B4A Scrolling Grid using CustomListViews and a Horizontal ScrollView

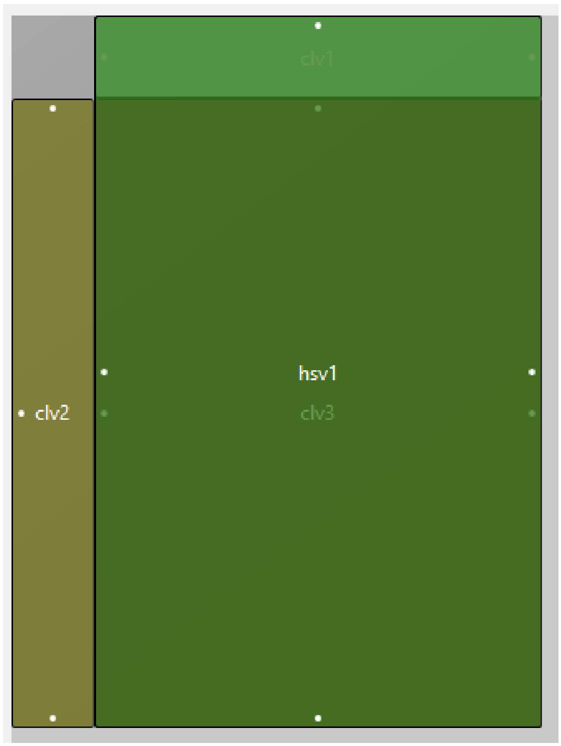
Have you ever wondered how you can make a CustomListView scroll horizontally?

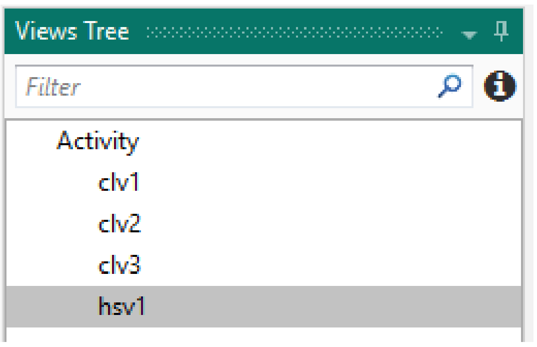
Well, in this small tutorial you can read all about it.

Start the B4A IDE environment and make a new project (possible name: scroll_grid) and select B4Xpages. Set in the Main tab the application label to Scroll Grid and in the project Build Configurations give the package a good name: b4a.scroll_grid.

Click on the B4XMainPage tab in the IDE.

Let's start by making a layout. Go to the designer and add 3 CustomListViews (XUI Views library!). And below the CustomListViews 1 and 3 add a Horizontal ScrollView that has the same size as the 2 CustomListViews. Give the CustomListViews a short name: clv1, clv2, clv3 and name the HorizontalScrollView: hsv1. Make sure that the hsv1 is the last in the list of views (see Views Tree) so it will not cover the clv1 and clv3.





