B4X Booklets

B4X CustomViews

Edition 2.5

Last update: 2024.01.05

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

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

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

Updated for:
B4A version 12.80
B4i version 8.50
B4J version 10.00

B4X Booklets:
B4X Getting Started
B4X B4X Language
B4X IDE Integrated Development Environment
B4X Visual Designer
B4X Help tools

B4X CustomViews
B4X Graphics
B4X XUI B4**X** **U**ser **I**nterface
B4X SQLite Database
B4X JavaObject NativeObject
B4XPages Cross-platform projects

B4R Example Projects

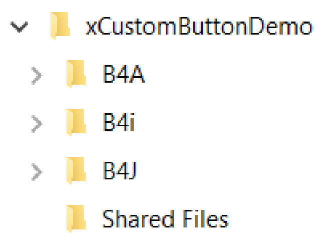
You can consult these booklets online in this link [B4X] Documentation Booklets.
Be aware that external links do not work in the online display.

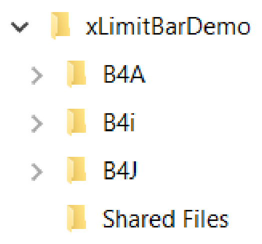
1 General information

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

It covers B4A, B4i, B4J and XUI.

All the source code and files needed (layouts, images etc) for the example projects in this guide are included in the SourceCode folder.
Each project is a B4XProject, with the structure below.





One folder for each platform, and a Shared Files folder.
This is the ‘standard’ structure for a B4XPages project.
For the projects in this booklet, the Shared Files folders are empty, no shared file.

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 us 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
```

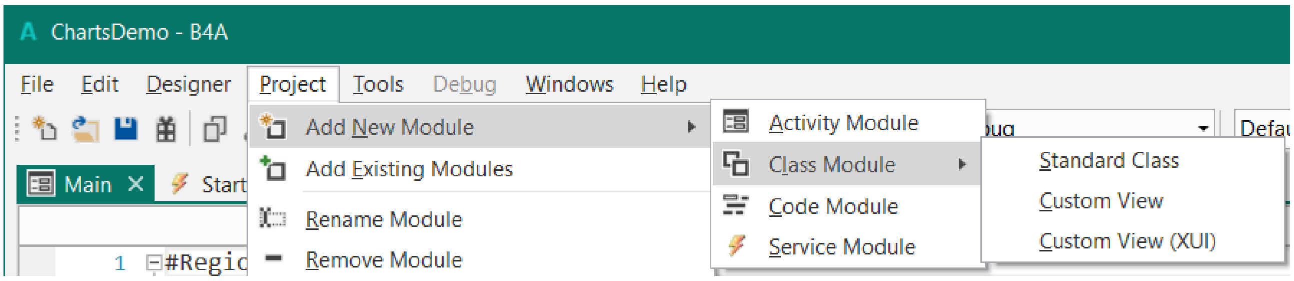
I will start by explaining the differences between classes, code modules and types.

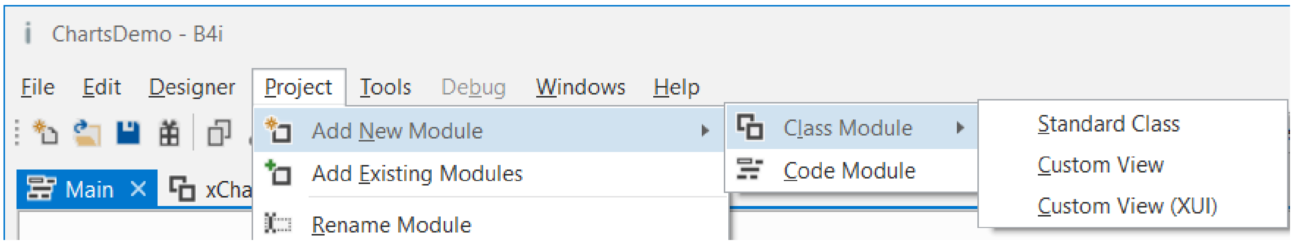
Similar to types, classes are templates. From this template, you can instantiate any number of objects.
The type fields are similar to the classes’ global variables. However, unlike types which only define the data structure, classes also define the behaviour. The behaviour is defined in the classes’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

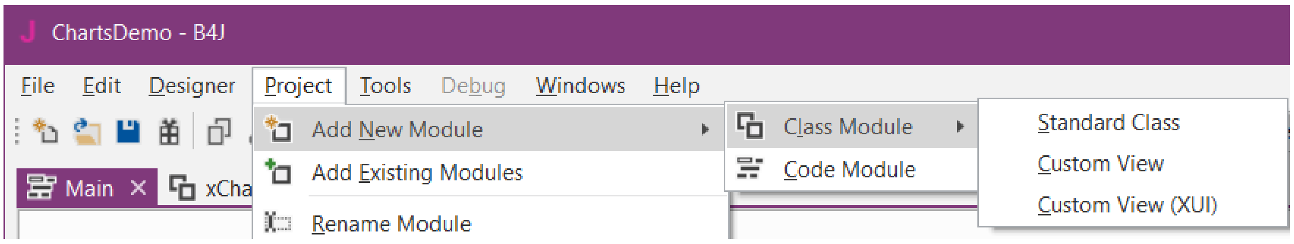
Unlike classes which are a template for objects, code modules are collections of subs. Another important difference between code modules and classes is that code modules always run in the context of the calling sub. The code module does not hold a reference to any context. For that reason, it is impossible 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

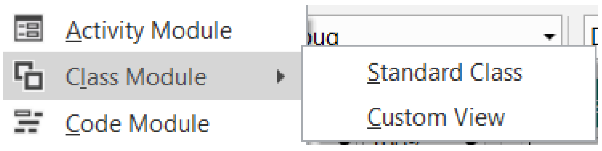
Adding a new or existing class module is done by choosing Project > Add New Module > Class module or Add Existing module.
Like other modules, classes are saved as files with *bas* extension.



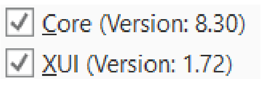




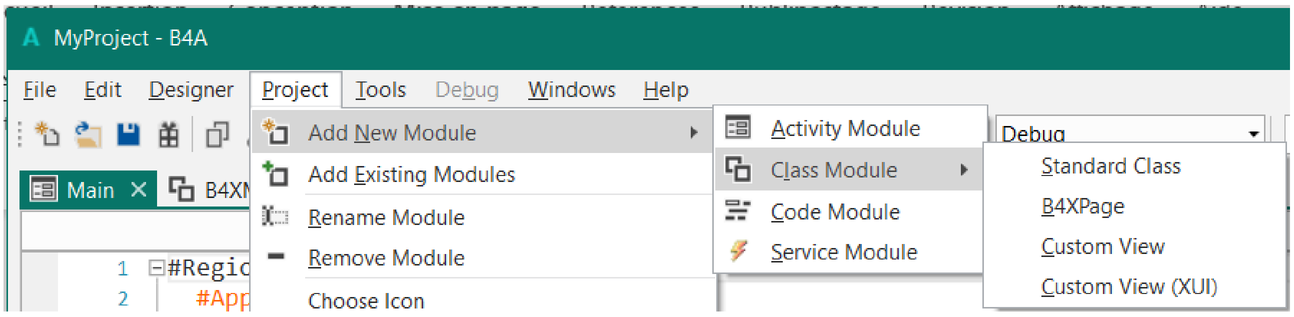
There are two class module types:
Standard Class
CustomView
CustomView (XUI)



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



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



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.
This is less important if you use B4XPages.

Android UI elements hold a reference to the parent activity. As the OS is allowed to kill background activities 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 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)
```

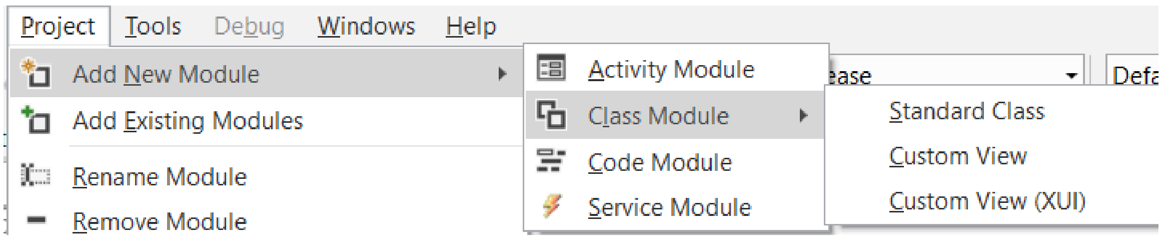
4 CustomViews

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

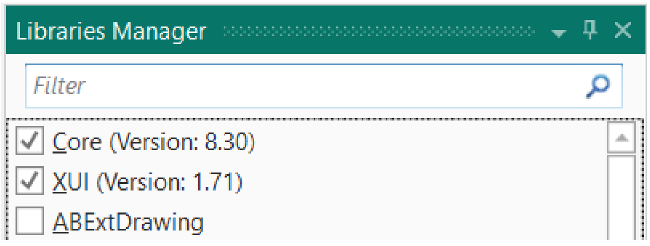
4.1 CustomView types

There are two CustomView types:

- CustomView
- CustomView (XUI)



To show the CustomView (XUI) option you must first add the XUI library.



The only differences between CustomView and CustomView (XUI) templates are the declarations.

- Standard

  ```
  Private mBase As Panel 'ignore

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

- XUI

  ```
  Private mBase As B4XView 'ignore

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

Advice: Use directly XUI and B4XViews even for a mono platform project.

4.2 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
   Public mBase As Panel
   Public Tag As Object
   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:

```
Public 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.

4.2.1 Event declarations

You should add Event declarations. If the event routine has parameters, these must also be declared.

`#Event: ExampleEvent (Value As Int)`	important for intellisense.
`#RaisesSynchronousEvents: ExampleEvent`	important for libraries.

4.2.2 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.

4.2.3 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
   Public mBase As B4XView
   Public Tag As Object
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.
Tag	The Tag contains the class object.

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`.

4.2.4 Initialization routine

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

You should not modify its signature.

```
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.

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
```

Or the XUI version, it has a third parameter *NumberOfArguments* for B4i, which has no effect in B4A nor in B4J but is cross-platform with the same code line.

```
' if a callback routine exists in the calling module we call it
If xui.SubExists(mCallback, mEventName & "_ValuesChanged", 2) Then
  CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
End If
```

4.2.5 Designer support routine DesignerCreateView

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.

- Standard

  B4A and B4i
  ```
  Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
    mBase = Base
  End Sub
  ```

  B4J
  ```
  Public Sub DesignerCreateView (Base As Pane, Lbl As Label, Props As Map)
    mBase = Base
  End Sub
  ```

- **XUI**
  ```
  Public Sub DesignerCreateView (Base As Object, Lbl As Label, Props As Map)
    mBase = Base
    Tag = mBase.Tag
    mBase.Tag = Me
  End Sub
  ```

`Base` Is the base Panel / Pane object defined in the Designer, it holds the Left, Top, Width, Height and Parent properties of the custom view. The Base object 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.

Default properties:
'activity' gets the parent view/node
```
mParent = Props.Get("activity")
```

`mBase`	is the base panel of the custom view.
Tag	holds the custom view object, can be used with GetView.
`mBase.Tag`	holds the Tag defined in the Designer.

Advice: Use directly XUI and B4XViews even for a mono platform project.

4.2.6 Accessing a CustomView from any container like xCustomListview

This code will not work:
```
Private B4XFloatTextField1 As B4XFloatTextField = CLV.GetPanel(x).GetView(y)
```

This code will work if the Tag is set as above.
```
Private B4XFloatTextField1 As B4XFloatTextField = CLV.GetPanel(x).GetView(y).Tag
```

4.2.7 Routine to get the base Panel

You can use this routine if you have not set mBase as Public and want to access the base panel / pane from other modules.

B4A / B4i

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

B4J

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

XUI

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

In the calling module:

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

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

```
Private pnlClass As B4XView
pnlClass = clsTest.GetBase
```

If you have declared mBase as Public, simply use:

```
pnlClass = clsTest.mBase
```

Advice: Use directly XUI and B4XViews even for a mono platform project.

4.3 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,`
XUI	a B4XView	`Public Sub AddToParent(Parent As Object,`

Example:
```
Public Sub AddToParent(Parent As B4XView, 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)
```

XUI Panel, with dip values (the dip values have no effect in B4J).

