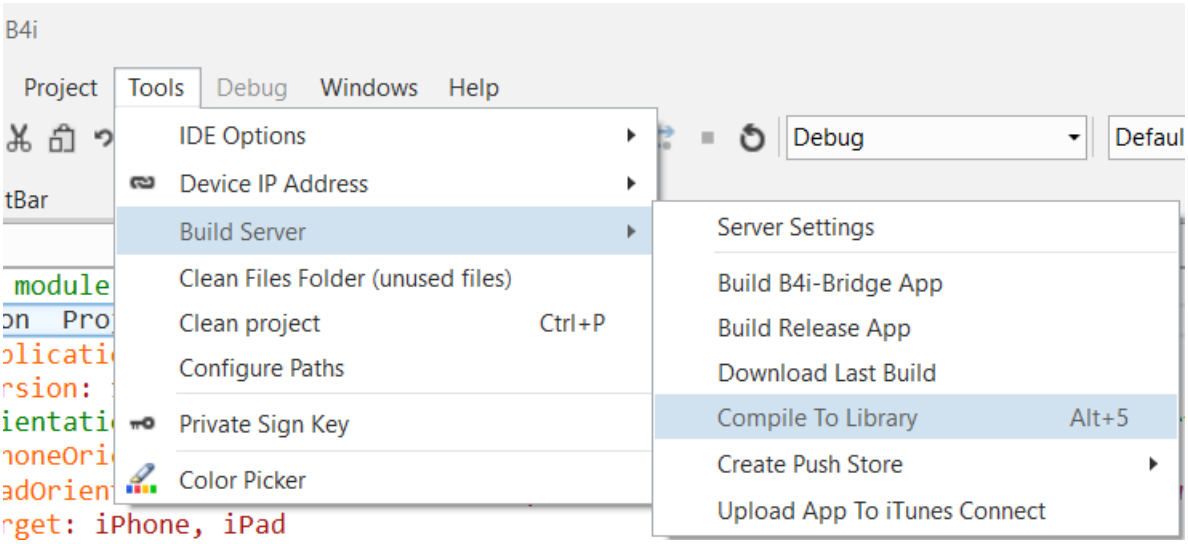
You can reference the library from other projects and access the same functionality as in the original project.

Compiling to a library is quite simple.
Under Project menu there is the compile option - "Compile To Library (Alt + 5)".
When you choose this option all the modules **except** of the main activity are compiled into a library.

B4A **B4J**

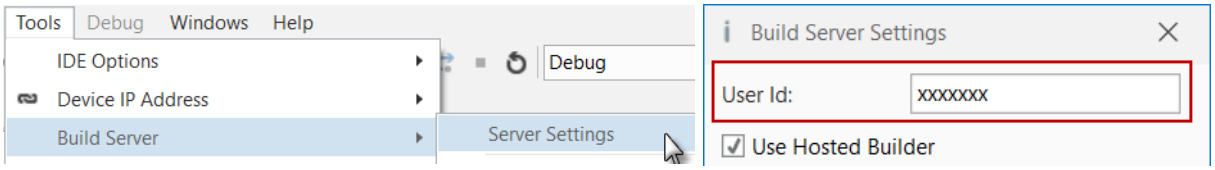


B4i



Note: If you are using the hosted builder then you need to first receive a permission to compile a specific library. Please contact support@basic4ppc.com and send your hosted builder id and the library name.

You find the hosted builder id in Tools / Build Server / Server Settings.



4.5.1 Library specific attributes

The following attributes are specific for library compilation:

Main module:

Project attributes (placed on top of the code in the Main module):
`#LibraryName`
- The compiled library name. Sets the library name.
`#LibraryAuthor`
- The library author. This value is added to the library xml file.
`#LibraryVersion`
- A number that represents the library version. This number will appear next to the library name in the libraries list.

Example, LimitBar projects.

B4A
```
#LibraryName: LimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```
B4i
```
#LibraryName: iLimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```
B4J
```
#LibraryName: jLimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```

All modules:
`#ExcludeFromLibrary` - Whether to exclude this module during library compilation. Values: True or False. Note that the Main activity is always excluded.

CustomView classes:
`#Event` - Adds an event to the list of events. This attribute can be used multiple times.
The parameters must be included.
Note that the events list only affects the IDE events autocompletion feature.
`#RaisesSynchronousEvents` - Needed if you compile the CustomView into a library.
It is used for the Rapid Debugger. You need one for each event.
Details in the LimitBar project here.

Example, LimitBar projects.
```
#Event: ValuesChanged(LimitLeft As Int, LimitRight As Int)
#RaisesSynchronousEvents: ValuesChanged
```
`ValuesChanged` is the name of the event for its call.

If you have other modules in the same project which should not be in the library, you must add
```
#ExcludeFromLibrary: True
```

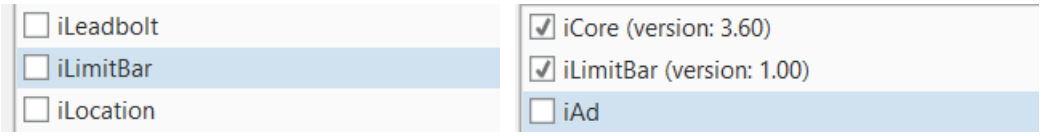
Notes

- You should right click on the libraries list and choose Refresh after a library update.
- CallSub / CallSubDelayed - The first parameter for these keywords is a reference to the target module. When working with modules that reside in a library you should pass the module reference and not the module name as string (this is the better way to reference all modules in all cases).
- Code obfuscation - Libraries can be obfuscated during library compilation. Strings will not be obfuscated in this mode.
- Services that host home screen widgets cannot be compiled into a library.

The library files are automatically saved in the Additional Libraries folder.

You can see it in the Libraries Manager Tab.

Right click somewhere in the Libraries Manager Tab and click on Refresh.



Example with the B4i LimitBar project.

The library name is the name you entered in `#LibraryName: iLimitBar`.

When you select the library, it moves on top of the list and shows the version number.
You should not have the modules and the library in the same project!

When you declare a custom view, you must use the Module/Object name:
Library: iLimitBar Object: LimitBar

```
Private ltbTest, ltbTest1 As LimitBar
```

4.6 Program flow

Below a comparison of the program flow with two custom views, one added in the Designer and the other in the code.

B4A	B4i	B4J
0 Process_Globals	0 Process_Globals	0 Process_Globals
0 Globals	0 Application_Start	0 AppStart
0 Activity_Create	1 Class_Globals	1 Class_Globals
1 Class_Globals	1 Class Initialize	1 Class Initialize
1 Class Initialize	1 DesignerCreateView	1 DesignerCreateView
1 DesignerCreateView	1 Base_Resize	1 Base_Resize
2 Class_Globals	2 Class_Globals	2 Class_Globals
2 Class Initialize	2 Class Initialize	2 Class Initialize
2 AddToParent	2 AddToParent	2 AddToParent
0 Activity_Resume	0 Page1_Resize	0 MainForm_Resize
Turn device	**Turn device**	**Resize Main Form**
0 Activity_Pause	0 Page1_Resize	0 MainForm_Resize
1 Class_Globals	1 Base_Resize	1 Base_Resize
1 Class Initialize		
1 DesignerCreateView		
2 Class_Globals		
2 Class Initialize		
2 AddToParent		
0 Activity_Resume		

0 = Main
1 = CustomView Designer
2 = CustomView code

Note: The B4A example project above has no Starter service module.

We notice that when we start the program the flow is the same in B4i and B4J but in B4A it is a bit different.

When we turn the B4i device or resize the B4J form the program flow is the same.
In B4A it is quite different.

In B4A, the Activity is destroyed and recreated.
In B4i and B4J, the layout remains and a Resize event is raised.

The advantage of adding custom views in the Designer, in B4i and B4J, is that it handles the resize event and reapplies the anchors and designer script (and variant changes).
In B4A this is also executed because the Activity is recreated at every change.
This is shown in the LimitBar projects.

4.7 Intellisense help

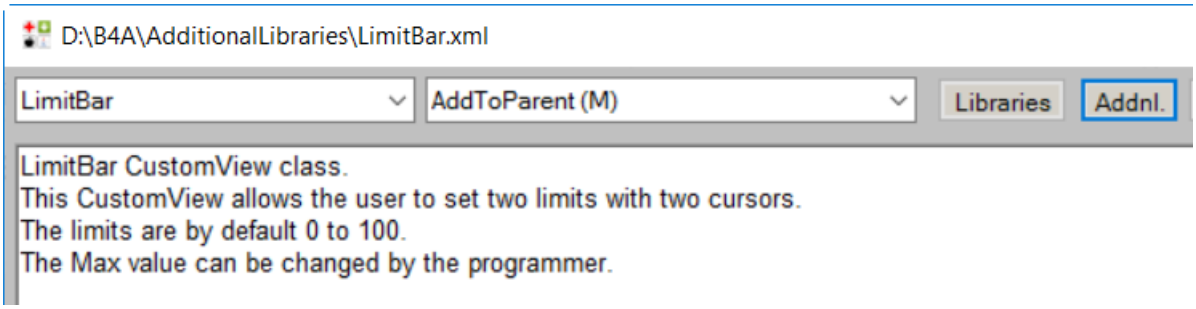
It is adviced to add help comments in the code for the users of your library.

4.7.1 Comments before Sub Class_Globals

Comments before `Sub Class_Globals` are considered as the help header when the class is compiled to a Library.

