Using source code templates in programming.

Introduction

The word “template” has different meanings according to the context in which it is used.

In a Microsoft Word application you can use a template as a blue-print for forms, letters, mailings and so on. The template typically has some fields that can be filled in when a document is created that is based on the template. The layout is usually preset.

In the programming languages a template can be used to reuse code. With the use of a templating “engine” (a class) the source code is generated. The templates can be build using the specific syntax from the templating engine.

In this text you can learn how to make a template and generate the source code from it.

Make a template

You can use or copy an existing piece of source code and place some placeholders in it. Before generating the result code you give these placeholders a value.

A placeholder is surrounded by special markings like for instance {{...}} or {%...%} or in my case [@...]. Inside these markings you place a (variable) name or a reference.

An example:

```
Sub load_list
    Private lst As List
    lst.Initialize
    If File.Exists(xui.DefaultFolder,"[@filename]") Then
        lst = File.ReadList(xui.DefaultFolder," [@filename]")
        For i = 0 To lst.Size-1
            [@clv_var].AddTextItem(lst.Get(i), lst.Get(i))
        Next
    End If
End Sub
```

In this example the [@filename] placeholder refers to the filename being used for instance "categorie.txt" and the [@clv_var] refers to the name of the CustomListView for instance clv1.

If you want to translate the texts of a application (titles, label texts, button texts, messages, …) you can put placeholders for the texts that need to translated.

A templating engine also supports loops like for instance {for each …}…{/for each} or {% for … %}…{% endfor %} or in my case [@loop1]…[/@loop1].

These changes can be done with an editor like Notepad++ or with a special tool (see below).

The templating engine that is used in the tool also supports 1 level of nesting template files. An example:

clv_page_editable_with_textitems.tpl:

[@B4J_header.tpl]
[@clv_page_declarations.tpl]
[@clv_editable_with_textitems.tpl]
[@clv_load_list_with_textitems_from_file.tpl]
[@clv_save_list_with_textitems_to_file.tpl]

Fill in the placeholders

Before you can generate the code from the template you need to fill the placeholders with a value.

In PHP you would write a script that gets all the information needed by the template and pass that information on as arguments for the templating engine.

The B4J tool uses subroutines to fil a map with the necessary values. That map is then used in the generating process. The template class provides a method called find_placeholders to provide the map with the necessary keys (the placeholders).

Generate the resulting code

Now it's time for the templating engine to do its thing.

The template text and the values map are used in the template class methods: replace_placeholders, process_level1_template, process_loops.

The result of the generating process is a *.bas file. This file (if it is complete) can be added in the IDE as an existing module.

A few examples:

The load_list subroutine (see above) when generated looks like this:

```
Sub load_list
	Private lst As List
	lst.Initialize
	If File.Exists(xui.DefaultFolder,"categorie.txt") Then
		lst = File.ReadList(xui.DefaultFolder,"categorie.txt")
		For i = 0 To lst.Size-1
			clv1.AddTextItem(lst.Get(i), lst.Get(i))
		Next
	End If
End Sub
```