```
Private clsTest2 As ClsCustomView

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

4.4 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.
Usage: `xxx.Max = 100`

`Public Sub SetMax` is considered as a Public Subroutine.
Usage: `xxx.SetMax(100)`

4.5 Add Events

You can add events seen from outsides the class.

In the class you can add event routines like in any other interface module.
From the internal event routines, you can call external routines generating external events.

Example: TheTouch event of the xClsLimitBar project.

4.5.1 Code in the Class

Code in the class, Touch event of the front panel, which is a B4XView (Panel / Pane), irrelevant code has been removed, for simplification:

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

   ' select the Action type
   Select Action
      Case ltbPanelFront.TOUCH_ACTION_DOWN
         If xx < 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
      Case ltbPanelFront.TOUCH_ACTION_MOVE
         If xui.SubExists(mCallback, mEventName & "_ValuesChanged", 2) Then
            CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
         End If

      Case ltbPanelFront.TOUCH_ACTION_UP
         'call the ValuesChanged routine if it exists
         If xui.SubExists(mCallback, mEventName & "_ValuesChanged", 2) Then
            CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
         End If
   End Select
End Sub
```

We use the CallSub keyword to generate the ‘external’ event.
There are different routines depending on the number of parameters to transmit.

- **CallSub**(Component As Object, Sub As String)
- **CallSub2**(Component As Object, Sub As String, Argument As Object)
- **CallSub3**(Component As Object, Sub As String, Argument1 As Object, Argument2 As Object)

Component = the calling object.
In the example: mCallBack which is initialized in the Initialize routine.
Sub = sub name. Composed of EventName and event type.
In the example: `mEventName & "_ValuesChanged"`
`mEventName` Event name, initialized in the Initialize routine.
`"_ValuesChanged"` Event type.
Argument = prarameter(s) to transmit.

`mCallback` and `mEventName` initialization:

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

You can use Objects like List, Map, Type variable or Array to transmit more arguments.

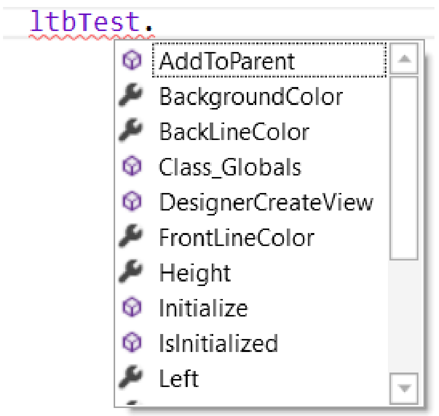
4.5.2 Event declaration in the class

These declarations are added at the top of the class.

```
'Events declaration
```

- `#Event: ValuesChanged(LimitLeft As Int, LimitRight As Int)`

  Important for intellisense to show properties and methods.

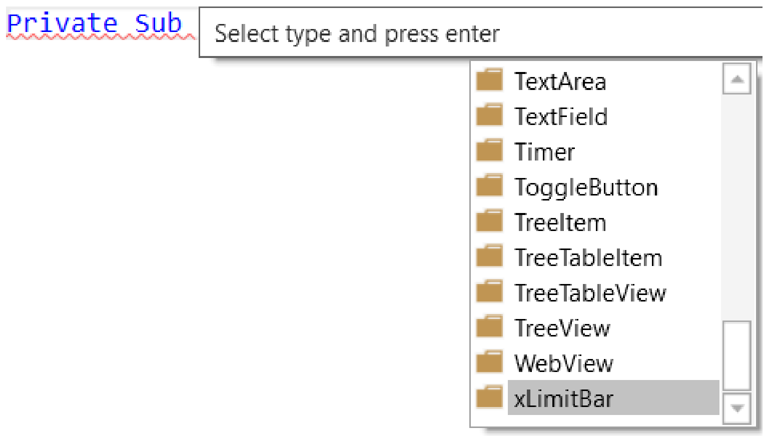


```
ltbTest.
```

AddToParent
BackgroundColor
BackLineColor
Class_Globals
DesignerCreateView
FrontLineColor
Height
Initialize
IsInitialized
Left

- `#RaisesSynchronousEvents: ValuesChanged`

  Important to show the intellisense for event routines.



```
Private Sub
```

Select type and press enter

TextArea
TextField
Timer
ToggleButton
TreeItem
TreeTableItem
TreeTableView
TreeView
WebView
xLimitBar

4.5.3 Code in the calling module

Event routine in the calling module.
In the example: **`ltbTest_ValuesChanged`** with two parameters.

```
Private ltbTest As xLimitBar

Private Sub ltbTest_ValuesChanged(LimitLeft As Int, LimitRight As Int)
   lblLimitLeft.Text = LimitLeft
   lblLimitRight.Text = LimitRight
End Sub
```

4.6 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 *CustomViews\CustomViewsSourceCode\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:

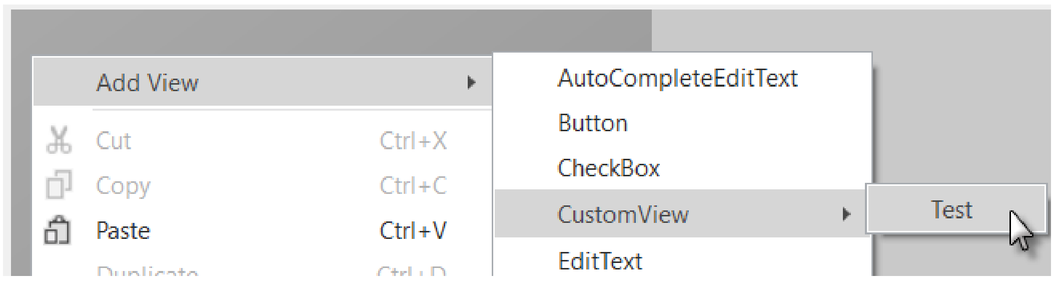
`Key`	Is the key value for the Map. This will be used to get the value from the Props map.
`DisplayName`	Is the name displayed in the Designer property grid.
`FieldType`	Is the type of the field. Possible values: String, Int, Double, Boolean or Color.
`DefaultValue`	Is the default value which is set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

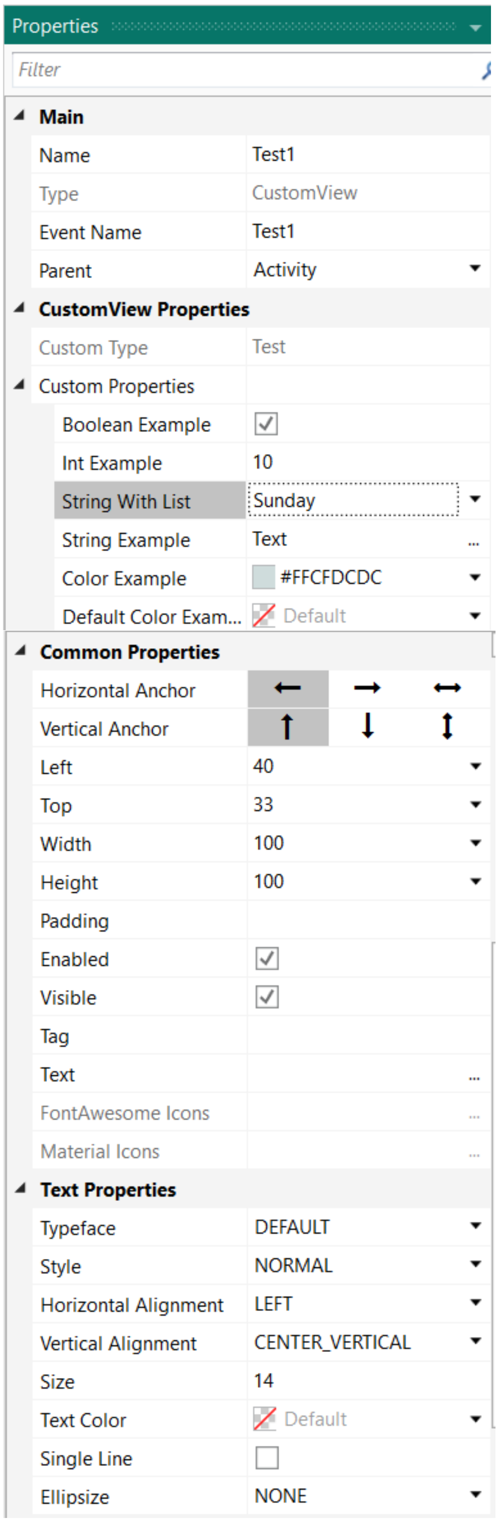
Optional fields:

`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 *Test* in the example.





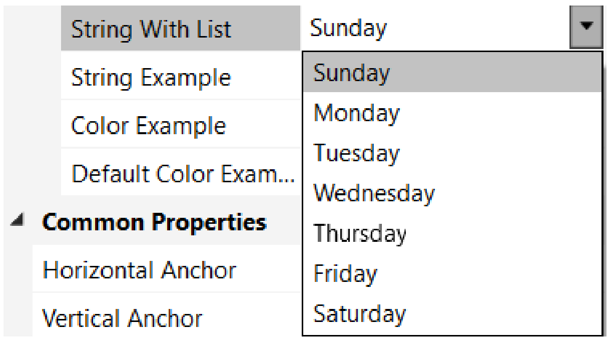
Properties	
Filter	
Main	
Name	Test1
Type	CustomView
Event Name	Test1
Parent	Activity
CustomView Properties	
Custom Type	Test
Custom Properties	
Boolean Example	✓
Int Example	10
String With List	Sunday
String Example	Text
Color Example	#FFCFDCDC
Default Color Exam...	Default
Common Properties	
Horizontal Anchor	← → ↔
Vertical Anchor	↑ ↓ ↕
Left	40
Top	33
Width	100
Height	100
Padding	
Enabled	✓
Visible	✓
Tag	
Text	
FontAwesome Icons	
Material Icons	
Text Properties	
Typeface	DEFAULT
Style	NORMAL
Horizontal Alignment	LEFT
Vertical Alignment	CENTER_VERTICAL
Size	14
Text Color	Default
Single Line	☐
Ellipsize	NONE

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

Custom properties:

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

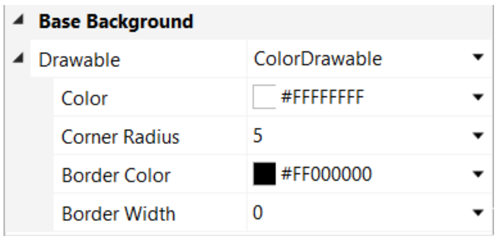
Example with the String With List property.



String With List	Sunday
String Example	
Color Example	
Default Color Exam...	
Common Properties	
Horizontal Anchor	
Vertical Anchor	

Common Properties:
The common properties like any view.

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



Base Background	
Drawable	ColorDrawable
Color	#FFFFFFFF
Corner Radius	5
Border Color	#FF000000
Border Width	0

v
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 Object, 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 = xui.PaintOrColorToColor(Props.Get("ColorExample"))
  DefaultColorTest = Props.Get("DefaultColorExample")
End Sub
```

You can also get properties with default values with `Props.GetDefault`, like this:

```
BooleanTest = Props.GetDefault("BooleanExample", True)
```

This is useful especially when you add new Designer properties later.
Then, if you do not open and close the Designer with an ‘old’ layout you will get an error because the new property is unknown in the old layout.