Activity
clv1
clv2
clv3
hsv1

clv1: left 50, height 50, horizontal anchors both sides

clv2: top 50, width 50, vertical anchors both sides

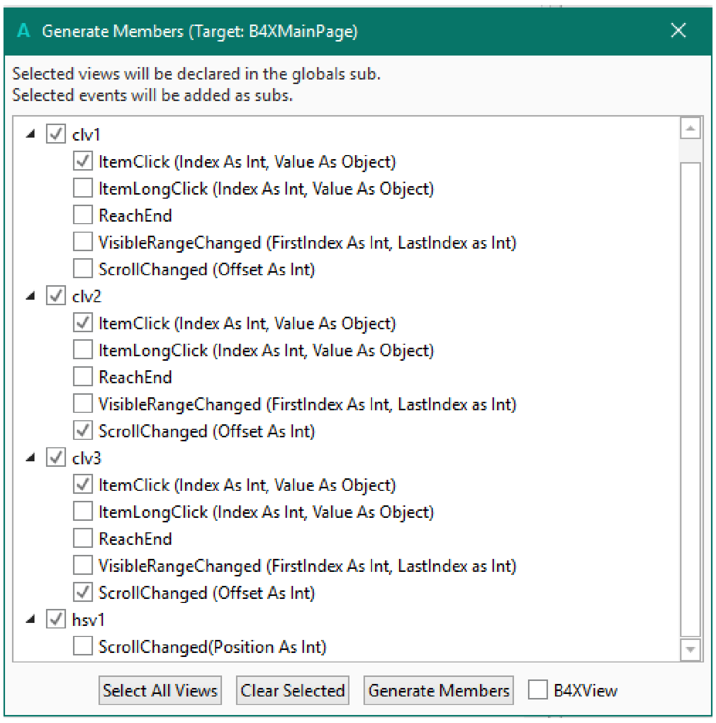
clv3: left 50, top 50, anchors on all sides

hsv1: left 50, anchors on all sides

Now let's generate the members.

Check the following:

- ✓ clv1 and its ItemClick event
- ✓ clv2 and its ItemClick and ScrollChanged events
- ✓ clv3 and its ItemClick and ScrollChanged events
- ✓ hsv1



Generate Members (Target: B4XMainPage)

Selected views will be declared in the globals sub.
Selected events will be added as subs.

- ✓ clv1
  - ✓ ItemClick (Index As Int, Value As Object)
  - ☐ ItemLongClick (Index As Int, Value As Object)
  - ☐ ReachEnd
  - ☐ VisibleRangeChanged (FirstIndex As Int, LastIndex as Int)
  - ☐ ScrollChanged (Offset As Int)
- ✓ clv2
  - ✓ ItemClick (Index As Int, Value As Object)
  - ☐ ItemLongClick (Index As Int, Value As Object)
  - ☐ ReachEnd
  - ☐ VisibleRangeChanged (FirstIndex As Int, LastIndex as Int)
  - ✓ ScrollChanged (Offset As Int)
- ✓ clv3
  - ✓ ItemClick (Index As Int, Value As Object)
  - ☐ ItemLongClick (Index As Int, Value As Object)
  - ☐ ReachEnd
  - ☐ VisibleRangeChanged (FirstIndex As Int, LastIndex as Int)
  - ✓ ScrollChanged (Offset As Int)
- ✓ hsv1
  - ☐ ScrollChanged(Position As Int)

And then it is time to starting coding.

Add the following declarations of variables to the Class_Globals subroutine:

```
Private reclst As List
Private delim As String
Private devheight As Int
Private totalwidth As Int
Private totalheight As Int
Private numrows As Int
Private numcols As Int
```

The generated variables are already there, right?

In the B4XPage_Created subroutine we do some initialization work.

```
reclst.Initialize
delim = ";"
devheight = GetDeviceLayoutValues.height - 40dip
```

The reclst list variable will be used to store the rows in. Each row contains a string of the

columns data separated by a delimiter (variable delim), a semi-colon in this case ';').
We will use the devheight variable to adjust the clv2 and clv3 scrolling height.

The subroutines fill_reclst and add_row_numbers make the list ready for use in the grid.

```
        fill_reclst
        add_row_numbers
```

Then we make some more calculations needed in the other subroutines:

```
        Dim rec As String = reclst.Get(0)
        Dim columns() As String = Regex.Split(delim,rec)
        numrows = reclst.Size
        numcols = columns.Length
        totalwidth = numcols*100dip
        totalheight = numrows*60dip
```

We get the first record from the list and create an array for the columns.
The number of rows is equal to the size of the reclst.
The number of columns is equal to the lenght of the columns array.
The totalwidth is equal to the number of columns times the width of a column. In this example we use a fixed 100 dip (display independant pixels).
The totalheight uses (fixed) 60 dip times the number of rows.