```
'LimitBar CustomView class.
'This CustomView allows the user to set two limits with two cursors.
'The Min value is 0 and the Max value is 100.
'The Max value can be changed by the programmer.
Sub Class_Globals
```

Example with the B4x Help Viewer and the LimitBar library.

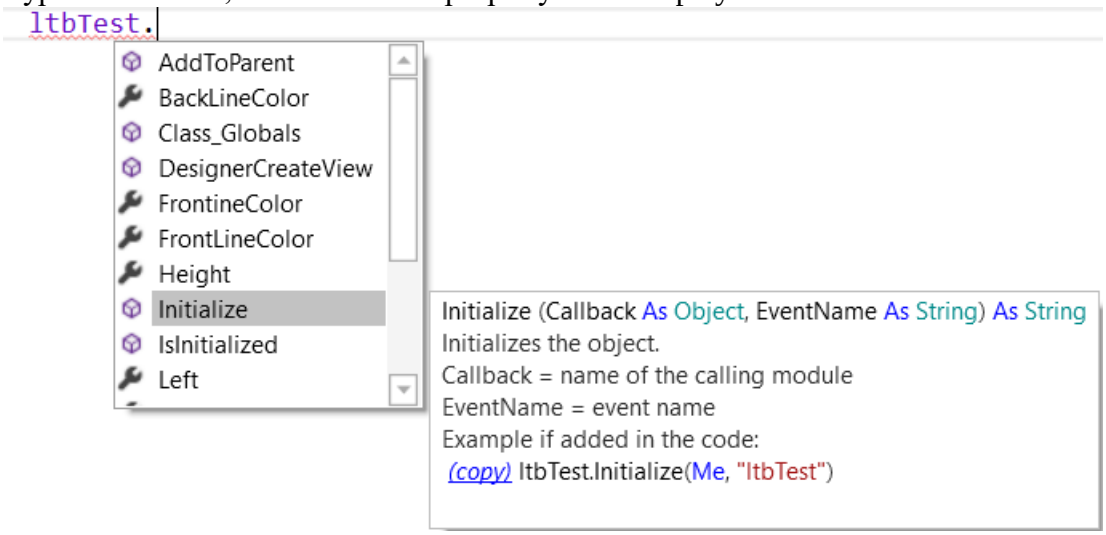


4.7.2 Comments before a routine

Comments before a routine are considered as intellisense help.

```
'Initializes the object.
'Callback = name of the calling module
'EventName = event name
'Example if added in the code:
'<code>ltbTest.Initialize(Me, "ltbTest")'</code>
Public Sub Initialize(Callback As Object, EventName As String)
```

Type '`lbtTest.`' , the method and property list is displayed.

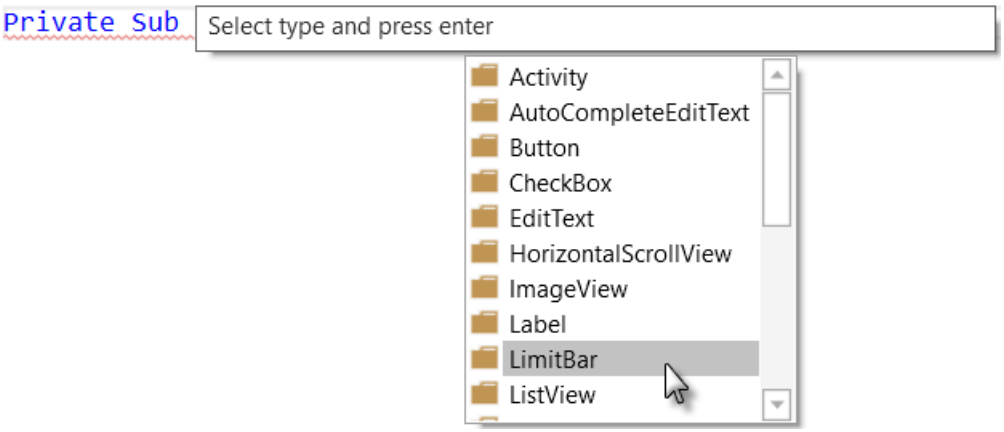


4.7.3 Comments before an event routine

Events declared on top of the code in the class module with `#Event:` are displayed as intellisense when the class is compiled to a Library.

```
'Custom View class LimitBar
'Events declaration
#Event: ValuesChanged(LimitLeft As Int, LimitRight As Int)
```

When you use the library in another project, type '`Public Sub '` (with a space at the end) and press on Tab to show the objects list.



Select LimitBar.

Select ValuesChanged(LimitLeft As Int, LimitRight As Int).

```
Private Sub EventName_ValuesChanged(LimitLeft As Int, LimitRight As Int)

End Sub
```

The sub frame is added.

```
Private Sub ltbTest1_ValuesChanged(LimitLeft As Int, LimitRight As Int)

End Sub
```

Enter the LimitBar name and press Return and the sub frame is finished.

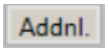
```
Private Sub ltbTest1_ValuesChanged(LimitLeft As Int, LimitRight As Int)