The declarations of the **`DesignerCreateView`** routine are different depending on the product.

B4A / B4i
```
Public Sub DesignerCreateView (Base As Panel, Lbl As Label, Props As Map)
```

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

XUI the Base declaration is different, `Object` instead of `Panel` or `Pane`.
```
Public Sub DesignerCreateView (Base As Object, 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
```

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

Private mTextColor As Int
mTextColor = xui.PaintOrColorToColor(Lbl.TextColor)
```
we must convert the color.

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

4.7 Add layouts to a CusomView

You can add specific layouts to a CustomView.

Code to load a layout.

```
Public Sub DesignerCreateView (Base As Object, Lbl As Label, Props As Map)
	mBase = Base
	
	Sleep(0)
	mBase.LoadLayout("CustomViewLayout")
End Sub
```

This code loads the *CustomViewLayout* onto mBase.
Note: `Sleep(0)` before loading the layout.
This is necessary to make sure that the dimensions of mBase are OK.

You need one layout file for each product!

If you create a B4X Library you must add the layout files in the Files folder of the library.
See more details in chapter Generate a B4X Library *.b4xlib.

4.8 Libraries

Why should we create 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.
- Share cross platform libraries (B4X Libraries).

4.8.1 Generate a B4X Library *.b4xlib

You can also create XUI cross platform libraries: **xxx.b4xlib**.
These libraries contain cross platform classes which do not need to be compiled as libraries.

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

- Code modules. All types are supported including Activities and Services.
- Files, including layout files.
- Optional manifest file with the following fields:
    - Version
    - Author
    - DependsOn (list of required libraries), Supported Platforms. Fields can be shared between the platforms or be platform specific.
    - Comment: adds a comment to the library.
      This comment will be shown in the Libraries Manager Tab.
- A folder Snippets to add code Snippets for the *.b4xlib library.
  These snippets can be copied directly in the IDE when the library is selected.

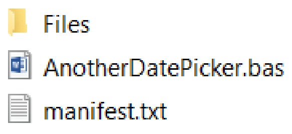
Files and code modules can also be platform specific.

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

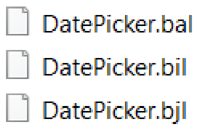
Note that the source code can be extracted from a b4x library.

B4X libraries appear like all other libraries in the Libraries Manager tab.

Example: AnotherDatePicker.b4xlib
The zip file structure:


Files
AnotherDatePicker.bas
manifest.txt

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


DatePicker.bal
DatePicker.bil
DatePicker.bjl

- *AnotherDatePicker.bas* is the Custom View file.
- *manifest.txt* contains:

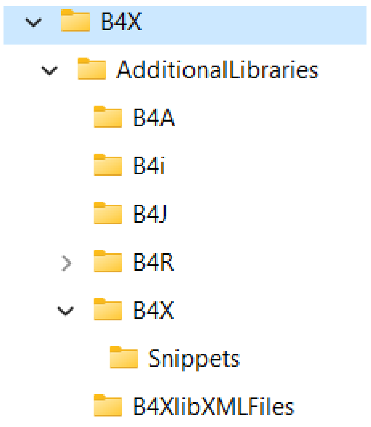
`Version=2.00`	version number.
`Author=Erel`	version number.
`B4J.DependsOn=jXUI, jDateUtils`	libraries used for B4J.
`B4A.DependsOn=XUI, DateUtils`	libraries used for B4A.
`B4i.DependsOn=iXUI, iDateUtils`	libraries used for B4i.
`IDE Comment= Test version 12`	IDE comment, optional

Be careful, no empty character between Comment and =.
IDE Comment = will not be displayed.

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

4.8.1.1 Code Snippets

4.8.1.2 AdditionalLibraries folder



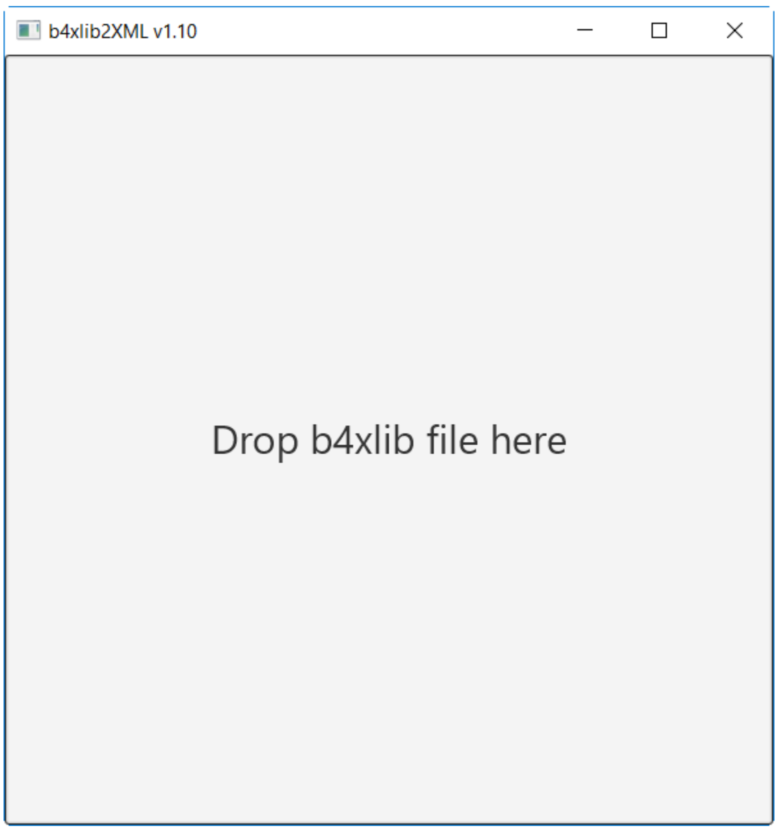
Structure of the AdditionalLibraries folder:

4.8.1.3 Xml help files for B4X Libraries

Erel has written an application to create xml help files for B4X Libraries.

You can download it from the forum HERE.

It looks like this:



Simply, drag and drop a xxx.b4xlib file into the from.
The xml file will be created, and you will be asked where you want to save it.

Tip:
Save all the b4xlib xml files into a specific folder.
Example: \AdditionalLibraries\B4XlibXMLFiles.

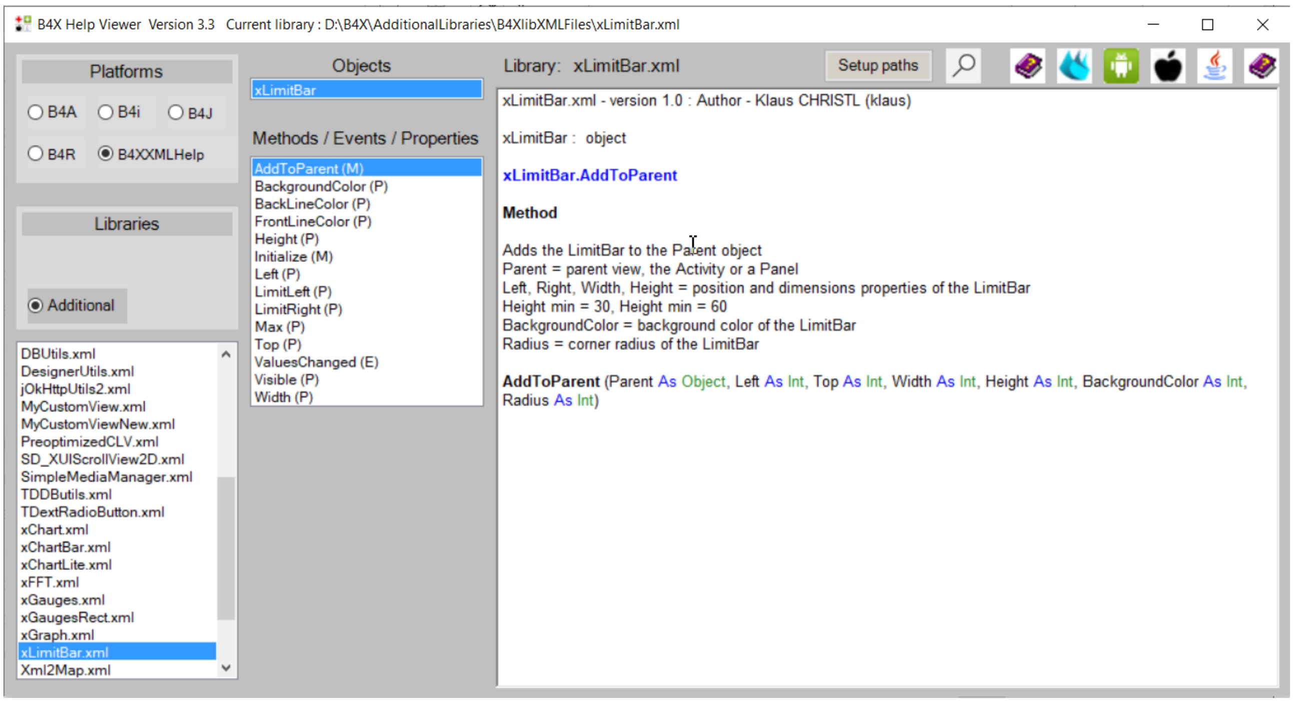
The xml files are useful for the HelpViewer applications like:
B4X Help Viewer
B4X Object Browser

If you have the B4X Help Viewer you can look at the help for the library.

Example with the B4A LimitBar library.

Select ◉ B4X and click on Addnl. and load xLimitBar.xml.

And the result.


B4X Help Viewer Version 3.3 Current library : D:\B4X\AdditionalLibraries\B4XlibXMLFiles\xLimitBar.xml
Platforms
B4A
B4i
B4J
B4R
B4XXMLHelp
Libraries
Additional
Objects
Methods / Events / Properties
Library: xLimitBar.xml
Setup paths

DBUtils.xml
DesignerUtils.xml
jOkHttpUtils2.xml
MyCustomView.xml
MyCustomViewNew.xml
PreoptimizedCLV.xml
SD_XUIScrollView2D.xml
SimpleMediaManager.xml
TDDButils.xml
TDextRadioButton.xml
xChart.xml
xChartBar.xml
xChartLite.xml
xFFT.xml
xGauges.xml
xGaugesRect.xml
xGraph.xml
xLimitBar.xml
Xml2Map.xml

xLimitBar

AddToParent (M)
BackgroundColor (P)
BackLineColor (P)
FrontLineColor (P)
Height (P)
Initialize (M)
Left (P)
LimitLeft (P)
LimitRight (P)
Max (P)
Top (P)
ValuesChanged (E)
Visible (P)
Width (P)

xLimitBar.xml - version 1.0 : Author - Klaus CHRISTL (klaus)

xLimitBar : object

xLimitBar.AddToParent

Method

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
Height min = 30, Height min = 60
BackgroundColor = background color of the LimitBar
Radius = corner radius of the LimitBar

AddToParent (Parent As Object, Left As Int, Top As Int, Width As Int, Height As Int, BackgroundColor As Int, Radius As Int)

4.8.2 Complie to a product specific library

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

For cross platform libraries you should use B4XLibraries!
You can use B4XLibs even for mono platform projects.

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, an xxx.a and a xxx.h file are created in the Mac Libs folder.

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