And then it is time to fill the clvs with the calls to each clv:

```
        fill_clv1
        fill_clv2
        fill_clv3
```

The first subroutine is the fill_reclst. In this routine the list is filled using 2 For loops.
The comment line already reveals that you can use this app to load the contents of a csv (comma separated values) file.

```
Private Sub fill_reclst
'       reclst = File.ReadList(File.DirAssets,"persons.csv")
        For row = 0 To 19
                Dim colstr As String = ""
                For col = 0 To 9
                        If col < 9 Then
                                colstr = colstr & "row " & row & " col " & col & delim
                        Else
                                colstr = colstr & "row " & row & " col " & col
                        End If
                Next
                reclst.Add(colstr)
        Next
        Log(reclst)
End Sub
```

In the add_row_numbers we add a row number to each row in the reclst list. These row numbers will be displayed in the clv2 listview. We replace the record by removing and inserting it at the given index.

```
private Sub add_row_numbers
        For i = 0 To reclst.Size -1
                Dim rrec As String = reclst.get(i)
                rrec = i & delim & rrec
                reclst.RemoveAt(i)
                reclst.InsertAt(i,rrec)
        Next
End Sub
```

The first of 3 fill subroutines fill_clv1 will put the column headers in the clv1 listview. The width of the clv1 is set using the Base_Resize method. Each panel contains a label and has a event tag set to “lblhead”. The clv1 listview is added to the HorizontalScrollView hsv1.

```
Private Sub fill_clv1
        clv1.Base_Resize(totalwidth,totalheight)
        clv1.sv.Height = 50dip
        clv1.Clear
        Dim rec As String = reclst.get(0)
        rec = rec.SubString(rec.IndexOf(delim)+1)           ' not show the first column
        Dim pnl As Panel = set_item(rec,0,"lblhead")
        clv1.Add(pnl,rec)
        clv1.sv.RemoveViewFromParent
        hsv1.Panel.AddView(clv1.sv,0dip,0dip,clv1.sv.Width,clv1.sv.Height)
End Sub
```

The CustomListView clv2 shows the rownumbers. Here the panel has a “lblrow” event tag. Clicking on a row panel triggers this event. The height of the device is used to set the scrolling height. The 110 dip represent the height of the actionbar and the height of the clv1 list.

```
Private Sub fill_clv2
        clv2.Base_Resize(50dip,totalheight)
        clv2.sv.Height = devheight - 110dip
        clv2.Clear
        Dim rec As String
        For i = 1 To reclst.Size -1                         ' row 0 contains the header columns
                rec = reclst.Get(i)
                Dim fields() As String = Regex.Split(delim,rec)
                Dim pnl As Panel = set_item(rec,i,"lblrow")
                clv2.Add(pnl,fields(0))
        Next
End Sub
```

The fill_clv3 subroutine is used to show the data. The first column (rownumber) and the first row (header row) are not shown.

```
Private Sub fill_clv3
	clv3.Base_Resize(totalwidth,totalheight)	'totalwidth-100dip first column not show
	clv3.sv.Height = devheight - 110dip
	clv3.Clear
	For i = 1 To reclst.Size -1
		Dim rec As String = reclst.Get(i)
		rec = rec.SubString(rec.IndexOf(delim)+1)  ' not show the first column
		'clv3.AddTextItem(rec,rec)
		Dim pnl As Panel = set_item(rec,i,"lbldata")
		clv3.Add(pnl,rec)
	Next
	clv3.sv.RemoveViewFromParent
	hsv1.Panel.AddView(clv3.sv,0dip,53dip,clv3.sv.Width,clv3.sv.Height)
	hsv1.Panel.Width = totalwidth				'totalwidth-100dip first column not show
End Sub
```