End Sub
```

4.8 Help tool

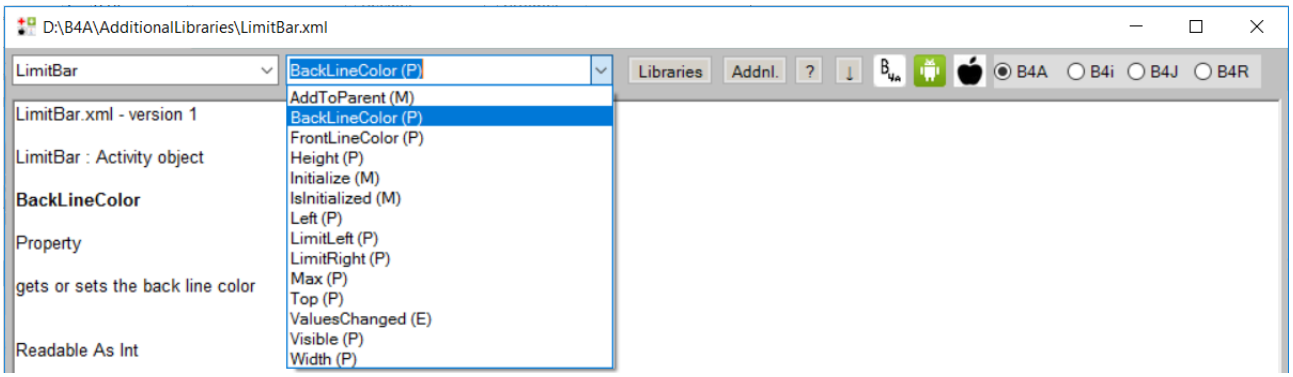
If you have the B4x Help Viewer you can look at the help for the library.

Example with the B4A LimitBar library.

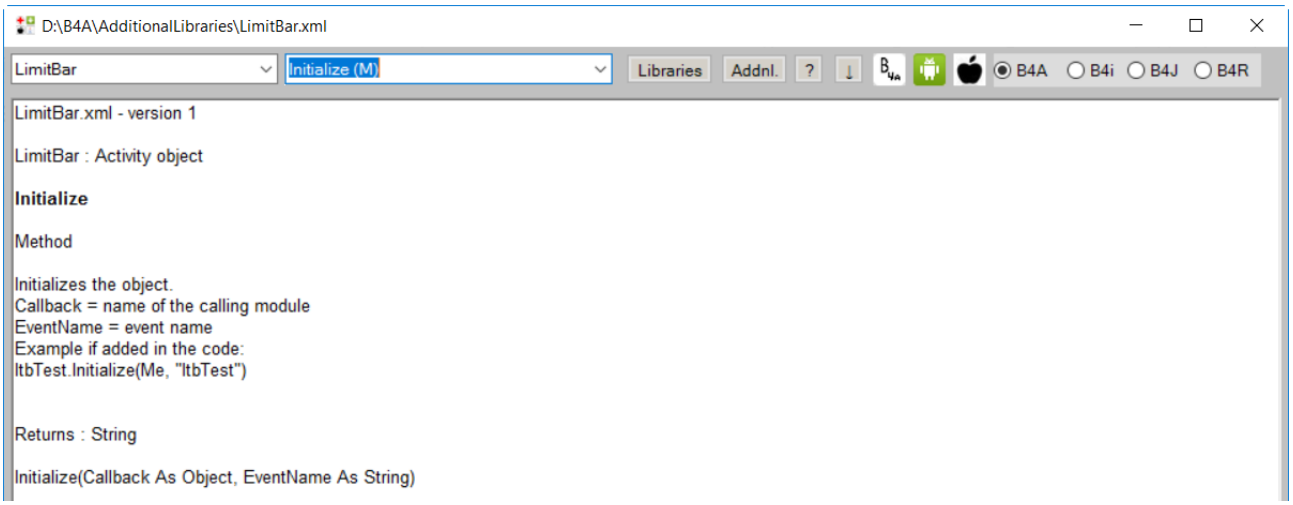
Click on 

to load LimitBar.xml.

And the result.



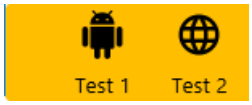
Then you can select any method, event or property in the list to show the help.



5 First example CustomButton

We will make a simple CustumButton.

B4A B4i B4J



The button has a transparent base Panel (B4A, B4i) / Pane (B4J) plus one Label with a Material Icon and a second Label with text.

The CustumButton can be added in the Designer or in the code.

The CustumButton has two events Click and LongClick.
B4J has no Click nor LongClick event, Click is called Action or MouseClicked.
I kept the Click name and added the LongClick event.

The code is kept simple and minimalistic, the main goal here is to show the principle.
Feel free to add more properties and functionalities.

5.1 Event declarations

First, we declare the events on top of the code.
Needed when the class is compiled to a library.

```
'CustomButton Class
#Event: Click
#Event: LongClick
#RaisesSynchronousEvents: Click
#RaisesSynchronousEvents: LongClick
```

5.2 Custom properties for the Designer

We have only one custom property: Text.
This is the text below the icon.

The other properties, like icon character and text color, are defined in the Designer or in the AddToParent routine.
The icon and text sizes are calculated in the class code acording to the button height.

```
#DesignerProperty: Key: Text, DisplayName: Text, FieldType: String, DefaultValue: Text,
Description: Text at the bottom of the button.
```

5.3 Class help header

We add a header text, just before `Sub Class_Globals`, explaining the purpose of the button as a help for the user.

```
'CustomButton is a button based on a Panel with two Labels
'one with a Material Icon and the other with text.
'It has two events: Click and LongClick.
Sub Class_Globals
```

5.4 Global variables

We define the global variables below. There are some differences between the three operating systems.

B4A

```
Sub Class_Globals
   Private mEventName As String
   Private mCallBack As Object
   Private mBase As Panel

   Private mLeft, mTop, mWidth, mHeight As Int
   Private mText, mIcon As String
   Private mIconTypeface As Typeface
   Private mTextColor As Int
   Private mIconTextSize, mTextSize As Float
   Private mTag As Object

   Private mLabel, mIconLabel As Label
   Private mParent As Panel
End Sub
```

B4i `Typeface` is replaced by `Font`.

```
Private mIconTypeface As Font
```

B4J `Typeface` is replaced by `Font`. For the color `Int` is replaced by `Paint`.
We add another variable for the LongClick event timing

```
Private mIconTypeface As Font
Private mTextColor As Paint
Private mClickTime As Long      'used to distinguish Click and LongClick
```

5.5 Initialize routine

We get the CallBack module and EventName and initialize three default values.

B4A / B4i

```
Public Sub Initialize (Callback As Object, EventName As String)
   mEventName = EventName
   mCallBack = Callback

   mIcon = Chr(0xE859)
   mText = "Test"
   mTextColor = Colors.Black
End Sub
```

B4J the color is `fx.Colors.Black` instead of `Colors.Black`.

```
mTextColor = fx.Colors.Black
```

5.6 DesignerCreateView routine

Here we get the properties from the Designer.
We initialize mBase and add it to the parent view.
We need this because we use event routines of the base panel / pane.
Just setting `mBase = Base` does not enable to use events.

B4A

```
Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
   mLeft = Base.Left
   mTop = Base.Top
   mWidth = Base.Width
   mHeight = Base.Height
   mIcon = Lbl.Text
   mText = Props.Get("Text")
   mBase.Initialize("mBase")
   mParent = Base.Parent
   mParent.AddView(mBase, mLeft, mTop, mWidth, mHeight)

   mTextColor = Lbl.TextColor
   mIconTypeface = Lbl.Typeface
   mTag = Base.Tag

   InitClass
End Sub
```

B4i `Typeface` is replaced by `Font`.

```
mIconTypeface = Lbl.Font
```

Plus, the Base_Resize routine.

B4J `Typeface` is replaced by `Font`.

```
mIconTypeface = Lbl.Font
```

Plus, the Base_Resize routine.
The `InitClass` routine is moved to the **`Base_Resize`** routine.

5.7 Base_Resize routine B4i / B4J only

The Base_Resize routine is called every time a resize is done.
Device orientation change in B4i or a Form resize in B4J.

B4i

```
Private Sub Base_Resize (Width As Double, Height As Double)
   mHeight = Height
   mWidth = Width
End Sub
```

B4J

```
Private Sub Base_Resize (Width As Double, Height As Double)
   mWidth = Width
   mHeight = Height
   mBase.PrefWidth = mWidth
   mBase.PrefHeight = mHeight

   InitClass
End Sub
```

5.8 AddToParent routine

This routine is needed when we add the CustomButton in the code.
We memorize the position, dimensions and properties.
And call InitClass

B4A / B4i

```
Public Sub AddToParent(Parent As Panel, Left As Int, Top As Int, Width As Int, Height
As Int, TextColor As Int, Icon As String, Text As String)
   mLeft = Left
   mTop = Top
   mWidth = Width
   mHeight = Height
   mParent = Parent

   mBase.Initialize("mBase")
   Parent.AddView(mBase, mLeft, mTop, mWidth, mHeight)

   mIcon = Icon
   mText = Text
   mTextColor = TextColor

   InitClass
End Sub
```

B4J `Parent.AddView` is relaced by `Parent.AddNode`.