```
#ExcludeFromLibrary: True
```

The Main module is always excluded from the library!

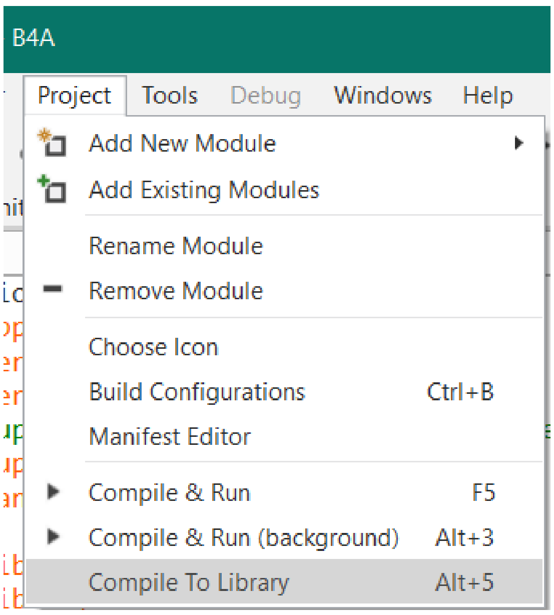
The Main module and the other excluded modules can be used to test the library.

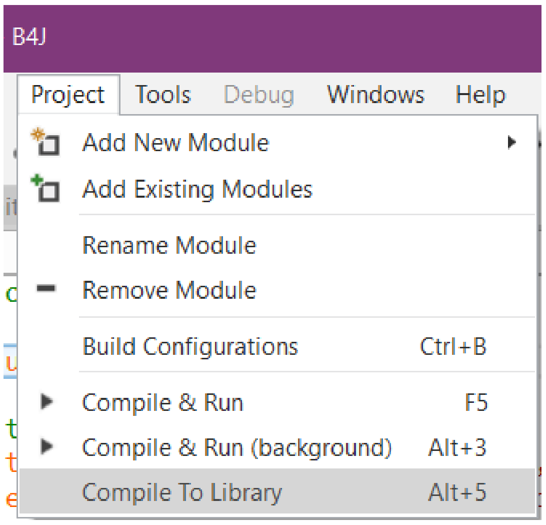
You can reference the library from other projects and access the same functionality as in the original project.

XUI:
With XUI, a cross platform CustomView class code module can be a unique module shared between the projects for the three products. You can make a B4XLibrary with it.
But, if you want to compile the CustomView to a library, you must compile three libraries, one for each product, because the library code is product specific.

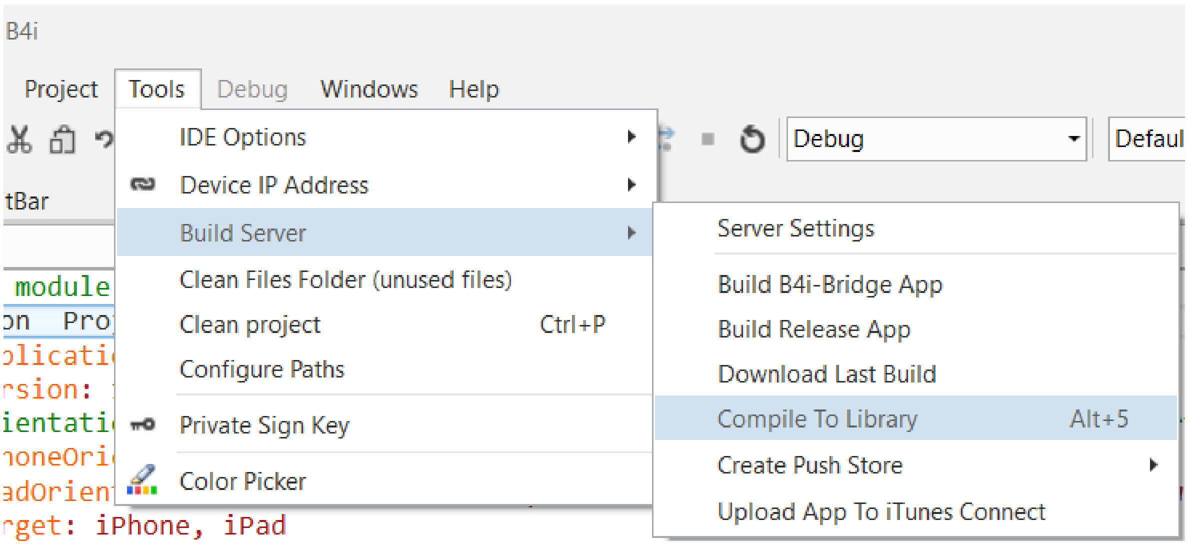
Compiling to a platform specific library is quite simple.
Under Project menu there is the compile option - "Compile To Library (Alt + 5)".
When you choose this option all the modules **except** of the main activity are compiled into a library.

B4A **B4J**



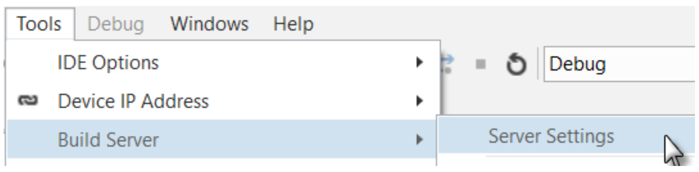


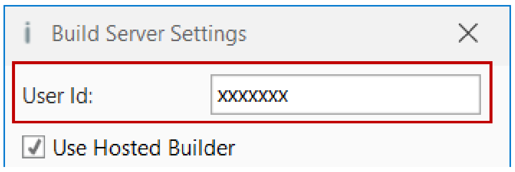
B4i



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.





User Id: xxxxxxx

[✓] Use Hosted Builder

4.8.2.1 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	B4i	B4J
`#LibraryName: xLimitBar`	`#LibraryName: ixLimitBar`	`#LibraryName: jxLimitBar`
`#LibraryAuthor: Klaus Christl`	`#LibraryAuthor: Klaus Christl`	`#LibraryAuthor: Klaus Christl`
`#LibraryVersion: 1.0`	`#LibraryVersion: 1.0`	`#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, xLimitBar 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
```

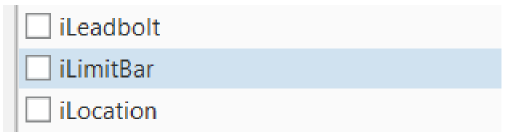
Notes

- 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.

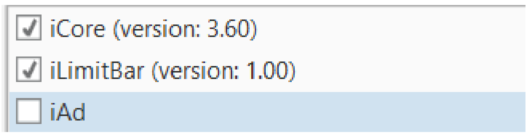
The library files are automatically saved in the Additional Libraries folder.

You can see it in the Libraries Manager Tab.

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



- [ ] iLeadbolt
- [ ] iLimitBar
- [ ] iLocation



- [✓] iCore (version: 3.60)
- [✓] iLimitBar (version: 1.00)
- [ ] iAd

Example with the B4i LimitBar project.

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

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: jxLimitBar Object: xLimitBar

```
Private ltbTest, ltbTest1 As xLimitBar
```

4.8.2.2 Tip for MaterialIcons and Fontawesome fonts

If you use FontAwesome or MaterialIcons fonts in a class which is compiled into a library, you may get an error like this one:

```
Error occurred on line: 25 (Main)
java.lang.RuntimeException: Font asset not found b4x_fontawesome.otf
      at android.graphics.Typeface.createFromAsset(Typeface.java:879)
```

To avoid this, add the line below in the Main module where you use the library, not in the code where you compile the library.

B4A:

```
If False Then Log(Typeface.MATERIALICONS)
```

and/or

```
If False Then Log(Typeface.FONTAWESOME)
```

B4i:

```
If False Then Log(Typeface.MATERIALICONS)
```

and/or

```
If False Then Log(Typeface.FONTAWESOME)
```

B4J:

```
If False Then Log(Typeface.MATERIALICONS)
```

and/or

```
If False Then Log(Typeface.FONTAWESOME)
```

4.9 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.

4.10 Intellisense help

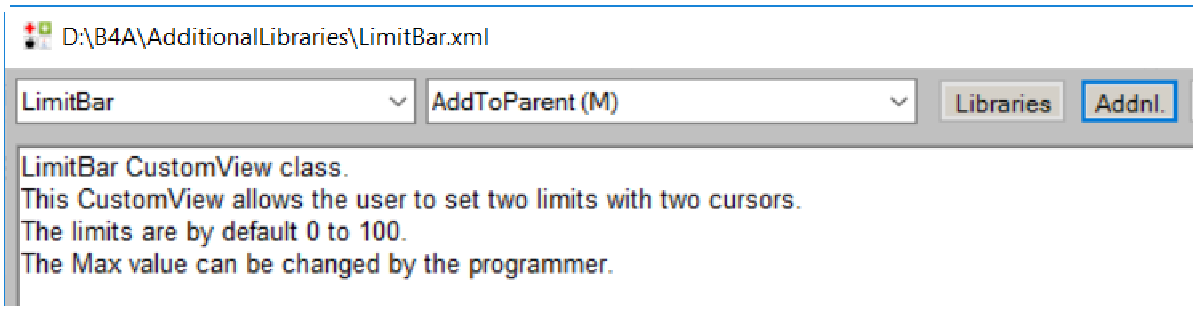
It is advised to add help comments in the code for the users of your library.

4.10.1 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.



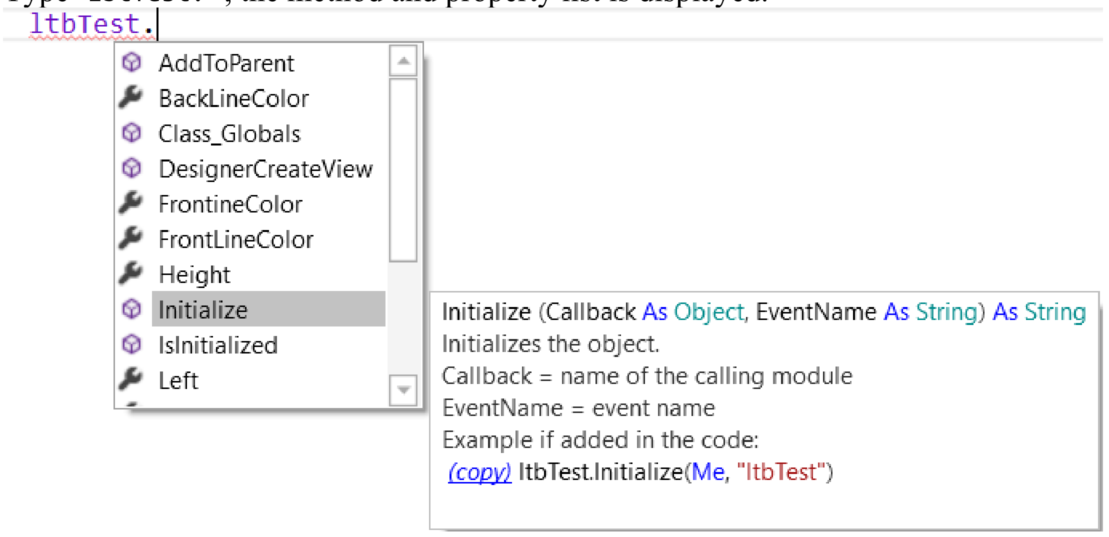
LimitBar CustomView class.
This CustomView allows the user to set two limits with two cursors.
The limits are by default 0 to 100.
The Max value can be changed by the programmer.

4.10.2 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.