The set_item subroutine is called from the 3 fill_clv subroutines. It assembles a panel with labels (for each column).

```
Private Sub set_item(rec As String, irow As Int,event As String) As Panel
	Dim clr As Int = Colors.LightGray
	Dim columns() As String = Regex.Split(delim,rec)
	Dim pnl As Panel
	pnl.Initialize("")
	If irow Mod 2 <> 0 Then
		clr = Colors.White
	End If
	For x = 0 To columns.Length -1
		Dim lbl As Label
		lbl.Initialize(event)
		lbl.Tag = irow & delim & x
		lbl.Text = columns(x)
		lbl.Padding = Array As Int (3dip, 1dip, 3dip, 1dip)
		lbl.Color = clr
		pnl.AddView(lbl,0dip+(x*100dip),0dip,100dip-3dip,60dip)
	Next
	pnl.SetLayout(0dip,0dip,(x+1)*100dip,62dip)
	Return pnl
End Sub
```

The ScrollChanged subroutines synchronise the 2 CustomListViews: clv2 (row numbers) and clv3 (data).

```
Private Sub clv3_ScrollChanged (Offset As Int)
	clv2.sv.ScrollViewOffsetY = Offset
End Sub
```

```
Private Sub clv2_ScrollChanged (Offset As Int)
	clv3.sv.ScrollViewOffsetY = Offset
End Sub
```

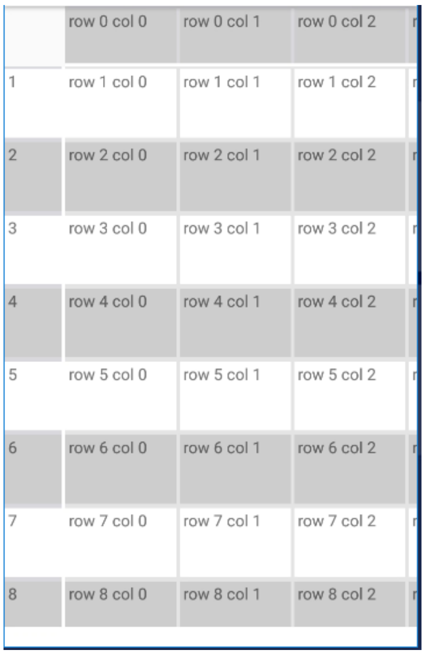
The ItemClick subroutines can be used to get some specific information from the clvs. If you swipe from right to left to the last column then you can tap on the empty space behind each row. This will trigger the clv1_ItemClick or clv3_ItemClick subroutine.

```
Private Sub clv1_ItemClick (Index As Int, Value As Object)
	Log("clv1 itemclick: " & Value)
	xui.MsgboxAsync(Value,"clv1 itemclick")
End Sub
Private Sub clv2_ItemClick (Index As Int, Value As Object)
	Log("clv2 itemclick: " & Value)
	xui.MsgboxAsync(Value,"clv2 itemclick")
End Sub
Private Sub clv3_ItemClick (Index As Int, Value As Object)
	Log("clv3 itemclick: " & Value)
	xui.MsgboxAsync(Value,"clv3 itemclick")
End Sub
```

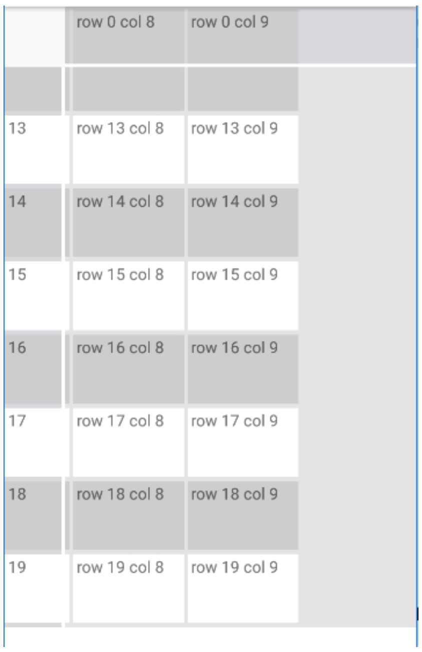
When the event tags are set then these subroutines are triggered by tapping on a panel. Notice that you can get the complete record by tapping on the row number.

```
private Sub lblhead_Click			' clv1
	Dim lbl As Label = Sender
	Dim tag As String = lbl.Tag
	Log("lblhead click: " & tag)
	xui.MsgboxAsync(tag,"lblhead click")
End Sub
private Sub lblrow_Click			' clv2
	Dim lbl As Label = Sender
	Dim tag As String = lbl.Tag
	Log("lblrow click: " & tag)
	xui.MsgboxAsync(tag,"lblrow click")
	Dim row As Int = tag.SubString2(0,tag.IndexOf(delim))
	xui.MsgboxAsync(reclst.Get(row),"lblrow click record")
End Sub
private Sub lbldata_Click			' clv3
	Dim lbl As Label = Sender
	Dim tag As String = lbl.Tag
	Log("lbldata click: " & tag)
	xui.MsgboxAsync(tag,"lbldata click")
End Sub
```