```
Parent.AddNode(mBase, mLeft, mTop, mWidth, mHeight)
```

5.9 InitClass routine

Here we initialize the common part independent if the CustomButton is added in the Designer or in the code.

B4A

```
Private Sub InitClass
   'calculate the dimensions of the internal Labels
   Private lblLeft, lblWidth  As Int
   lblWidth = 2 * mHeight / 3    'icon Label width and height = 2/3 of button height
   lblLeft = (mWidth - lblWidth) / 2

   'initialize and add the icon Label
   mIconLabel.Initialize("")
   mIconTextSize = mHeight / 2 / GetDeviceLayoutValues.Scale            'B4A, B4i
   mIconLabel.Typeface = mIconTypeface                                  'B4A
   mIconLabel.TextSize = mIconTextSize                                  'B4A, B4J
   mIconLabel.Gravity = Gravity.CENTER                                  'B4A
   mIconLabel.TextColor = mTextColor
   mBase.AddView(mIconLabel, lblLeft, 0, lblWidth, lblWidth)            'B4A, B4i
   mIconLabel.Text = mIcon

   'initialize and add the text Label
   mLabel.Initialize("")
   mTextSize = lblWidth / 3 / GetDeviceLayoutValues.Scale               'B4A, B4i
   mLabel.TextSize = mTextSize                                          'B4A, B4J
   mLabel.TextColor = mTextColor
   mLabel.Gravity = Bit.Or(Gravity.CENTER_HORIZONTAL, Gravity.TOP)      'B4A
   mBase.AddView(mLabel, 0, 2 * mHeight / 3, mWidth, mHeight / 3)       'B4A, B4i
   mLabel.Text = mText
End Sub
```

`'B4A` means that this line is only for B4A and different from B4i and B4J.
`'B4A, B4i` means that this line is the same for B4A and B4i, but is different in B4J.

B4i

```
Private Sub InitClass
   'calculate the dimensions of the internal Labels
   Private lblLeft, lblWidth  As Int
   lblWidth = 2 * mHeight / 3    'icon Label width and height = 2/3 of button height
   lblLeft = (mWidth - lblWidth) / 2

   'initialize and add the icon Label
   mIconLabel.Initialize("")
   mIconTextSize = mHeight / 2 / GetDeviceLayoutValues.Scale            'B4i, B4A
   mIconFont = Font.CreateNew2(mIconFont.Name, mIconTextSize)           'B4i
   mIconLabel.Font = mIconFont                                          'B4i, B4J
   mIconLabel.TextAlignment = mIconLabel.ALIGNMENT_CENTER               'B4i
   mIconLabel.TextColor = mTextColor
   mBase.AddView(mIconLabel, lblLeft, 0, lblWidth, lblWidth)            'B4i, B4A
   mIconLabel.Text = mIcon

   'initialize and add the text Label
   mLabel.Initialize("")
   mTextSize = lblWidth / 3 / GetDeviceLayoutValues.Scale               'B4i, B4A
   mLabel.Font = Font.CreateNew(mTextSize)                              'B4i
   mLabel.TextColor = mTextColor
   mLabel.TextAlignment = mIconLabel.ALIGNMENT_CENTER                   'B4i
   mBase.AddView(mLabel, 0, 2 * mHeight / 3, mWidth, mHeight / 3)       'B4i, B4A
   mLabel.Text = mText
End Sub
```

B4J

```
Private Sub InitClass
   'calculate the dimensions of the internal Labels
   Private lblLeft, lblWidth  As Int
   lblWidth = 2 * mHeight / 3    'icon Label width and height = 2/3 of button height
   lblLeft = (mWidth - lblWidth) / 2

   'initialize and add the icon Label
   mIconLabel.Initialize("")
   mIconTextSize = mHeight / 2                                          'B4J
   mIconLabel.Font = mIconFont                                          'B4J, B4i
   mIconLabel.TextSize = mIconTextSize                                  'B4J, B4A
   mIconLabel.Alignment = "CENTER"                                      'B4J
   mIconLabel.TextColor = mTextColor
   mBase.AddNode(mIconLabel, lblLeft, 0, lblWidth, lblWidth)            'B4J
   mIconLabel.Text = mIcon

   'initialize and add the text Label
   mLabel.Initialize("")
   mTextSize = lblWidth / 3                                             'B4J
   mLabel.TextSize = mTextSize                                          'B4J, B4A
   mLabel.TextColor = mTextColor
   mLabel.Alignment = "TOP_CENTER"                                      'B4J
   mBase.AddNode(mLabel, 0, 2 * mHeight / 3, mWidth, mHeight / 3)       'B4J
   mLabel.Text = mText
End Sub
```

5.10 Click / LongClick event routines

The two event routines.

B4A / B4i

```
Private Sub mBase_Click
   If SubExists(mCallBack, mEventName & "_Click") = True Then
      CallSub(mCallBack,  mEventName & "_Click")
   End If
End Sub

Private Sub mBase_LongClick
   If SubExists(mCallBack, mEventName & "_LongClick") = True Then
      CallSub(mCallBack,  mEventName & "_LongClick")
   End If
End Sub
```

B4J
Very different, because the LongClick event doesn't exist in B4J.

```
Private Sub mBase_MousePressed (EventData As MouseEvent)
   mClickTime = DateTime.Now
End Sub

Private Sub mBase_MouseReleased (EventData As MouseEvent)
   If DateTime.Now - mClickTime < 500 Then
      If SubExists(mCallBack, mEventName & "_Click") = True Then
         CallSub(mCallBack,  mEventName & "_Click")
      End If
   Else
      If SubExists(mCallBack, mEventName & "_LongClick") = True Then
         CallSub(mCallBack,  mEventName & "_LongClick")
      End If
   End If
End Sub
```

In **mBase_MousePressed**, we memorize the time when the mouse is pressed.

In **mBase_MousePressed**, we check the time elapsed between press and release.
If the time is less than 500 milli-seconds, then we admid a Click and if time is longer we admit a LongClick event.

5.11 Property routines

Below the routine to set the IconTypeFace / IconFont property.

B4A

```
'set the icon typeface
'must be FontAwsome or Material Icons
Public Sub setIconTypeface(IconTypeface As Typeface)
   mIconTypeface = IconTypeface
End Sub
```

B4i / B4J

```
'set the icon typeface
'must be FontAwsome or Material Icons
Public Sub setIconFont(IconFont As Font)
   mIconFont = IconFont
End Sub
```

And the Tag property.

B4A / B4i / B4J

```
'get or set the Tag property
Public Sub setTag(Tag As Object)
   mTag = Tag
   mBase.Tag = Tag
End Sub

Public Sub getTag As Object
   Return mTag
End Sub
```

I haven’t added other properties to not overload the code.

5.12 Main code

5.12.1 Globals

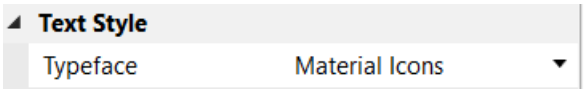
Only two variables, in addition to the default declarations in B4i and B4J.

B4A / B4i B4J

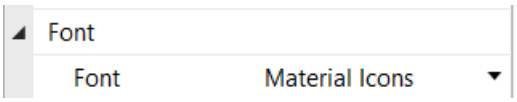
```
Sub Globals
   Private cbtTest10 As CustomButton
   Private lblDummy As Label
End Sub
```

We need a dummy invisible Label to get the Material Icons TypeFace / Font for the icon Label when the CustomButton is added in the code.

B4A



B4i / B4J



5.12.2 Program start

B4A

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

   cbtTest10.Initialize(Me, "cbtTest")
   cbtTest10.IconTypeface = lblDummy.Typeface
   cbtTest10.AddToParent(Activity, 20dip, 200dip, 60dip, 60dip, Colors.RGB(0, 0, 139),
Chr(0xE149), "Test 10")
   cbtTest10.Tag = 10
End Sub
```

B4i

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

   cbtTest10.Initialize(Me, "cbtTest")
   cbtTest10.IconFont = lblDummy.Font
   cbtTest10.AddToParent(Page1.RootPanel, 30, 100, 60, 60, Colors.RGB(0, 0, 139),
Chr(0xE05C), "Test 10")
   cbtTest10.Tag = 10
End Sub
```

B4J

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

   MainForm.Title = "jClsCustomButton"

   cbtTest10.Initialize(Me, "cbtTest")
   cbtTest10.IconFont = lblDummy.Font
   cbtTest10.AddToParent(MainForm.RootPane, 100, 100, 60, 60, fx.Colors.RGB(0, 0, 139),
Chr(0xE05C), "Test 10")
   cbtTest10.Tag = 10
End Sub
```

Besides the default operating system methods, the CustomButton declaration is also a bit different.