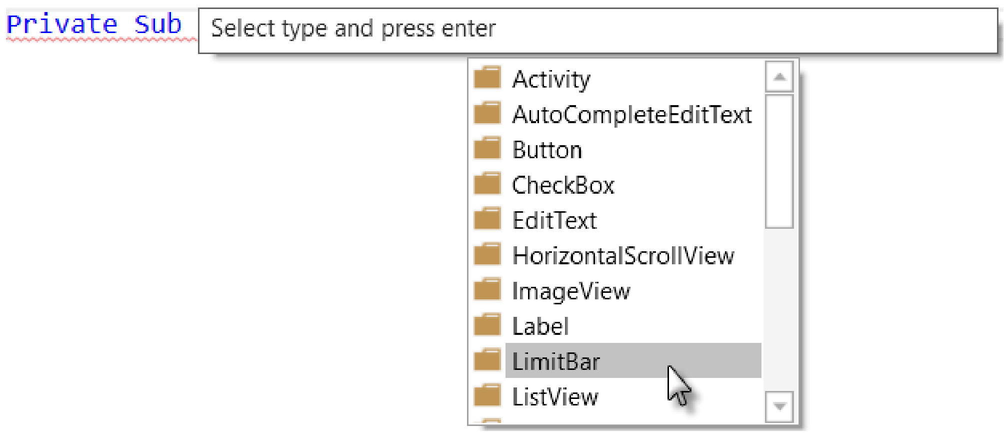
Initialize (Callback As Object, EventName As String) As String
Initializes the object.
Callback = name of the calling module
EventName = event name
Example if added in the code:
(copy) ltbTest.Initialize(Me, "ltbTest")

4.10.3 Comments before an event routine

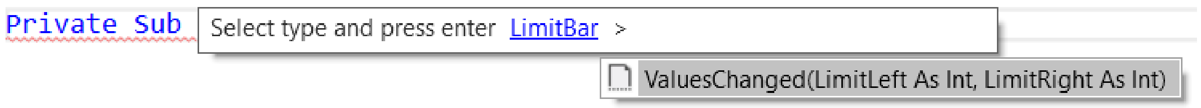
Events declared on top of the code in the class module with `#Event:` are displayed as intellisense when the class is compiled to a Library.

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

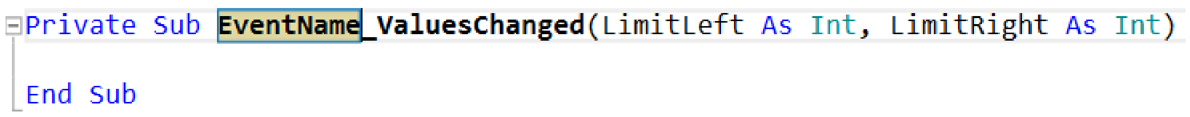
When you use the library in another project, type ‘`Public Sub ‘` (with a space at the end) and press on Tab to show the objects list.



Select LimitBar.

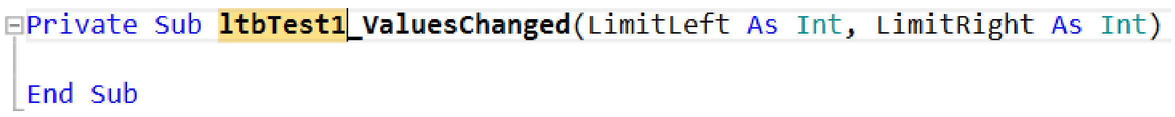

Private Sub
Select type and press enter LimitBar >
ValuesChanged(LimitLeft As Int, LimitRight As Int)

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



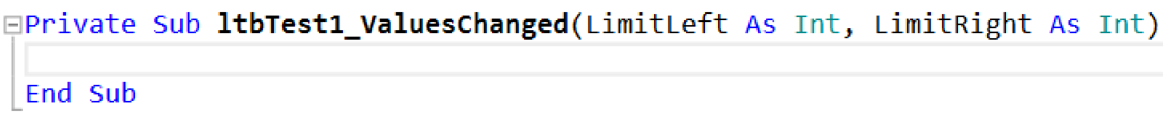
```
Private Sub EventName_ValuesChanged(LimitLeft As Int, LimitRight As Int)

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
```

4.11 Code Snippets

You can add code snippets to the IDE.
A code snippet is a piece of code that can be added to your code in very few clicks.
The snippet can include any number of variables or placeholders that will be highlighted and edited right after the snippet is inserted.

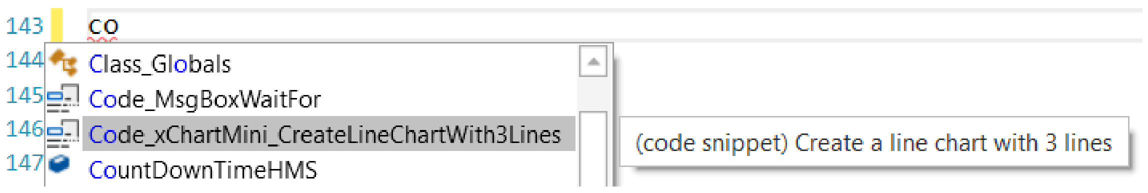
These are simple text files with the code.

You can use any number of variables in a code snippet. These variables are between two $ characters like $MyVarable$.
When the code snippet is copied, these variables are copied without the $ characters and highlighted.
An example is shown in the *Code snippet in a b4xlib library* chapter.

There is a reserved variable name *end* which moves the cursor to this location when the code snippet was copied.
An example is shown in *the Simple code snippet in the AdditionalLibraries\B4X\Snippets folder* chapter.

4.11.1 Copy a Snippet in the IDE

In the IDE type 'co' for code.
All snippets begin with Code_.
When you click on them you will see a comment if there is one.



```
co
Class_Globals
Code_MsgBoxWaitFor
Code_xChartMini_CreateLineChartWith3Lines
CountDownTimeHMS
```

(code snippet) Create a line chart with 3 lines

In the example above, you see two snippets.

- Code_MsgBoxWaitFor
  A code in the AdditionalLibraries\B4X\Snippets folder.
  You see:
  - Code the code snippet prefix
  - MsgBoxWaitFor the name of the snippet
- Code_xChartMini_CreateLineChartWith3Lines
  A code snippet in the xChartMini b4xlib library.
  - Code the code snippet prefix
  - xChartMini the name of the b4xlib library CustomView
  - CreateLineChartWith3Lines the name of the snippet

When you click on Code_xChartMini_CreateLineChartWith3Lines the code snippet is copied in the editor.

4.11.2 Code Snippet in a b4xlib library CustomView

It is a snippet for the xChartMini b4xlib library to create a line chart.

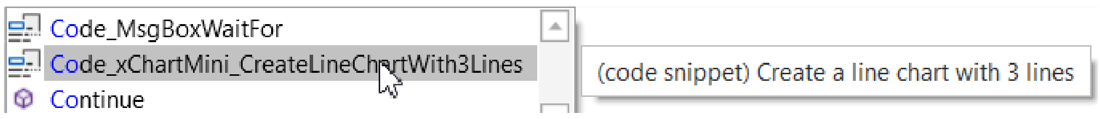
The code:

```
'Create a line chart with 3 lines
Private Sub CreateLineChart
   ' clear previous data
   $xChart1$.ClearData

   ' initialize the line data
   $xChart1$.AddLine("Random", xui.Color_Blue)
   $xChart1$.AddLine("Cos", xui.Color_Red)
   $xChart1$.AddLine("Sin", xui.Color_Magenta)

   ' set the max and min scale values
   $xChart1$.YScaleMaxValue = 10
   $xChart1$.YScaleMinValue = 0

   ' Add the line points.
   Dim Val1, Val2, Val3 As Double
   For i = 0 To 720 Step 15
      ' In the case of 2 lines or more we are adding an array of values.
      ' One for each line.
      Val1 = Rnd(-100, 101) / 50 + 5
      Val2 = 3 * CosD(i) + 5
      Val3 = 4 * SinD(i) + 5
      $xChart1$.AddLineMultiplePoints(i, Array As Double(Val1, Val2, Val3), i Mod 90 = 0)
   Next

   ' draw the chart
   $xChart1$.DrawChart
End Sub
```

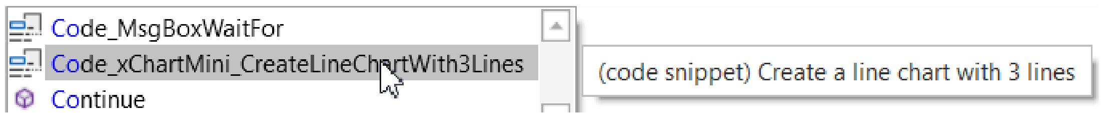
In the code i use a variable *$xChat1$*, this is used as a placeholder for the xChart object in the code. When the user copies the snippet in his code, *$xChat1$* will be highlighted and the user can change it. When he changes the first occurrence, all the others will be automatically updated.

The code is saved as a text file in the Snippets folder of the xChartMini b4xlib library with the name CreateLineChartWith3Lines.
In the IDE we see:

Code_	this is the code snippet prefix.
xChartMini	this is the name of the b4xlib library.
CreateLineChartWith3Lines	this is the file name of the snippet.
Create a line chart with 3 lines	this is the comment, it is the first line in the snippet code above.

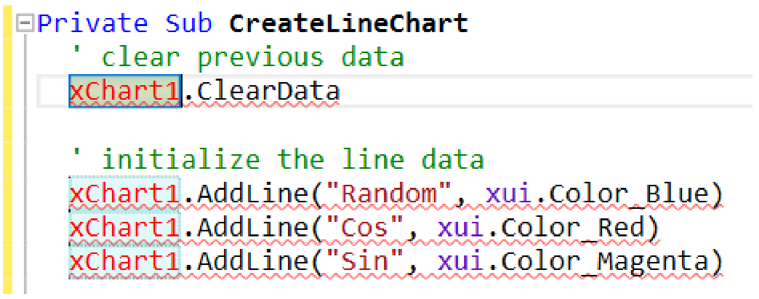

Code_MsgBoxWaitFor
Code_xChartMini_CreateLineChartWith3Lines
Continue
(code snippet) Create a line chart with 3 lines

When we click on:



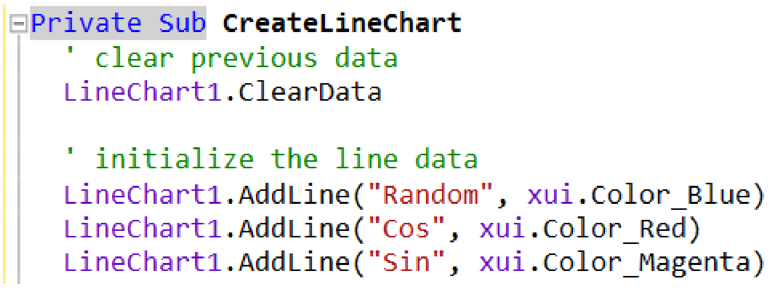
(code snippet) Create a line chart with 3 lines

The code is copied into the IDE and the variable xChart1 is highlighted.
In red in this cas because xChart1 has not been declared.



```
Private Sub CreateLineChart
    ' clear previous data
    xChart1.ClearData

    ' initialize the line data
    xChart1.AddLine("Random", xui.Color_Blue)
    xChart1.AddLine("Cos", xui.Color_Red)
    xChart1.AddLine("Sin", xui.Color_Magenta)
```

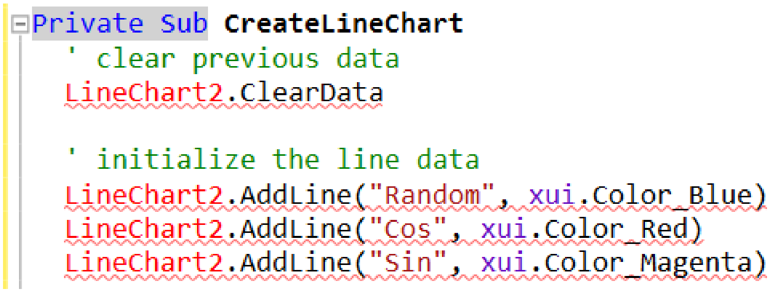
When we change xChart1 into LineChart1, all xChart1 variables are updated.



```
Private Sub CreateLineChart
    ' clear previous data
    LineChart1.ClearData

    ' initialize the line data
    LineChart1.AddLine("Random", xui.Color_Blue)
    LineChart1.AddLine("Cos", xui.Color_Red)
    LineChart1.AddLine("Sin", xui.Color_Magenta)
```

xChart1 is replaced by LineChart1, the color of LineChart1 is OK because the variable had already been declared in the test project.