So there you have it, a scrolling grid with a fixed first column and a fixed header row.

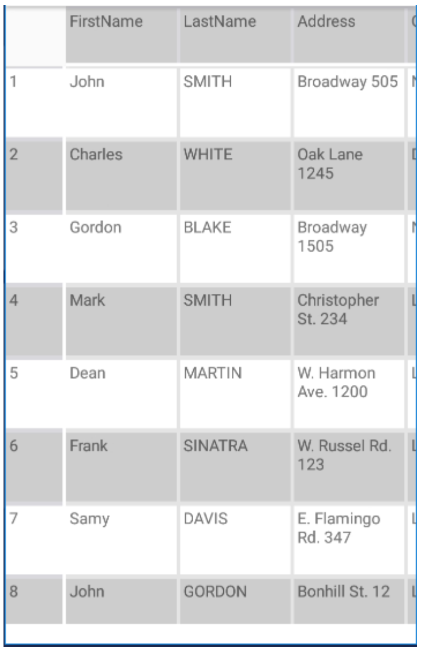


	row 0 col 0	row 0 col 1	row 0 col 2	r
1	row 1 col 0	row 1 col 1	row 1 col 2	r
2	row 2 col 0	row 2 col 1	row 2 col 2	r
3	row 3 col 0	row 3 col 1	row 3 col 2	r
4	row 4 col 0	row 4 col 1	row 4 col 2	r
5	row 5 col 0	row 5 col 1	row 5 col 2	r
6	row 6 col 0	row 6 col 1	row 6 col 2	r
7	row 7 col 0	row 7 col 1	row 7 col 2	r
8	row 8 col 0	row 8 col 1	row 8 col 2	r

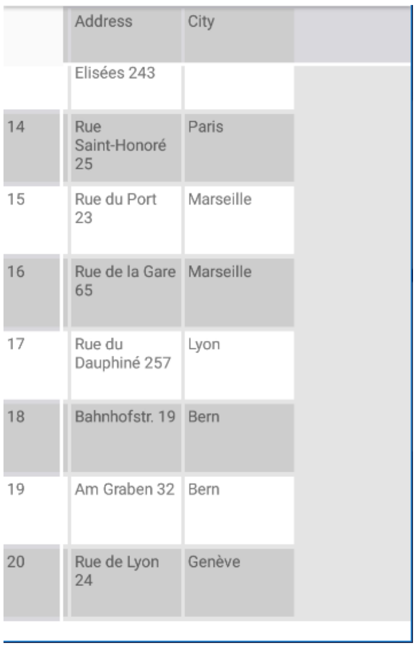


	row 0 col 8	row 0 col 9
13	row 13 col 8	row 13 col 9
14	row 14 col 8	row 14 col 9
15	row 15 col 8	row 15 col 9
16	row 16 col 8	row 16 col 9
17	row 17 col 8	row 17 col 9
18	row 18 col 8	row 18 col 9
19	row 19 col 8	row 19 col 9