- Parent object: **B4A** Activity (Panel), **B4i** Page1.RootPanel, **B4J** MainForm.RootPane.
- Font type:
  B4A TypeFace
  `cbtTest10.IconTypeface = lblDummy.Typeface`
  B4i / **B4J** Font
  `cbtTest10.IconFont = lblDummy.Font`

5.13 Click event routine

The Click event routine is the same for all three operating systems:

```
Private Sub cbtTest_Click
   Private cbt As CustomButton
   Private Index As Int

   cbt = Sender
   Index = cbt.Tag

   Select Index
   Case 1
       Log("cbtTest1_Click")
   Case 2
       Log("cbtTest2_Click")
   Case 10
       Log("cbtTest10_Click")
   Case Else
       Log("cbtTest" & Index & "_Click")
   End Select
End Sub
```

I set the same event name for all CustomButtons and use the Tag property of the Sender object to know which button raised the event.

The LongClick event routine is almost the same, `LongClick` replaces `Click`.

5.14 Compile to Library

We add the library declarations on top of the code in the Main module.

B4A

```
#LibraryName: CustomButton
#LibraryAuthor: Klaus CHRISTL
#LibraryVersion: 1.0
```

B4i

```
#LibraryName: iCustomButton
#LibraryAuthor: Klaus CHRISTL
#LibraryVersion: 1.0
```

B4J

```
#LibraryName: jCustomButton
#LibraryAuthor: Klaus CHRISTL
#LibraryVersion: 1.0
```

And we compile the CustomButton module to a Library.

The Library files are automatically copied to the AdditionalLibraries folder.

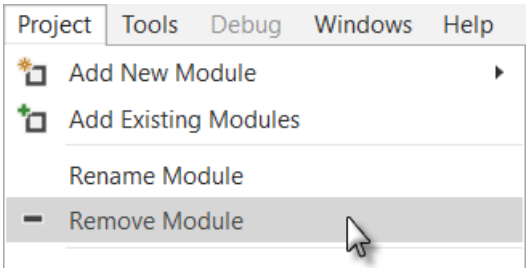
If you use the hosted compiler for B4i, you must ask Erel for permission to be able to compile a library.

5.15 Use the library in a program

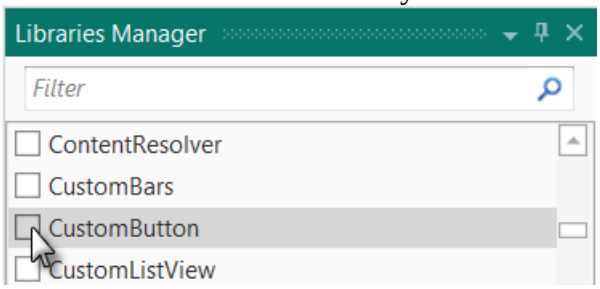
Copy the projects to new folders where you replace *Cls* by *Lib*.
The source codes are in the *LibCustomButton* folders

Then:

- Load the projects in the IDE.
- Rename the package name.
- Rename the `#ApplicationLabel: LblCustomButton` (B4A and B4i only)
- Remove the three lines:
```
#LibraryName: CustomButton
#LibraryAuthor: Klaus CHRISTL
#LibraryVersion: 1.0
```
- Remove the CustomButton class module.
Select the CustomButton module and remove it.



- Select the CustomButton library in the Labaraies Manager Tab.



- Run the program.

The rest of the code in the Main module remains the same.
The layout file remains the same.

6 LimitBar

Another concrete example, a LimitBar.

The LimitBar looks like this, images from the B4A project:

Two cursors allow to define two limits between 0 and a max value.

In the demo program, we add two labels, one on each side, to display the two limit values these are not part of the custom view.

There are two projects for each operating system:

- ClsLimitBarDemo, project with the custom view class module.
- LibLimitBarDemo, project with the custom view as a library (class module compiled to a library).

It supports adding a LimiBar in the Designer or in the code.
In the demo projects two LimitBars are added, one in the Designer and one in the code.

The source codes are in the \ClsLimitBar and \LibLimitBar folders, one folder for each operating system.

We use for:
B4A / B4i
Two panels:

- `ltbPanelBack` the background with the background color and the red 'background' line.
  
- `ltbPanelFront` the foreground, transparent with the 'foreground' line and the two cursors.
  

and two canvases:

- `cvsPanelBack` to draw the background line onto `ltbPanelBack`.
- `cvsPanelFront` to draw the foreground line and the cursors onto `ltbPanelFront`.

B4J
One pane as the custom view holder `ltbPanelBack` and two canvases `cvsBack` and `cvsFront`.
The difference in B4J is that canvases are objects on their own, not linked to anoter object like in B4A and B4i.

The LimitBar customView raises one event:

```
ValuesChanged(LimitLeft As Int, LimitRight As Int)
```

In the next chapters, the code is explained from top to down.

6.1 Event declaration

On top of the code we declare the event:

```
'Events declaration
#Event: ValuesChanged(LimitLeft As Int, LimitRight As Int)
#RaisesSynchronousEvents: ValuesChanged
```

We need this for the intellisense system when the class module is compiled to a library.

6.2 Custom properties for the Designer

The LimitBar has following custom properties:

- Max
  Sets or gets the max limit value when the curser is at the most right position.
  The default value is 100.
- LimitLeft
  Sets or gets the left limit value. The default value is 0.
- LimitRight
  Sets or gets the right limit value. The default value is 100.
- BackLineColor
  Sets or gets the back-line color. The default value is red (0xFFFF0000).
- FrontLineColor
  Sets or gets the front-line color. The default value is light blue (0x FF33B5E5).

Tu support setting these properties in the Designer we must declare them:

```
'Designer property declarations
#DesignerProperty: Key: Max, DisplayName: Max, FieldType: Int, DefaultValue: 100,
Description: Sets the max value.
#DesignerProperty: Key: LimitLeft, DisplayName: Left limit, FieldType: Int,
DefaultValue: 10, Description: Sets the left limit value.
#DesignerProperty: Key: LimitRight, DisplayName: Right limit, FieldType: Int,
DefaultValue: 100, Description: Sets the right limit value.
#DesignerProperty: Key: Radius, DisplayName: Radius, FieldType: Int, DefaultValue: 5,
Description: Sets the corner radius.
#DesignerProperty: Key: BackgroundColor, DisplayName: BackgroundColor, FieldType:
Color, DefaultValue: 0xFF0000FF, Description: Sets the background color.
#DesignerProperty: Key: BackLineColor, DisplayName: BackLineColor, FieldType: Color,
DefaultValue: 0xFFFF0000, Description: Sets the back line color.
#DesignerProperty: Key: FrontLineColor, DisplayName: FrontLineColor, FieldType: Color,
DefaultValue: 0xFF33B5E5, Description: Sets the front line color.
```

We will add also code to set or get these properties in the code.

6.3 Class help header

Class header help text, just before `Sub Class_Globals`.

```
'LimitBar CustomView class.
'This CustomView allows the user to set two limits with two cursors.
'The Min value is 0 and the Max value is 100.
'The Max value can be changed by the programmer.
Sub Class_Globals
```

6.4 Global variables

In `Sub Class_Globals` we declare the objects and variables.