If the new variable had not yet been declared, the new variable would remain in red.



```
Private Sub CreateLineChart
    ' clear previous data
    LineChart2.ClearData

    ' initialize the line data
    LineChart2.AddLine("Random", xui.Color_Blue)
    LineChart2.AddLine("Cos", xui.Color_Red)
    LineChart2.AddLine("Sin", xui.Color_Magenta)
```

4.12 CustomViews (XUI)

XUI CustomViews are like ‘standard’ CustomViews but cross platform.

4.12.1 CustomViews (XUI) class structure

Several declarations and routines are predefined:

Default template of a CustomView (XUI) class:

```
#DesignerProperty: Key: BooleanExample, DisplayName: Show Seconds, FieldType: Boolean,
DefaultValue: True
#DesignerProperty: Key: TextColor, DisplayName: Text Color, FieldType: Color,
DefaultValue: 0xFFFFFFFF, Description: Text color

Sub Class_Globals
   Private mEventName As String 'ignore
   Private mCallBack As Object 'ignore
   Public mBase As B4XView 'ignore
   Private xui As XUI 'ignore
End Sub

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

'Base type must be Object
Public Sub DesignerCreateView (Base As Object, Lbl As Label, Props As Map)
   mBase = Base
   Tag = mBase.Tag
   mBase.Tag = Me
   Dim clr As Int = xui.PaintOrColorToColor(Props.Get("TextColor")) 'Example of getting
a color value from Props
End Sub

Public Sub Base_Resize (Width As Double, Height As Double)

End Sub
```

It is similar to the ‘standard’ CustomView class.
The main differences are:

- Declaration of the XUI library.
- In the DesignerCreateView routine, the type of Base is Object and not a Panel or Pane.
- Example on how to get a color property.

All the other principles are the same, except that you might use B4X objects instead of ‘standard’ objects.

4.13 GetView

To be able to get CustomView from a parent view, you can use the code below.

In the CustomView class add the code below in the DesignerCreateView routine, mBase is the base view of the CustomView:

```
mBase.Tag = Me
```

And in the main code, MyCustomView is the CustomView name:

```
Private MyView As MyCustoView = Parent.GetView(0).Tag
```

4.14 Add many CustomViews in the code

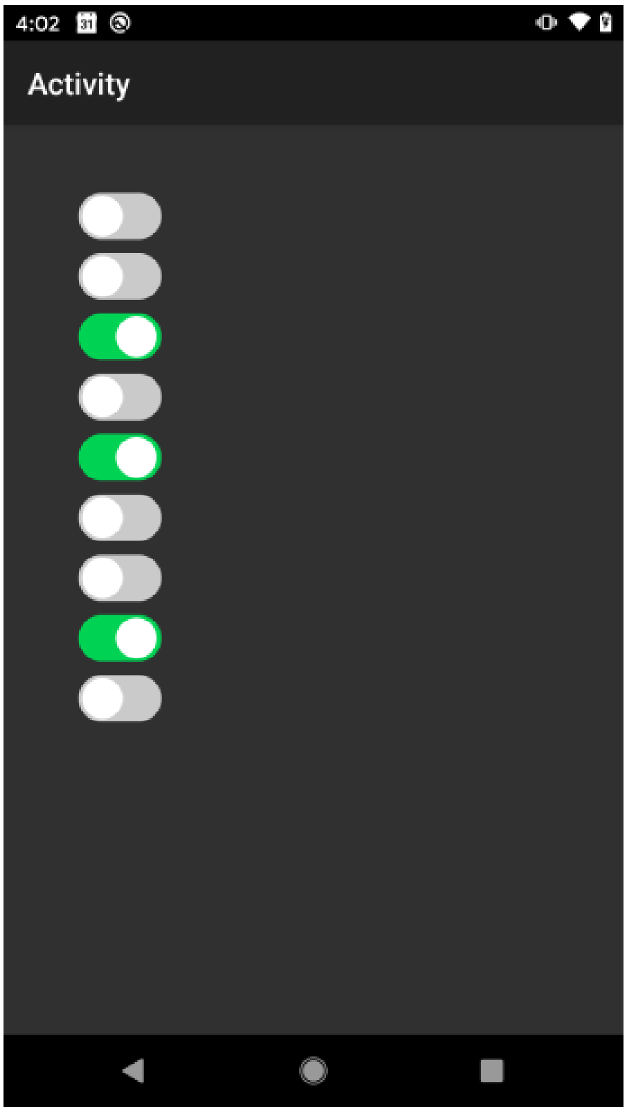
Custom views are designed to be added with the designer.

It is however very simple to create a layout file with the custom view and load it multiple times.

Tip: remove the call to AutoScaleAll from the designer script.

Complete example:

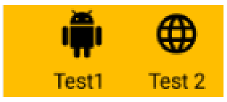
```
Sub Globals
   Private B4XSwitch1 As B4XSwitch
End Sub

Sub Activity_Create(FirstTime As Boolean)
   For i = 1 To 20
      AddSwitch(50dip, 40dip * i, i)
   Next
End Sub

Sub AddSwitch (Left As Int, Top As Int, Tag As Object) As B4XSwitch
   Activity.LoadLayout("B4XSwitch")
   B4XSwitch1.mBase.Left = Left 'B4XSwitch1 global variable will point to the last one
added
   B4XSwitch1.mBase.Top = Top
   B4XSwitch1.Tag = Tag
   Return B4XSwitch1
End Sub

Sub B4XSwitch1_ValueChanged (Value As Boolean)
   Dim switch As B4XSwitch = Sender
   Log(switch.Tag)
End Sub
```



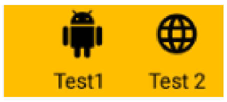
5 First example xCustomButton

We will make a simple xCustomButton.

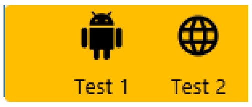
B4A



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 all declared as B4XViews.

The xCustomButton can be added in the Designer or in the code.
For B4A, in the Designer, you must set the Alpha property to 0 to make sure that the base Panel is transparent.

There are two types of buttons:

- STANDARD with two events Click and LongClick.
- TOGGLE with one event CheckedChange.

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.

It is a XUI B4XPages project.
The source code is in the CustomViewsSourceCode\xCustomButtonDemo folder.

5.1 Event declarations

First, we declare the events on top of the code.

```
'xCustomButton Class
#Event: Click
#Event: LongClick
#Event: CheckedChange(Checked As Boolean)
#RaisesSynchronousEvents: Click
#RaisesSynchronousEvents: LongClick
#RaisesSynchronousEvents: CheckedChange
```

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 according 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.

```
'xCustomButton is a button based on a Panel with two Labels
'one with a Material Icon and the other with text.
'It has two button types:
'STANDARD with two events: Click and LongClick.
'TOGGLE with one event CheckedChange.
Sub Class_Globals
```

5.4 Global variables

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

```
Sub Class_Globals
#If B4J
  Private fx As JFX
#End If
  Private xui As XUI

  Private mEventName As String
  Private mCallBack As Object
  Private mBase As B4XView
  Public Tag As Object

  Private mLeft, mTop, mWidth, mHeight As Int
  Private mText, mIcon As String

  Private mLabelFont, mIconFont As B4XFont

  Private mTextColor As Int
  Private mIconTextSize, mLabelTextSize As Double

  Private mLabel, mIconLabel As B4XView
  Private mParent As B4XView
  Private cvsBase As B4XCanvas
  Private rectBase As B4XRect
  Private mPressedColor, mBackgroundColor, mOnStateColor, mOnStateBackgroundColor As
Int
  Private mONState = False As Boolean
  Private mButtonType As String
  Private mOnStateText As String

  Private tmrClick As Timer
  Private mTimeDown, mClickTime As Long      'used to distinguish Click and LongClick
End Sub
```

5.5 Initialize routine

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

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

  mIcon = Chr(0xE859)
  mText = "Test"
  mTextColor = xui.Color_Black

  mClickTime = 400
  tmrClick.Initialize("tmrClick", mClickTime)

  mPressedColor = xui.Color_Yellow
  mButtonType = "STANDARD"
  mBackgroundColor = xui.Color_Transparent
  mOnStateColor = xui.Color_Blue
End Sub
```

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.

```
Public Sub DesignerCreateView (Base As Object, Lbl As Label, Props As Map)
   'we use a Dummy object to get the Tag property and the Parent.
   Private mDummy As B4XView
   mDummy = Base
   mLeft = mDummy.Left
   mTop = mDummy.Top
   mWidth = mDummy.Width
   mHeight = mDummy.Height
   Tag = mDummy.Tag
   mIcon = Lbl.Text
   mText = Props.Get("Text")
   mPressedColor = xui.PaintOrColorToColor(Props.Get("PressedColor"))
   mBackgroundColor = xui.PaintOrColorToColor(Props.Get("BackgroundColor"))
   If mBackgroundColor = 16777215 Then
       mBackgroundColor = xui.Color_Transparent
   End If
   mButtonType = Props.Get("ButtonType")
   mOnStateText = Props.Get("OnStateText")
   mOnStateColor = xui.PaintOrColorToColor(Props.Get("OnStateColor"))
   mOnStateBackgroundColor =
xui.PaintOrColorToColor(Props.Get("OnStateBackgroundColor"))

   'we create a new mBase object to get the Touch event
   mBase = xui.CreatePanel("mBase")
   xParent = mDummy.Parent
   xParent.AddView(mBase, mLeft, mTop, mWidth, mHeight)
   mBase.Tag = Me

   mTextColor = xui.PaintOrColorToColor(Lbl.TextColor)

   'we remove the Base object, no more needed
   mDummy.RemoveViewFromParent

   mIconFont = Lbl.As(B4XView).Font

   InitClass
End Sub
```

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 xCustomButton in the code.
We memorize the position, dimensions and properties.
And call InitClass

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

  mBase = xui.CreatePanel("mBase")

  mParent.AddView(mBase, mLeft, mTop, mWidth, mHeight)
  mBase.Tag = Me

  mIcon = Icon
  mText = Text
  mTextColor = xui.PaintOrColorToColor(TextColor)

  xIconFont = xui.CreateMaterialIcons(10)

  InitClass
End Sub
```

