

DYNAMIC VIEWS (How to...)

PART 1 : Creating Dynamic elements (I.e., create and delete views using code)

1) To use dynamic views in B4A you must first define the type of item and then initialize the View for the selected type. (Initialization is not required if you use the B4A Designer)

Suppose to **create a Label** called **DynLab** and write:

Dim DynLab as Label

DynLab.Initialize("Label")

Once this is done, using the following line of code, you create a graphical representation of the element

Activity.AddView(DynLab, Left, Top, Width, Height)

The desired object is created, without triggering any file layout, as its source is the code.

To complete the item as you would, you need to add the members to what is, for the moment, only part of the Activity (which is the Parent)

You must define the members applicable to the dynamic view (you have them also listed into Rem B4a for each view .

To do this you just need to write a line of code, which is equivalent to selecting an entry or assign a value into the Designer.

Here are a few examples.

DynLab.TextSize = 14

DynLab.TextColor = Colors.Blue

DynLab.Color = Colors.White

DynLab.Visible = True

.....and so on.

Remember that the Characteristics of the Text use different syntaxes like Gravity and Typeface, see the following example:

DynLab.Typeface = Typeface.DEFAULT_BOLD

DynLab.Gravity = Gravity.CENTER_VERTICAL

DynLab.Gravity = Gravity.CENTER_HORIZONTAL

For the Events, the dynamic view offers the ones of the core view to which you refer in the Dim declaration.

2) The previous paragraph shows how to create a single dynamic view of any type (**Button, Label, Panel, EditText** etc)

However, the maximum value found using the multiple views organized as lists (horizontal or vertical)

In fact, using a simple **For / Next loop** connected to the change of the variable Left or Top you can get an array whose elements are the dynamic created view.

The code probably says it best:

For x = 1 To 3

Dim DynLab As Label

DynLab.Initialize("Label")

DynLab.Visible = True

DynLab.Color = Colors.White

```
DynLab.TextColor = Colors.Black
```

.....

```
DynLab.Tag = x
```

```
Activity.AddView(DynLab, (Left + (x-1) * (Width+D)), Top, Width, Height) 'Horizontal Alignment
```

-or-

```
Activity.AddView(DynLab, Left, (Top + (x-1) * (Height+D)), Width, Height) 'Vertical Alignment
```

```
Next
```

Note: **D** is a variable to define distance between the labels

A list like this is not indexed (despite being created through a progressive index)

So to turn on or read an item is used the following system:

after having initialized the dynamic view, with Senders, the label Click on the desired dynamics view is reported to the code contained in the Sub .

Here is the code:

```
Sub Label_Click
```

```
Dim DynLab As Label
```

```
DynLab.Initialize("Label")
```

```
DynLab = Sender
```

```
k = DynLab.Tag
```

```
LL = DynLab.Left
```

```
End Sub
```

The code also provides a system for indexing indirectly the elements of the dynamic list.

It is the use of the **member Tag**.

In the code used to create the DynLab you can note that **the For/Next loop attaches to tag each label the value of its index**.

So when you press one of the Sender label there is the index of the item but you can still detect it by reading the tags

In the example sub the last instruction reads the position Left of the label (Sender) clicked

3) So far we have created a dynamic single label and a list of labels.

This methods are generally combined with a layout that contains other elements of a program to add some dynamic items ..

But we can also create multiple dynamic objects (even of the same type but with different uses) and use them to make a program without Layout.

We still use the Labels to show text, to use as buttons and to form a list of labels.

The reference object is the same for all 3 types ("Label") but use them separately is a need

I got it declaring different names but they all refer to the Label view.

To clarify, I used a code as follows:

```
Global
```

```
Dim DynLab as Label
```

```
Dim LabelA as Label
```

```
Dim LabelB as Label
```

e così via per ogni altro elemento label presente.

Quanto sopra permette di diversificare gli eventi Click utilizzando:

```
Sub Label_Click
```

```
DynLab.initialize("Label")
```

```
DynLab = Sender
```

```
.....  
End Sub
```

```
Sub LabelA_Click  
DynLabA.initialize("LabelA")  
DynLabA = Sender
```

```
.....  
End Sub
```

```
Sub LabelB_Click  
DynLabB.Initialize("LabelB")  
DynLabB = Sender
```

```
.....  
End Sub
```

For the "creation" remains only to remember the code used to populate the lists of label that allows you to do so simultaneously with the creation of the list.

The system uses a string where the contents of the labels is shown separating each element with a comma.

Here is an example : **Lista = "First,Second,Third,.....,Last,"** of the string that the following code uses to populate the label of a list.

```
Tx = (Lista).IndexOf(",")  
DynLab.Text = Lista.SubString2(0, Tx)  
Nlist = Lista.SubString2(Tx+1, Lista.Length) '(remaining Lista string)  
Lista = Nlist
```

Remember that the above code must be used only if you create a Dynamic List of view that need to contain text for each item.