The clv_page_editable_with_textitems page when generated looks like this:

```
B4J=True
Group=Default Group
ModulesStructureVersion=1
Type=Class
Version=9.8
@EndOfDesignText@
Sub Class_Globals
	' template declarations.tpl
	Private Root As B4XView 'ignore
	Private xui As XUI 'ignore
	Private clv1 As CustomListView
	Private tf1 As TextField
	Private btn1 As Button
	Private clvindex As Int
End Sub
Public Sub Initialize As Object
	Return Me
End Sub
Private Sub B4XPage_Created (Root1 As B4XView)
	Root = Root1
	Root.LoadLayout("category_layout")
	B4XPages.SetTitle(Me,"Voorbeeld01 - Categorie")
	xui.SetDataFolder("Voorbeeld01")
End Sub
Private Sub B4XPage_Appear
	btn1.Text = "Voegtoe"
	clv1.Clear
	clvindex = 0
	tf1.Text = ""
	load_list
End Sub
Private Sub B4XPage_Disappear
	If clv1.Size > 0 Then
		save_list
	End If
End Sub
Sub clv1_ItemClick (Index As Int, Value As Object)
	tf1.Text = Value
	btn1.Text = "Wijzig"
	clvindex = Index
End Sub
Private Sub clv1_ItemLongClick (Index As Int, Value As Object)   ' B4J = right click
	tf1.Text = Value
	btn1.Text = "Verwijder"
	clvindex = Index
End Sub
```

```
Private Sub btn1_Click
	Select btn1.Text
		Case "Voegtoe"
			clv1.AddTextItem(tf1.text,tf1.text)
		Case "Wijzig"
			clv1.RemoveAt(clvindex)
			clv1.InsertAtTextItem(clvindex,tf1.text,tf1.text)
		Case "Verwijder"
			Log(clvindex)
			clv1.RemoveAt(clvindex)
			If clv1.Size < 1 Then
				File.Delete(xui.DefaultFolder,"categorie.txt")
			End If
	End Select
	If clv1.Size > 0 Then
		save_list
	End If
	B4XPage_Appear
End Sub
Sub load_list
	Private lst As List
	lst.Initialize
	If File.Exists(xui.DefaultFolder,"categorie.txt") Then
		lst = File.ReadList(xui.DefaultFolder,"categorie.txt")
		For i = 0 To lst.Size-1
			clv1.AddTextItem(lst.Get(i), lst.Get(i))
		Next
	End If
End Sub
Sub save_list
	Private lst As List
	lst.Initialize
	For i = 0 To clv1.Size-1
		lst.Add(clv1.GetValue(i))
	Next
	File.Delete(xui.DefaultFolder,"categorie.txt")
	If lst.Size > 0 Then
		File.WriteList(xui.DefaultFolder,"categorie.txt",lst)
	End If
End Sub
```

Notice that the texts were translated into Dutch. The header template information is necessary to add the page to the IDE.

You can find more examples in the zip-file provided with this text.

Library references

When you add the generated code to the IDE project you will probably get some errors about the libraries that are used.

In the project source template you can provide the information about the libraries.

An example called example01.b4j.tpl (the top part):

```
AppType=JavaFX
Build1=Default,b4j.example
[@loop1][@file_var]=[@layout_filename]
[/@loop1][@loop2][@filegroup_var]=[@group_name]
[/@loop2]Group=Default Group
[@loop3][@library_var]=[@library_name]
[/@loop3][@loop4][@module_var]=[@module_path]
[/@loop4]NumberOfFiles=[@number_of_files]
NumberOfLibraries=[@number_of_libraries]
NumberOfModules=[@number_of_modules]
Version=9.8
@EndOfDesignText@
#Region Project Attributes
	#MainFormWidth: 600
	#MainFormHeight: 600
#End Region
' from template example01.b4j.tpl
Sub Process_Globals
	Private fx As JFX
	Private MainForm As Form
End Sub

Sub AppStart (Form1 As Form, Args() As String)
	MainForm = Form1
	MainForm.Show
	Dim PagesManager As B4XPagesManager
	PagesManager.Initialize(MainForm)
End Sub
```

And the generated file (also the top part) looks like this:

```
AppType=JavaFX
Build1=Default,b4j.example
File1=category_layout.bjl
File2=MainPage.bjl
File3=subcategory_layout.bjl
FileGroup1=Default Group
FileGroup2=New Group
Group=Default Group
Library1=b4xpages
```