5.9 InitClass routine

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

```
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
   Private lbl As Label
   lbl.Initialize("")
   mIconLabel = lbl

   mIconTextSize = mHeight / 2 / xui.Scale
   mLabelTextSize = lblWidth / 3 / xui.Scale
   mLabelFont = xui.CreateDefaultFont(10)

   mIconLabel.Font = mIconFont
   mIconLabel.SetTextAlignment("CENTER", "CENTER")
   mIconLabel.TextSize = mIconTextSize
   mIconLabel.TextColor = xui.PaintOrColorToColor(mTextColor)
   mBase.AddView(mIconLabel, lblLeft, 0, lblWidth, lblWidth)
   mIconLabel.Text = mIcon

   'initialize and add the text Label
   Private lbl As Label
   lbl.Initialize("")
   mLabel = lbl
   mLabel.SetTextAlignment("TOP", "CENTER")
   mLabel.Font = mLabelFont
   mLabel.TextSize = mLabelTextSize
   mLabel.TextColor = xui.PaintOrColorToColor(mTextColor)
   mBase.AddView(mLabel, 0, 2 * mHeight / 3, mWidth, mHeight / 3)
   mLabel.Text = mText

   cvsBase.Initialize(mBase)
   rectBase.Initialize(0, 0, mBase.Width, mBase.Height)
End Sub
```

5.10 Click / LongClick event routines

Here we have a problem, because B4J does not have a Click nor a LongClick event.
Therefore, we create our own cross-platform with the Touch event.
We need a Timer for the LongClick event.

```
Private Sub mBase_Touch (Action As Int, X As Float, Y As Float)
   Select Action
      Case mBase.TOUCH_ACTION_DOWN
         mTimeDown = DateTime.Now
         tmrClick.Enabled = True
         Select mButtonType
            Case "STANDARD"
               mBase.Color = mPressedColor
            Case "TOGGLE"
               If mONState = False Then
                  mBase.Color = mOnStateBackgroundColor
                  mONState = True
                  mLabel.Text = mOnStateText
                  mLabel.TextColor = mOnStateColor
                  mIconLabel.TextColor = mOnStateColor
               Else
                  If mBackgroundColor = xui.Color_Transparent Then
                     cvsBase.ClearRect(rectBase)
                     cvsBase.Invalidate
                  Else
                     mBase.Color = mBackgroundColor
                  End If
                  mONState = False
                  mLabel.Text = mText
                  mLabel.TextColor = mTextColor
                  mIconLabel.TextColor = mTextColor
               End If
         End Select
      Case mBase.TOUCH_ACTION_UP
         Select mButtonType
            Case "STANDARD"
               If DateTime.Now - mTimeDown <= mClickTime Then
                  If xui.SubExists(mCallBack, mEventName & "_Click", 0) Then
                     CallSubDelayed(mCallBack, mEventName & "_Click")
                     tmrClick.Enabled = False
                  End If
               End If
               mBase.Color = mBackgroundColor
            Case "TOGGLE"
               If xui.SubExists(mCallBack, mEventName & "_CheckedChange", 1) Then
                  CallSubDelayed2(mCallBack, mEventName & "_CheckedChange", mONState)
                  tmrClick.Enabled = False
               End If
         End Select
   End Select
End Sub
```

And the Timer routine :

```
Private Sub tmrClick_Tick
   If xui.SubExists(mCallBack, mEventName & "_LongClick", 0) Then
      CallSubDelayed(mCallBack, mEventName & "_LongClick")
      tmrClick.Enabled = False
   End If
End Sub
```

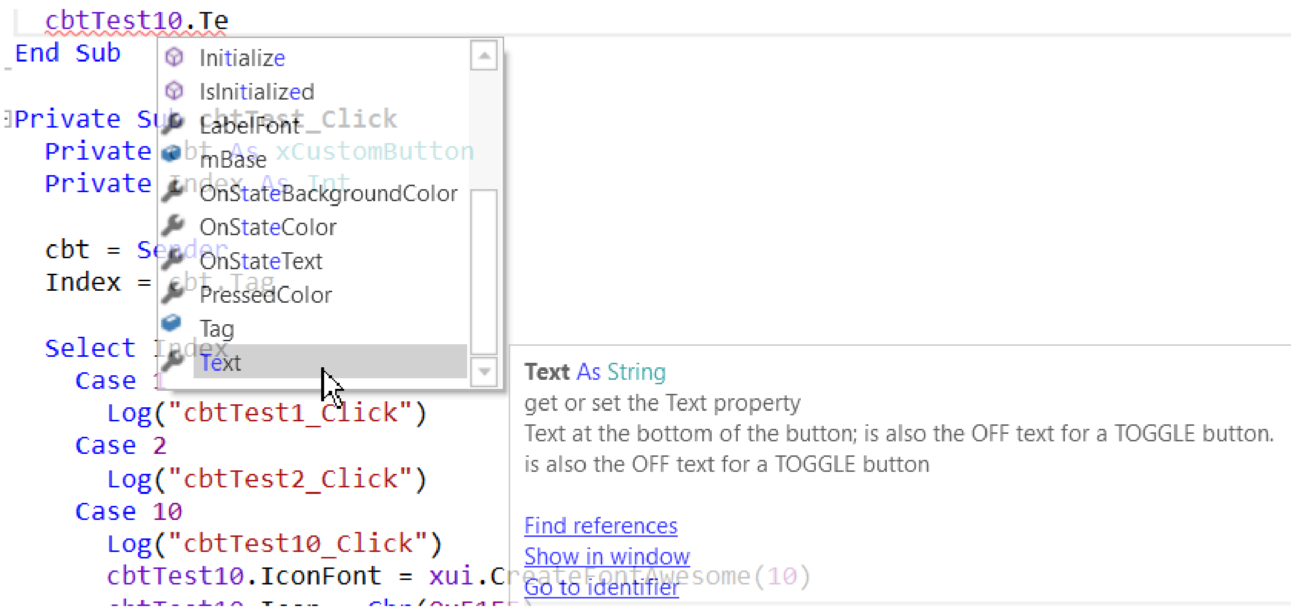
5.11 Property routines

Not all property routines are shown, only the two below, the principle is the same for all the others.

Text property.

```
'get or set the Text property
'Text at the bottom of the button; is also the OFF text for a TOGGLE button.
'is also the OFF text for a TOGGLE button
Public Sub setText(Text As String)
       mText = Text
End Sub

Public Sub getText As String
       Return mText
End Sub
```

The comments above the Public Sub line are shown in the intellisense.



```
  cbtTest10.Te
End Sub

Private Sub cbtTest_Click
  Private cbt As xCustomButton
  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")
      cbtTest10.IconFont = xui.CreateFontAwesome(10)
```

Text As String
get or set the Text property
Text at the bottom of the button; is also the OFF text for a TOGGLE button.
is also the OFF text for a TOGGLE button

Find references
Show in window
Go to identifier

IconFont property.

```
'set the icon IconFont property
'should be "FontAwesome or "Material Icons" B4XFont font
Public Sub setIconFont(IconFont As B4XFont)
   mIconFont = IconFont
   mIconLabel.Font = mIconFont
   mIconLabel.TextSize = mIconTextSize
End Sub

Public Sub getIconFont As B4XFont
   Return mIconFont
End Sub
```

5.12 Main code

In the B4XMainPage module

5.12.1 Globals

In addition to the default declarations only one variable `cbtTest10`, for the xCustomButton added in the code,.

```
Sub Class_Globals
   Private Root As B4XView
   Private xui As XUI

   Private cbtTest10 As xCustomButton
End Sub
```

5.12.2 Program start B4XPage_Created

We have the common declaration on top.
Then we set the MainPage title and add an xCustomButton in the code.

```
Private Sub B4XPage_Created (Root1 As B4XView)
   Root = Root1
   Root.LoadLayout("Main")

   B4XPages.SetTitle(B4XPages.MainPage, "xCustomButtonDemo")

   'adds a button in the code
   cbtTest10.Initialize(Me, "cbtTest")
   cbtTest10.AddToParent(Root, 100dip, 100dip, 60dip, 60dip, xui.Color_RGB(0, 0, 139),
Chr(0xE855), "Test 10")
   cbtTest10.Tag = 10
   cbtTest10.PressedColor = xui.Color_ARGB(96, 255, 0 ,0)
End Sub
```

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 xCustomButtons 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 Create the b4xlib library

We make a B4X Library for the xCustomButton CustomView.

We generate the manifest file, it's a text file with the content below,

```
Version=1.0
Author=Klaus CHRISTL (klaus)
B4J.DependsOn=jXUI
B4A.DependsOn=XUI
B4i.DependsOn=iXUI
```

`Version:`	the version number
`Author:`	the author's name
`B4J.DependsOn:`	the list of all B4J libraries the custom view depends on.
`B4A.DependsOn:`	the list of all B4A libraries the custom view depends on.
`B4i.DependsOn:`	the list of all B4iJ libraries the custom view depends on.

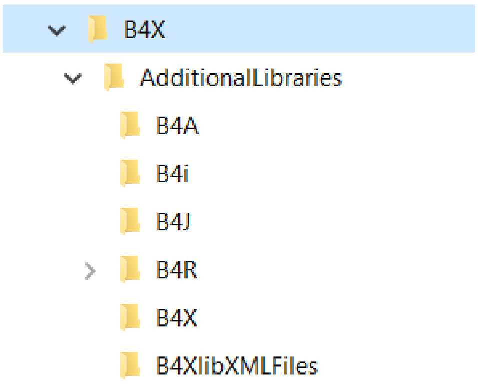
And save it with the name: *manifest.txt*.

In our case xCustomButton depends only on the xui libraries.

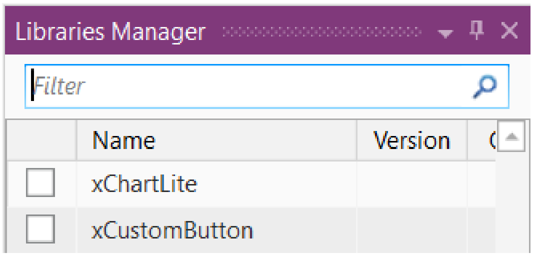
Then we zip the manifest.txt file and the xCustomButton.bas file to generate the xCustomButton.b4xlib file.
The extension must be *b4xlib*.

Copy this file into the \AdditionalLibraries\B4X folder.

Remember the subfolder structure of the AdditionalLibraries folder.



When you refresh the Libraries Manager Tab, you will see the new library.



More information in chapter Generate a B4XLibrary.

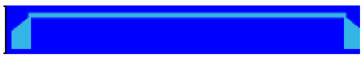
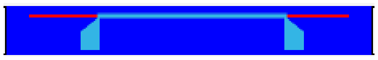
6 XUI xLimitBar

Another concrete example, a LimitBar, which is a XUI CustomView and b4xlib.

The demo project is a B4XPages project.

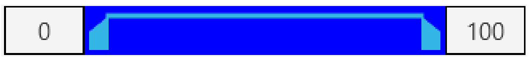
The source code is in the CustomViewsSourceCode\xLimitBarDemo folder

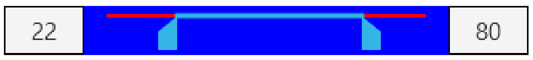
The xLimitBar looks like this:

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

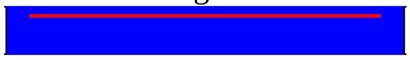
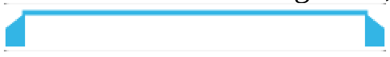
In the demo program, we add two labels, one on each side, to display the two limit values these are not part of the custom view.