B4A / B4i the only difference is `cdwBackground As ColorDrawable` used in B4A.

```
Sub Class_Globals
   Private mCallback As Object          ' calling module
   Private mEventName As String         ' event name
   Private mWidth, mHeight, mRadius As Int

   Private ltbPanelBack As Panel        ' the background panel
   Private ltbPanelFront As Panel       ' the foreground panel
   Private cvsPanelBack As Canvas       ' the background canvas
   Private cvsPanelFront As Canvas      ' the foreground canvas
   Private rectPanelFront As Rect       ' a rectangle for the foreground canvas

   Private mBackgroundColor As Int      ' color for the background
   Private cdwBackground As ColorDrawable     ' background drawable, only for B4A
   Private mBackLineColor As Int        ' color for the background line
   Private mFrontLineColor As Int       ' color for the foreground line
   Private mMargin As Int               ' left and right margins for the line
   Private x0, y0, x1, y1, y2 As Int    ' backline and cursor coordinates
   Private mMaxValue As Int             ' value of the Max property
   Private mScale As Double             ' scale between position value and pixels
   Private mLimit(2) As Int             ' values of the limits
   Private PositionPixels(2) As Int     ' left and right positions in pixels
   Private Paths(2) As Path             ' two paths for the cursor shape
   Private PosIndex As Int              ' current index of the cursor position
                                        ' 0 = left  1 = right  cursor
End Sub
```

B4J

```
Sub Class_Globals
   Private fx As JFX                     ' B4J specific
   Private mCallback As Object           ' calling module
   Private mEventName As String          ' event name
   Private mWidth, mHeight, mRadius As Double

   Private ltbPanelBack As Pane          ' the background pane
   Private cvsBack As Canvas             ' the background canvas
   Private cvsFront As Canvas            ' the foreground canvas

   Private mBackgroundColor As Paint     ' color for the background
   Private mBackLineColor As Paint       ' color for the background line
   Private mFrontLineColor As Paint      ' color for the foreground line
   Private mMargin As Double             ' left and right margins for the line
   Private x0, y0, x1, y1, y2 As Double  ' backline and cursor coordinates
   Private mMaxValue As Int              ' value of the Max property
   Private mScale As Double              ' scale between position value and pixels
   Private mLimit(2) As Int              ' value of the limits
   Private PositionPixels(2) As Double   ' left and right positions in pixels
   Private PosIndex As Int
                                         ' 0 = left   1 = right
   Private Graph, jCanvasFront As JavaObject   ' B4J specific used to draw the cursors
End Sub
```

There are some differences between B4J and B4A / B4i. B4J needs of the jFX library.
Some differences for the canvases. No foreground panel but two JavaObjects.

6.5 Initialize routine

Then we need the routine to initialize the LimitBar, the code is self explanatory.
This routine is automatically called if you add the LimitBar in the Designer.
If you add the LimitBar in the code, you must call this routine first.
You should not modify the signature of this routine

B4A / B4i / B4J

```
'Initializes the object.
'Callback = name of the calling module
'EventName = event name
'Example if added in the code:
'<Code>ltbTest.Initialize(Me, "ltbTest")'</Code>
Public Sub Initialize(Callback As Object, EventName As String)
   mCallback = Callback
   mEventName = EventName

   ' initialize default values
   mBackgroundColor = Colors.Blue
   mBackLineColor = Colors.Black
   mFrontLineColor = Colors.RGB(51, 181, 229)
   mMargin = 15
   mMaxValue = 100
   mLimit(0) = 0
   mLimit(1) = mMaxValue
End Sub
```

6.6 DesignerCreateView routine

Then we have the DesignerCreateView routine.
This routine is called automatically after Initialize when the LimitBar is added in the Designer.
It is NOT used when you add the LimitBar in the code.

B4A / B4i

```
Public Sub DesignerCreateView(Base As Panel, Lbl As Label, Props As Map)
   ' we use the Base panel as the background panel
   ltbPanelBack = Base

   ' we memorize the Base Width and Height properties
   mWidth = Base.Width
   mHeight = Base.Height

   ' we memorize the custom properties
   mMaxValue = Props.Get("Max")
   mLimit(0) = Props.Get("LimitLeft")
   mLimit(0) = Max(0, mLimit(0))       ' we check the min value, not less than 0

   mLimit(1) = Props.Get("LimitRight")
   mLimit(1) = Min(mMaxValue, mLimit(1)) ' we check the max value, not higher than Max

   mRadius = Props.Get("Radius")
   mBackgroundColor = Props.Get("BackgroundColor")
   mBackLineColor = Props.Get("BackLineColor")
   mFrontLineColor = Props.Get("FrontLineColor")

   InitClass ' initializes the common parts for Designer and code
End Sub
```

We use the Base Panel with the name `ltbPanelBack`, and get the custom properties from the Props Map object.

As the LimitBar custom view can also be added in the code we initialize the rest in the `InitClass` routine.

B4J at end of the routine with B4i specific code.

```
   mBackLineColor = Props.Get("BackLineColor")
   mFrontLineColor = Props.Get("FrontLineColor")
End Sub
```

The `InitClass` routine is moved to a B4J specific routine **`Base_Resize`**:

Note that the signature of the routine is different:

```
Public Sub DesignerCreateView(Base As Panel, Lbl As Label, Props As Map)
```

`Base As Pane` instead of `Base As Panel`.

6.7 Base_Resize routine B4i / B4J only

In B4i and B4J there is a specific routine `Private Sub Base_Resize`.
This routine is executed every time a resize is operated.

B4i

```
Private Sub Base_Resize (Width As Double, Height As Double)
   setWidth(Width)
End Sub
```

B4J

```
Private Sub Base_Resize (Width As Double, Height As Double)
   mWidth = ltbPanelBack.Width
   mHeight = ltbPanelBack.Height

   If cvsBack.IsInitialized = False Then
      InitClass      ' initializes the common parts for Designer and code
   End If
   setWidth(Width)
End Sub
```

In B4J the width and height of the Base pane is known only in the **`Base_Resize`** routine.
This routine is called directly after **`DesignerCreateView`** when the LimitBar is added in the Designer.
It is not called when the LimitBar is added in the code.

6.8 AddToParent routine

The AddToParent routine.
This routine must be called when you add the LimitBar in the code.
It is not used when the LimitBar is added in the Designer.

B4A / B4i / B4J

```
'Adds the LimitBar to the Parent object
'Parent = parent view, the Activity or a Panel
'Left, Right, Width, Height = position and dimensions properties of the LimitBar
'Color = background color of the LimitBar
'Radius = corner radius of the LimitBar
Public Sub AddToParent(Parent As Activity, Left As Int, Top As Int, Width As Int,
Height As Int, BackgroundColor As Int, Radius As Int)
   mWidth = Width
   mHeight = Max(Height, 30dip)              ' limits the height to min 30dip
   mRadius = Min(Radius, Height / 2)         ' limits the max radius to half the height
   mBackgroundColor = BackgroundColor

   ' initialize the background panel ltbPanelBack and add it onto the parent view
   ltbPanelBack.Initialize("")
   Parent.AddView(ltbPanelBack, Left, Top, Width, Height)

   InitClass ' initializes the common parts for Designer and code
End Sub
```

We memorize several properties, initialize `ltbPanelBack` and add it onto the parent view and set its background and call `InitClass`.

Example:

```
   'adds a second LimitBar in the code
   ltbTest1.Initialize(Me, "ltbTest1")
   ltbTest1.FrontLineColor = Colors.Blue
   ltbTest1.AddToParent(Activity, 30dip, 100dip, 200dip, 30dip, Colors.Red, 5dip)
```

6.9 InitClass routine

In this routine, we initialize the common code parts independent if the LimitBar is added in the Designer or in the code.
This routine is called either from the **`DesignerCreateView`** when the LimitBar is added in the Designer or from the **`AddToParent`** routine when the custom view is added in the code.