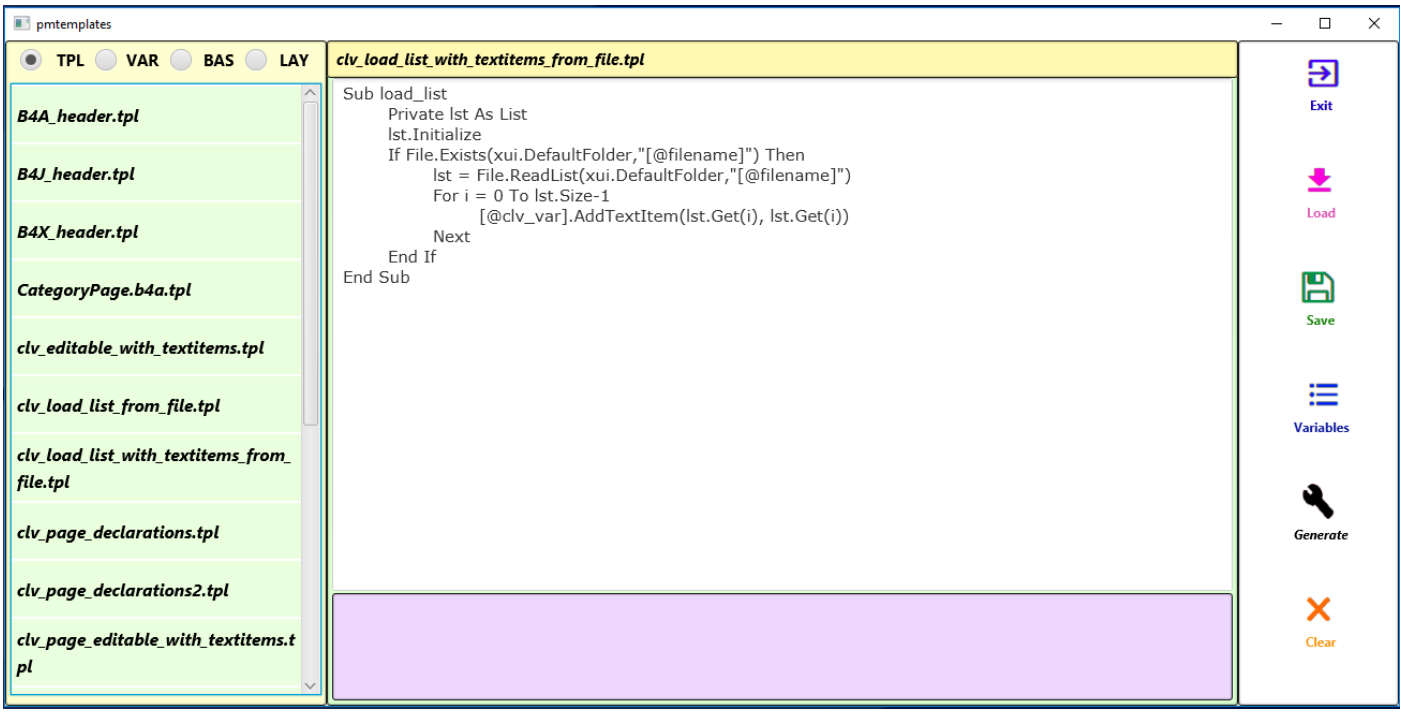
```
Library2=jcore
Library3=jfx
Library4=xui views
Module1=|relative|..\B4XMainPage
Module2=|relative|..\CategoryPage
Module3=|relative|..\SubCategoryPage
NumberOfFiles=3
NumberOfLibraries=4
NumberOfModules=3
Version=9.8
@EndOfDesignText@
#Region Project Attributes
	#MainFormWidth: 600
	#MainFormHeight: 600
#End Region
' from template example01.b4j.tpl
Sub Process_Globals
	Private fx As JFX
	Private MainForm As Form
End Sub

Sub AppStart (Form1 As Form, Args() As String)
	MainForm = Form1
	MainForm.Show
	Dim PagesManager As B4XPagesManager
	PagesManager.Initialize(MainForm)
End Sub
```

Layout files

If you load a layout file in the code then you have to make a layout. The best way by far is to use the IDE designer.
In the tool (see below) you can review a representation of an existing layout. It provides the information from the layout so that you can fill in the variables map.

pmtemplates B4J tool



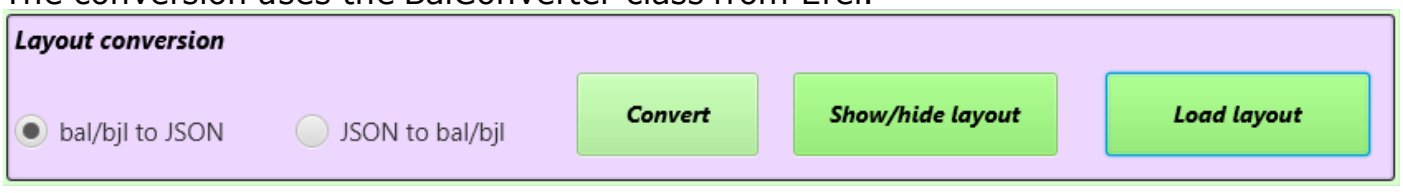
On the left side you find 4 files lists:

TPL: template files (extension = .tpl)
VAR: variables files (extension = .var)
BAS: generated files (extension = .bas)
LAY: layout files (extensions = .bal, .bil, .bjl, .json)