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

We use two B4XView panels:

- mBase 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 B4XCanavas objects:

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

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
```

6.2 Custom properties for the Designer

The xLimitBar has following custom properties:

- Max
  Sets or gets the max limit value when the curser is at the rightest 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.
- Radius
  Sets or gets the corner radius.
- BackgroundColor
  Sets or gets the background. The default value is blue (0xFF0000FF).
- 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).

To 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 Global variables

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

```
Sub Class_Globals
   Private xui As XUI
   Private mCallback As Object          ' calling module
   Private mEventName As String         ' event name
   Public xBase As B4XView
   Private xParent As B4XView
   'two paths for the cursor shape and the background
   Private CursorPaths(2), BackgroundPath As B4XPath

   Private mLeft, mTop, mWidth, mHeight, mRadius As Double

   Private ltbPanelFront As B4XView     ' the background panel
   Private cvsPanelBack As B4XCanvas    ' the background canvas
   Private cvsPanelFront As B4XCanvas   ' the foreground canvas
   Private rectPanelFront As B4XRect    ' a rectangle for the foreground canvas

   Private mBackgroundColor As Int      ' color for the background
   Private mBackLineColor As Int        ' color for the background line
   Private mFrontLineColor As Int       ' 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
End Sub
```

6.4 Initialize routine

Then we need the routine to initialize the xLimitBar, 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

```
'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 = xui.Color_Blue
   mBackLineColor = xui.Color_Black
   mFrontLineColor = xui.Color_RGB(51, 181, 229)
   mRadius = 10dip
   mMargin = 15dip
   mMaxValue = 100
   mLimit(0) = 0
   mLimit(1) = mMaxValue
End Sub
```

6.5 DesignerCreateView routine

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

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

   ' we memorize the Base Width and Height properties
   mLeft = xBase.Left
   mTop = xBase.Top
   mWidth = xBase.Width
   mHeight = xBase.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

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

   'we get the Radius and color properties
   mRadius = DipToCurrent(Props.Get("Radius"))
   mBackgroundColor = xui.PaintOrColorToColor(Props.Get("BackgroundColor"))
   mBackLineColor = xui.PaintOrColorToColor(Props.Get("BackLineColor"))
   mFrontLineColor = xui.PaintOrColorToColor(Props.Get("FrontLineColor"))

   #If B4A
      InitClass      ' initializes the common parts for Designer and code
   #End If
End Sub
```

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

As the xLimitBar custom view can also be added in the code we initialize the rest in the `InitClass` routine.
In B4A, the InitClass routine is called from the **`DesignerCreateView`** routine.
In B4i and B4J it is called from the **`Base_Resize`** routine to make sure that the width and height are known!

```
#If B4A
   InitClass         ' initializes the common parts for Designer and code
#End If
```

6.6 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.
The routine is set to `Public` to allow to call it from outsides.
We use it in the `Public Sub` **`B4XMainPage_Resize`** routine to adjust the xLimitBar added in the code when the main page is resized.

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

  If ltbPanelBack.IsInitialized = False Then
    InitClass            ' initializes the common parts for Designer and code
  Else
    rectPanelFront.Width = mWidth
    rectPanelFront.Height = mHeight

    ltbPanelBack.Width = mWidth
    ltbPanelBack.Height = mHeight

    ltbPanelFront.Width = mWidth
    ltbPanelFront.Height = mHeight

    cvsPanelBack.Resize(mWidth, mHeight)
    cvsPanelFront.Resize(mWidth, mHeight)

    InitCursors
    DrawBackGround
    DrawCursors
  End If
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 xLimitBar is added in the Designer.
It is not called when the xLimitBar is added in the code.

6.7 AddToParent routine

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

```
'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 xLimitBar
'Height min = 30, Height min = 60
'BackgroundColor = background color of the xLimitBar
'Radius = corner radius of the xLimitBar
Public Sub AddToParent(Parent As Object, Left As Int, Top As Int, Width As Int, Height
As Int, BackgroundColor As Int, Radius As Int)
  mLeft = Left
  mTop = Top
  mWidth = Width
  mHeight = Max(Height, 30dip)        ' limits the height to min 30 pixels
  mHeight = Min(Height, 60dip)        ' limits the height to max 60 pixels
  mRadius = Min(Radius, Height / 2)   ' limits the max radius to half the height
  mBackgroundColor = BackgroundColor
  xParent = Parent

  ' initialize the mBase panel and add it onto the parent view
  xBase = xui.CreatePanel("")
  xParent.AddView(xBase, Left, Top, Width, Height)

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

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

Example:

```
  'adds a second xLimitBar in the code
  ltbTest1.Initialize(Me, "ltbTest1")
  ltbTest1.FrontLineColor = xui.Color_Blue
  lblLimitLeft.Text = ltbTest.LimitLeft
  lblLimitRight.Text = ltbTest.LimitRight
  ltbTest1.AddToParent(Root, 40dip, 120dip, Root.Width - 80dip, 40dip, xui.Color_Red,
10dip)
```

6.8 InitClass routine

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

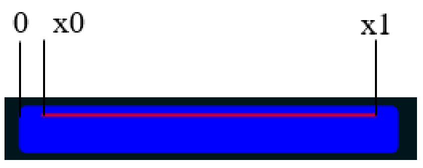
```
Private Sub InitClass
  ' initialize the background canvas and draw the background line
  cvsPanelBack.Initialize(mBase)

  ' initialize the foreground panel and canvas
  ltbPanelFront = xui.CreatePanel("ltbPanelFront")
  mBase.AddView(ltbPanelFront, 0, 0, mWidth, mHeight)
  cvsPanelFront.Initialize(ltbPanelFront)

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

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

  DrawBackGround
End Sub
```

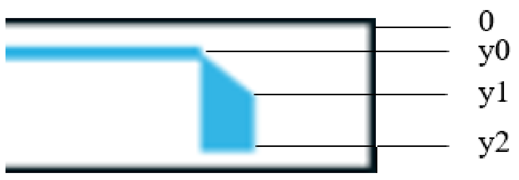
The code is self-explanatory.

6.9 InitCursors routine

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

```
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
```





6.10 Draw the background and background line

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

```
Private Sub DrawBackGround
   ' set the background color and the radius for the background panel
   cvsPanelBack.ClearRect(rectPanelFront)     'needed to have the round corners when the width is decreased
   BackgroundPath.InitializeRoundedRect(rectPanelFront, mRadius)
   cvsPanelBack.ClipPath(BackgroundPath)
   cvsPanelBack.DrawRect(rectPanelFront, mBackgroundColor, True, 1dip)

   'draw the background line
   cvsPanelBack.DrawLine(x0, y0, x1, y0, mBackLineColor, 2dip)
   cvsPanelBack.RemoveClip
   cvsPanelBack.Invalidate
End Sub
```

We draw the background with: `cvsPanelBack.DrawRect`.
And the background line, with: `cvsPanelBack.DrawLine`

6.11 DrawCursors routine

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

```
Private Sub DrawCursors
   ' draw a transparent rectangle to erase the foreground panel
   cvsPanelFront.ClearRect(rectPanelFront)

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

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

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

   cvsPanelFront.Invalidate
End Sub
```

We:

- Erase the whole foreground panel with `ClearRect`.
- Define both cursors according to the current position.
  The cursor shapes are defined with two Paths.
- Draw the cursors.
- Draw the foreground line.

6.12 Cursor moving

To detect cursor moves we use the touch event of the foreground panel:

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

  ' select the Action type
  Select Action
    Case ltbPanelFront.TOUCH_ACTION_DOWN
      If xx < 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((xx - x0) / mScale + .5)
      PositionPixels(PosIndex) = xx
      DrawCursors
    Case ltbPanelFront.TOUCH_ACTION_MOVE
      If xui.SubExists(mCallback, mEventName & "_ValuesChanged", 2) Then
        CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
      End If

      mLimit(PosIndex) = Floor((xx - x0) / mScale + .5)
      PositionPixels(PosIndex) = xx
      DrawCursors
    Case ltbPanelFront.TOUCH_ACTION_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

      'call the ValuesChanged routine if it exists
      If xui.SubExists(mCallback, mEventName & "_ValuesChanged", 2) Then
        CallSub3(mCallback, mEventName & "_ValuesChanged", mLimit(0), mLimit(1))
      End If
  End Select
End Sub
```

6.13 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
Public Sub setMax(MaxValue As Int)
   mMaxValue = MaxValue
   InitCursors
   DrawCursors
End Sub

Public Sub getMax As Int
   Return mMaxValue
End Sub
```

The LimitLeft property:

```
'gets or sets the left limit
Public Sub setLimitLeft(Pos As Int)
   ' if Pos is lower than 0 set cLimitLeft to 0
   mLimit(0) = Max(0, Pos)
   InitCursors
   DrawCursors
End Sub

'gets or sets the left limit
Public Sub setLimitLeft(Pos As Int)
       ' if Pos is lower than 0 set cLimitLeft to 0
       mLimit(0) = Max(0, Pos)
       InitCursors
       If ltbPanelFront.IsInitialized Then
              DrawCursors
       End If
End Sub

Public Sub getLimitLeft As Int
       Return mLimit(0)
End Sub

Public Sub getLimitRight As Int
   Return mLimit(1)
End Sub
```

The Visible property:

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

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

The Width property:

```
'gets or sets the Width property
Public Sub setWidth(Width As Int)
  mWidth = Width

  ' set the new widths
  mBase.Width = mWidth
  ltbPanelFront.Width = mWidth

  ' resize the two Canvases
  cvsPanelBack.Resize(mWidth, mHeight)
  cvsPanelFront.Resize(mWidth, mHeight)

  ' adjust the width of rectPanelFront
  rectPanelFront.Width = mWidth

  If ltbPanelFront.IsInitialized Then
      InitCursors
      DrawBackGround
      DrawCursors
  End If
End Sub

Public Sub getWidth As Int
  Return mWidth
End Sub
```

In this routine, as the width of the CustomView has changed, we need to:

- set the Width of all three B4XPanels `xBase`, `ltbPanelBack` and `ltbPanelFront`.
- resize the two B4XCanvases `cvsPanelBack` and `cvsPanelFront`.
- set the Width of the B4XRect `rectPanelFront`.
- and `InitCursors`, initialize the cursors.
- `DrawBackGround` draw the background
- `DrawCursors` draw the cursors.

We use this test `If ltbPanelFront.IsInitialized Then` to avoid an error when a property is set in the code before it is added to a parent view in the code.

The Height property routine is similar to the Width property routine.

There are other properties not explained here.

6.14 Make a B4X Library

We make a B4X Library for the xLimitBar CustomView.

We generate the manifest file, it's a text file with the content below,

```
Version=1.0
Author=Klaus CHRISTL (klaus)
B4J.DependsOn=jXUI
B4A.DependsOn=XUI
B4i.DependsOn=iXUI
```

`Version`:	the version number
`Author`:	the author's name
`B4J.DependsOn`:	the list of all B4J libraries the custom view depends on.
`B4A.DependsOn`:	the list of all B4A libraries the custom view depends on.
`B4i.DependsOn`:	the list of all B4iJ libraries the custom view depends on.

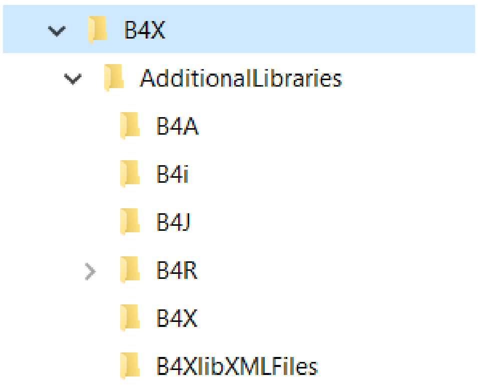
And save it with the name: *manifest.txt*.

In our case xLimitBar depends only on the xui libraries.

Then we zip the manifest.txt file and the xLimitBar.bas file to generate the xLimitBar.b4xlib file.
The extension must be *b4xlib*.

Copy this file into the \AdditionalLibraries\B4X folder.

Remember the subfolder structure of the AdditionalLibraries folder.



More information in chapter Generate a B4XLibrary.