B4A

```
Private Sub InitClass
   InitCursors

   ' initialize and set the ColorDrawable for the background panel
   cdwBackground.Initialize(mBackgroundColor, mRadius)
   ltbPanelBack.Background = cdwBackground

   ' initialize the foreground panel and add it onto the background panel
   ltbPanelFront.Initialize("ltbPanelFront")
   ltbPanelBack.AddView(ltbPanelFront, 0, 0, ltbPanelBack.Width, ltbPanelBack.Height)

   ' initialize the foreground panel rectangle used to erase ltbPanelFront
   rectPanelFront.Initialize(0, 0, ltbPanelFront.Width, ltbPanelFront.Height)
   ltbPanelFront.BringToFront

   ' initialize the background canvas and draw the background line
   cvsPanelBack.Initialize(ltbPanelBack)
   DrawBackGroundLine

   ' initialize the foreground canvas
   cvsPanelFront.Initialize(ltbPanelFront)

   ' set the limit max value, which calculates also the scale limit values <> pixels
   setMax(mMaxValue)
End Sub
```

The code is self explanatory.

B4i The background color setting is different from B4A.

```
Private Sub InitClass
   InitCursors

   ' set the background color and the radius for the background panel
   ltbPanelBack.Color = mBackgroundColor
   ltbPanelBack.SetBorder(0, mBackgroundColor, mRadius)

   ' initialize the foreground panel and add it onto the background panel
   ltbPanelFront.Initialize("ltbPanelFront")
   ltbPanelBack.AddView(ltbPanelFront, 0, 0, ltbPanelBack.Width, ltbPanelBack.Height)

   ' initialize the foreground panel rectangle used to erase ltbPanelFront
   rectPanelFront.Initialize(0, 0, ltbPanelFront.Width, ltbPanelFront.Height)
   ltbPanelFront.BringToFront

   ' initialize the background canvas and draw the background line
   cvsPanelBack.Initialize(ltbPanelBack)
   DrawBackGroundLine

   ' initialize the foreground canvas
   cvsPanelFront.Initialize(ltbPanelFront)

   setMax(mMaxValue)
End Sub
```

B4J the code is somewhat more specific.

```
Private Sub InitClass
   InitCursors

   ' set the background color and the radius for the background panel
   CSSUtils.SetBackgroundColor(ltbPanelBack, mBackgroundColor)
   CSSUtils.SetBorder(ltbPanelBack, 0, mBackgroundColor, mRadius)

   ' initialize the background canvas and draw the background line
   cvsBack.Initialize("cvsBack")
   ltbPanelBack.AddNode(cvsBack, 0, 0, mWidth, mHeight)
   DrawBackGroundLine

   ' initialize the foreground canvas
   cvsFront.Initialize("cvsFront")
   ltbPanelBack.AddNode(cvsFront, 0, 0, mWidth, mHeight)

   ' set a JavaObject for CanvasFront, needed for the drawing of the cursors
   jCanvasFront = cvsFront
   Graph = jCanvasFront.RunMethod("getGraphicsContext2D", Null)

   ' set the limit max value, which calculates also the scale limit values <> pixels
   setMax(mMaxValue)
End Sub
```

To set the background color and corner radius we need the CSSUtils library.
We initialize the back and front canvases and add them to the base pane.
As there is no direct Path object in B4J to draw the cursors we use a JavaObject `Graph` for this and initialize it here.

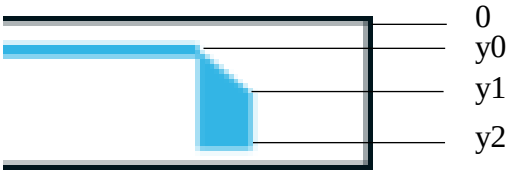
6.10 InitCursors routine

In this routine, we initialize the variables used for the background line and the cursors drawing.

B4A / B4i / B4J

```
Private Sub InitCursors
   x0 = mMargin
   x1 = mWidth - mMargin
   mScale = (x1 - x0) / mMaxValue
   PositionPixels(0) = mLimit(0) * mScale + x0
   PositionPixels(1) = mLimit(1) * mScale + x0

   y0 = 0.2 * mHeight
   y1 = y0 + 8dip + 0.05 * mHeight
   y2 = 0.9 * mHeight
End Sub
```



6.11 Draw the background line

We need to draw the background color and background line from several places in the code so we use a routine.

B4A

```
Private Sub DrawBackGround
   ltbPanelBack.Background = cdwBackground
   cvsPanelBack.Initialize(ltbPanelBack)
   cvsPanelBack.DrawLine(x0, y0, x1, y0, mBackLineColor, 2dip)
   ltbPanelBack.Invalidate
End Sub
```

We set the background of `ltbPanelBack`.
Reinitialize the background canvas, needed when the dimensions of `ltbPanelBack` change.
Draw the background line.
Force `ltbPanelBack` to redraw.
`ltbPanelBack.Invalidate` is equivalent to `cvsPanelBack.Refresh` in B4i.

B4i

```
Private Sub DrawBackGround
   cvsPanelBack.Initialize(ltbPanelBack)
   cvsPanelBack.DrawRectRounded(rectPanelFront, mBackgroundColor, True, 1, mRadius)
   cvsPanelBack.DrawLine(x0, y0, x1, y0, mBackLineColor, 2dip)
   cvsPanelBack.Refresh
End Sub
```

Reinitialize the background canvas, needed when the dimensions of `ltbPanelBack` change.
Draw the background onto the canvas.
Draw the background line.
Force `ltbPanelBack` to redraw.
`cvsPanelBack.Refresh` is equivalent to `ltbPanelBack.Invalidate` in B4A.

B4J

```
Private Sub DrawBackGround
   cvsBack.ClearRect(0, 0, mWidth, mHeight)
   cvsBack.DrawLine(x0, y0, x1, y0, mBackLineColor, 2)
End Sub
```

The background color is on the base pane.
So, we set the background canvas to transparent.
Draw the background line.
No need for `ltbPanelBack.Invalidate` nor `cvsPanelBack.Refresh` to redraw the canvas.

6.12 DrawCursors routine

The drawing routine for the cursors and the foreground line:
We use two Path objects to draw the cursor shapes.

B4A / B4i

```
Private Sub DrawCursors
   ' draw a transparent rectangle to erase the foreground panel
   cvsPanelFront.DrawRect(rectPanelFront, Colors.Transparent, True, 1)

   ' define the left cursor path according to its current position
   Paths(0).Initialize(PositionPixels(0), y0)
   Paths(0).LineTo(PositionPixels(0), y2)
   Paths(0).LineTo(PositionPixels(0) - 12dip, y2)
   Paths(0).LineTo(PositionPixels(0) - 12dip, y1)
   Paths(0).LineTo(PositionPixels(0), y0)

   ' define the right cursor path according to its current position
   Paths(1).Initialize(PositionPixels(1), y0)
   Paths(1).LineTo(PositionPixels(1), y2)
   Paths(1).LineTo(PositionPixels(1) + 12dip, y2)
   Paths(1).LineTo(PositionPixels(1) + 12dip, y1)
   Paths(1).LineTo(PositionPixels(1), y0)

   ' draw the two cursors and the front line
   cvsPanelFront.DrawPath(Paths(0), mFrontLineColor, True, 1)
   cvsPanelFront.DrawPath(Paths(1), mFrontLineColor, True, 1)
   cvsPanelFront.DrawLine(PositionPixels(0), y0, PositionPixels(1), y0, mFrontLineColor,
3dip)

   ltbPanelFront.Invalidate
End Sub
```

We:

- Erase the whole panel by drawing a transparent rectangle.
- Define both cursors according to the current position.
  The cursor shapes are defined with two Paths.
- Draw the cursors.
- Draw the foreground line.