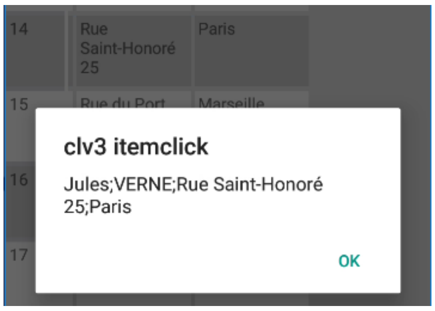
As mentioned earlier you can use this grid to display CSV-files. It could look like this:



	FirstName	LastName	Address	C
1	John	SMITH	Broadway 505	N
2	Charles	WHITE	Oak Lane 1245	D
3	Gordon	BLAKE	Broadway 1505	N
4	Mark	SMITH	Christopher St. 234	L
5	Dean	MARTIN	W. Harmon Ave. 1200	L
6	Frank	SINATRA	W. Russel Rd. 123	L
7	Samy	DAVIS	E. Flamingo Rd. 347	L
8	John	GORDON	Bonhill St. 12	L



	Address	City
	Elisées 243	
14	Rue Saint-Honoré 25	Paris
15	Rue du Port 23	Marseille
16	Rue de la Gare 65	Marseille
17	Rue du Dauphiné 257	Lyon
18	Bahnhofstr. 19	Bern
19	Am Graben 32	Bern
20	Rue de Lyon 24	Genève



14 Rue Saint-Honoré 25 Paris

15 Rue du Port Marseille

16

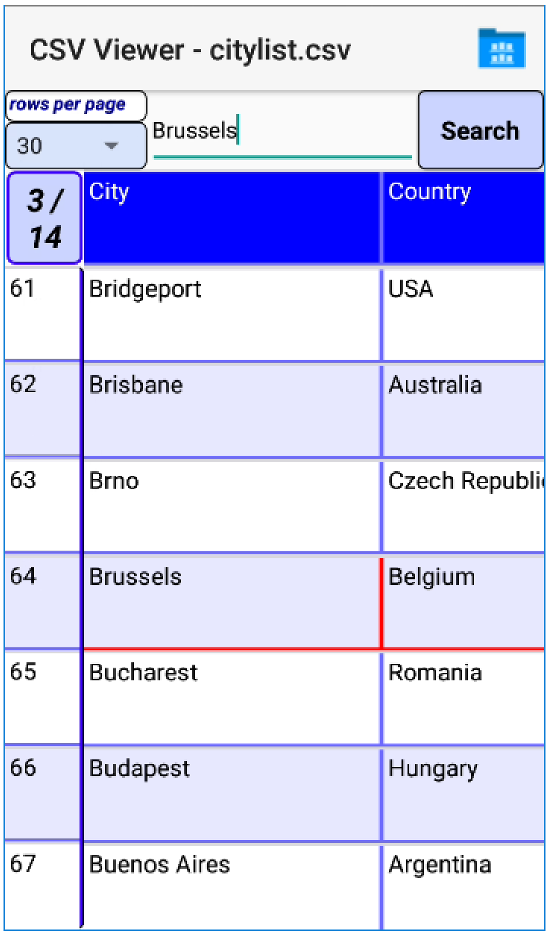
17

clv3 itemclick

Jules;VERNE;Rue Saint-Honoré 25;Paris

This grid will work fine with a small number of rows and columns. 20 rows with 4 columns result in a total of 80 panels and 80 labels. But if you use 500 rows and 10 columns then the result will be 5000 panels and 5000 labels. The memory of your smartphone is limited and the app will stop working if the limit is reached.

You can avoid this by limiting the scrolling to a workable, practical size. The use of pages will then make the data still accessible like in this example:



3 / 14	City	Country
61	Bridgeport	USA
62	Brisbane	Australia
63	Brno	Czech Republi
64	Brussels	Belgium
65	Bucharest	Romania
66	Budapest	Hungary
67	Buenos Aires	Argentina

Happy coding!