In this case the code must be inserted into the Loop For/Next, after the views creation code, at the end of the Loop

PART 2 : Adapting & Deleting Dynamic elements

1) In part 1 of these notes was dealt with how to create dynamic views in B4A.

There is also a different aspect of the creation of dynamic views that regards the adaptation to different types and sizes of the screen

The reference procedure is the '**AutoScaleAll**' script to fit the layout, but dynamic objects, arise directly from the code.

For this reason, their dimensions are present only in the line code Activity.AddView (.....) as we have seen.

Abandon the use of AutoScale would not be convenient and for this reason the Layout abandoned for the use of dynamic objects is called up into service.

It 'simply create, using the Designer, a main.bal that contains only one element. A Label.

The size of the label used (we'll call **LG i.e. Label Guide**) must be chosen so as to create for multiple and/or parts of them all required measures.

This will make it possible to activate the script AutoScaleAll applying it to dynamic views.

An example:

If the **LG** Label into a program, made entirely through code, has the size of **8x100 (Height x Width)**

Thus the **height of a menu** is **(LG.Height * 2)** -

The **width of the menu** is equal to **LG.Width** -

while the width of the WebView that displays the selected data appears to be **LG.Width * 3**.

Also do not forget that **even LG.TextSize can be used as a parameter**.

At this point, with a simple single object in Layout, we are able to resize all dynamic objects used in a program

It is obvious that, if your program already has a layout with a variety of views necessary measures can be invoked by the objects existing in it without needing Label Guide.

There remains one last problem to solve.

Although the layout is used to provide the script AutoScaleAll compatible measures is impossible to use the UI.Cloud Designer to see how our program can be adapted to different screens.

To solve it you just have to create a Layout "**fakemain**" where the designers are part of the same views in the project that we have created through code.

Without this, just save it and send it to the UI Cloud to graphically see how the adjustments to the parameters applied AutoScaleAll.

Obviously you will never use, in reality, the layout "fakemain" which will remain a version control that otherwise would cause the program to lock.

But to use the dynamic views is necessary not only to create them and make them adaptable but also, and above all to be able to delete them in the correct manner.

We can say that it is more difficult to delete a dynamic object than create it.

Consider that, whatever the system with which you add the view to a Activity these are identified by an index.

A member of the Activity, **NumberOfView (Read only)** gives the number of the views contained in what is usually the main Parent.

When using the designer numbering index remains unchanged, except if you use the instructions BringToFront or SendToBack.

But in this case the order only is altered but not the total index (ie the number of view).

Then the index is incremented after each addition of a view and, once you have created your optimal layout, index remains unchanged

The problem does not arise if, after you create a view with the code, the same will remain in effect forever, until closing the program or if the dynamic view is created and, after use, immediately cleared.

The thing is a bit 'different when using dynamic views along with a layout created with the designer and they are created and deleted as needed.

In this case the number of view present in an Activity becomes a variable to be evaluated equally dynamically.

In this light fits the suggestion to add dynamic objects only after those created by the designer.

In fact, the only statement applicable to delete dynamic objects is widely RemoveViewAt (index)
So adding at the end the dynamic objects you will have to check only the final value of the index
to delete the added dynamic views as the fixed views remain unchanged.

To do this, I used this code:

```
qq = Activity.NumberOfViews: BV = qq
```

This line of code reads the **Quantity (qq)** of view in the Activity and the value it assigns to the variable **BV**.

If placed at the beginning of the routine in our program before creating dynamic view, this code will provide us with the value of **BV (Basic View)**, which represents the number of standard view (not dynamic) already present.

When this is done when the program starts the same line of code, placed at the end of the routine of creating a dynamic object will give us the current value of the index will be assigned to a different variable as you can see.

```
qq = Activity.NumberOfViews: LN = qq-1
```

Once you have obtained these values can be passed to the Sub cancellation as follows:

```
Sub ClearList
```

```
qq = Activity.NumberOfViews: LN = qq-1
```

```
For m = LN To Step -1 BV
```

```
Activity.RemoveViewAt (m)
```

```
Next
```

```
end Sub
```

You can see that the line of code that detects the number of active views is inserted at the beginning of the cancellation Sub.

This ensures that the measured value is as current as possible before you start erasing.

This Sub is born for the cancellation of lists of dynamic views but can also operate to erase a single object.

However, when you create a single object it is possible to not use the above procedures and use the instruction RemoveView.

It may be the case of an ImageView dynamically create and removed immediately after seeing of the picture.

I apologize for 2 things.

The first is my poor english.

The second: to want to teach to you something .

In fact i am a beginner, if not for the basic, certainly for Android and B4A and therefore are not able to teach a lot,

I just compiled these notes to determine the experments done that surely work, since you are using the program built according to the principles explained.

Have fun.