The only difference between B4A and B4i is the redraw method at the end of the routine:
B4A `ltbPanelFront.Invalidate`
B4i `cvsPanelFront.Refresh`

B4J the routine is different because the Path object doesn't exist like in B4A / B4i.
The path methods are different and not directly exposed to B4J, but we can use those with the JavaObject `Graph` which is defined in the `InitClass` routine.

```
Private Sub DrawCursors
   ' draw a transparent rectangle to erase the foreground panel
   cvsFront.ClearRect(0, 0, cvsFront.Width, cvsFront.Height)

   ' draw the left cursor
   Graph.RunMethod("setFill", Array As Object(mFrontLineColor))
   Graph.RunMethod("beginPath", Null)
   Graph.RunMethod("moveTo", Array As Object(PositionPixels(0), y0))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(0), y2))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(0) - 12,  y2))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(0) - 12, y1))
   Graph.RunMethod("fill", Null)

   ' draw the right cursor
   Graph.RunMethod("setFill", Array As Object(mFrontLineColor))
   Graph.RunMethod("beginPath", Null)
   Graph.RunMethod("moveTo", Array As Object(PositionPixels(1), y0))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(1), y2))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(1) + 12dip,  y2))
   Graph.RunMethod("lineTo", Array As Object(PositionPixels(1) + 12dip, y1))
   Graph.RunMethod("fill", Null)

   ' draw the front line
   cvsFront.DrawLine(PositionPixels(0), y0, PositionPixels(1), y0, mFrontLineColor,
3dip)
End Sub
```

6.13 Cursor moving

To detect cursor moves we use the touch event of the foreground panel in B4A / B4i and the mouse events of the front canvas in B4J:

B4A / B4i same routine

```
Private Sub ltbPanelFront_Touch (Action As Int, X As Float, Y As Float)
   ' check if the cursor is outsides the limits
   X = Max(x0, X)
   X = Min(x1, X)

   ' select the Action type
   Select Action
   Case 0     'DOWN
      If X < Abs(PositionPixels(0) + PositionPixels(1)) / 2 Then
         ' if X is closer to the left cursor we choose it
         PosIndex = 0
      Else
         ' otherwise we choose the right cursor
         PosIndex = 1
      End If
      mLimit(PosIndex) = Floor((X - x0) / mScale + .5)
      PositionPixels(PosIndex) = X
      DrawCursors
   Case 2     ' MOVE
      mLimit(PosIndex) = Floor((X - x0) / mScale + .5)
      PositionPixels(PosIndex) = X
      DrawCursors
   Case 1     ' UP
      ' when Action is UP (mouse released) check if mLimit(0) > mLimit(1)
      ' if yes we invert the limit values and redraw the cursors
      If mLimit(0) > mLimit(1) Then
         Private val As Int
         val = mLimit(0)
         mLimit(0) = mLimit(1)
         mLimit(1) = val
         PositionPixels(0) = mLimit(0) * mScale + x0
         PositionPixels(1) = mLimit(1) * mScale + x0
         DrawCursors
      End If
   End Select

   ' if a callback routine exists in the calling module we call it
   If SubExists(mCallback, mEventName & "_ValuesChanged") Then
      CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
   End If
End Sub
```

B4J very different because the Touch event doesn't exist in B4J and we must use the mouse event routines. But, as we have a routine handling the Touch event in B4A and B4i we use the same routine and call it from the mouse event routines.

Mouse event routines:

- MouseClicked equivalent to the Click event.
- MouseDragged equivalent to the Touch event MOVE with a button pressed.
- MouseMoved no equivalent in B4A / B4i MOVE but no button pressed.
- MousePressed equivalent to the Touch event DOWN.
- MouseReleased equivalent to the Touch event UP.

These routines return a MouseEvent object with following properties.

- ClickCount
- Consume
- MiddleButtonDown
- MiddleButtonPressed
- PrimaryButtonDown
- PrimaryButtonPressed
- SecondaryButtonDown
- SecondaryButtonPressed

We use the MousePressed (DOWN), MouseDragged (MOVE) and MouseReleased (UP) events.

```
Private Sub cvsFront_MousePressed (EventData As MouseEvent)
   cvsFront_Touch (0, EventData.X)
End Sub

Private Sub cvsFront_MouseDragged (EventData As MouseEvent)
   cvsFront_Touch (2, EventData.X)
End Sub

Private Sub cvsFront_MouseReleased (EventData As MouseEvent)
   cvsFront_Touch (1, EventData.X)
End Sub
```

And another routine wich is internally exactly the same as the Touch event routine in B4A / B4i.

The only difference is its signature. `Private Sub cvsFront_Touch (Action As Int, X As Double)`

Instead of `Private Sub ltbPanelFront_Touch (Action As Int, X As Float, Y As Float)`.

We transmit the 'Action' parameter and only the X coordinate because we don't need the Y coordinate.

6.14 Properties

Finally, we add a few properties:
To add properties, see more details in Add properties.

The Max property:
```
'gets or sets the max value
Sub setMax(cMax As Int)
   MaxVal = cMax
   Scale = (x1 - x0) / MaxVal
End Sub

Sub getMax As Int
   Return MaxVal
End Sub
```

The LimitLeft property:
```
'gets or sets the left limit
Sub setLimitLeft(Pos As Int)
   ' if Pos is lower than 0 set cLimitLeft to 0
   cLimitLeft = Max(0, Pos)
   PosIndex = 0
   DrawPos(x0 + cLimitLeft * Scale, 0)
End Sub

Sub getLimitLeft As Int
   Return cLimitLeft
End Sub
```

The LimitRight property:
```
'gets or sets the right limit
Sub setLimitRight(Pos As Int)
   ' if Pos is higher than MaxVal set cLimitRight to MaxVal
   cLimitRight = Min(MaxVal, Pos)
   PosIndex = 1
   DrawPos(x0 + cLimitRight * Scale, 0)
End Sub

Sub getLimitRight As Int
   Return cLimitRight
End Sub
```

The Visible property:
```
'gets or sets the Visible property
Sub setVisible(IsVisible As Boolean)
   ltbPanelBack.Visible = IsVisible
End Sub

Sub getVisible As Boolean
   Return ltbPanelBack.Visible
End Sub
```

I didn't add more properties to keep the example code 'simple'.
But other properties could easily be added.

6.15 Compile to a Library

In the Project Attributes Region in the Main module we add following new attributes:

Example, LimitBar projects.

B4A
```
#LibraryName: LimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```

B4i
```
#LibraryName: iLimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```

B4J
```
#LibraryName: jLimitBar
#LibraryAuthor: Klaus Christl
#LibraryVersion: 1.0
```

And we compile the Limitbar module to a Library.

The Library files are automatically copied to the AdditionalLibraries folder.

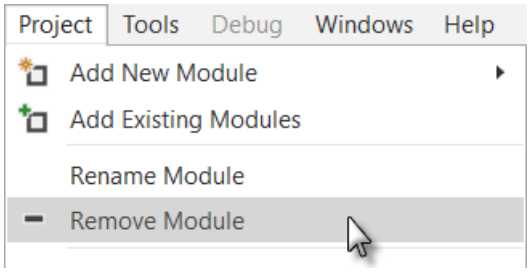
If you use the hosted compiler for B4i, you must ask Erel for permission to be able to compile a library.

6.15.1 Using the library in a program

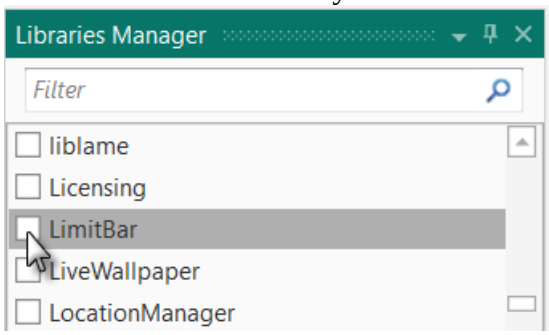
Copy the projects to new folders where you replace *Cls* by *Lib*.
The source codes are in the *LibLimitBar* folders.

Then:

- Load the projects in the IDE.
- Rename the package name.
- Rename the `#ApplicationLabel: LblLimitBar` (B4A and B4i only)
- Remove the three lines:
```
#LibraryName: LimitBar
#LibraryAuthor: Klaus CHRISTL
#LibraryVersion: 1.0
```
- Remove the LimitBar class module.
Select the LimitBar module and remove it.



- Select the LimitBar library in the Labaraies Manager Tab.



- Run the program.

The rest of the code in the Main module remains the same.
The layout file remains the same.