On the right side you find the menu items:
Exit: closes the application
Load: load a file (extensions .tpl, .var, .bas, .json)
Save: save the text to a file
Variables: fill in the variables map
Generate: generate the code
Clear: clears the working area and variables

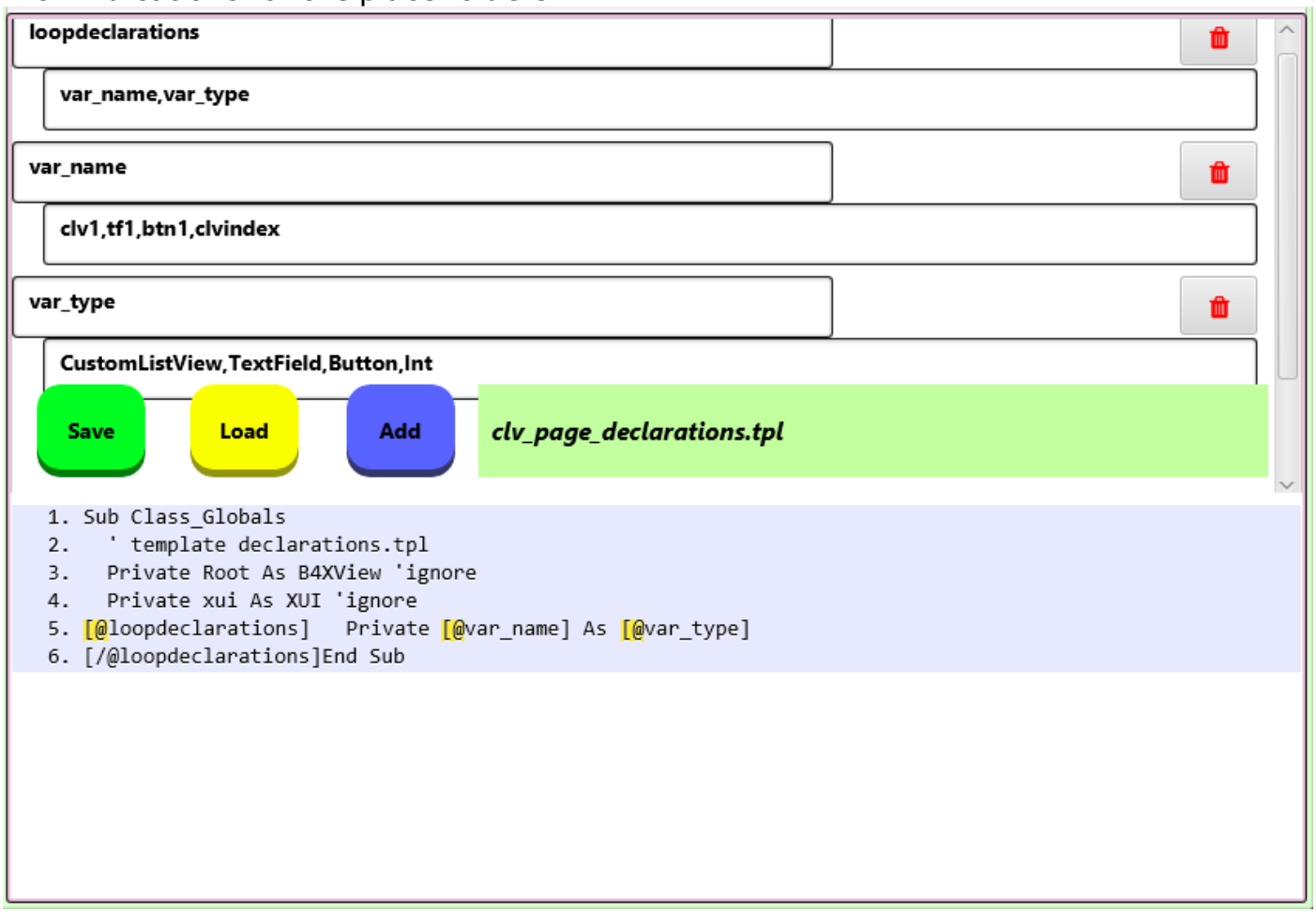
In the middle you can see the name of the file, the contents of the file (in a textarea) and some messages below the text.

When you select a .json layout file the bottom middle shows the layout buttons: convert a bal/bjl file to JSON or a JSON file to a bal/bjl file, show/hide a layout and load a layout (extension .bal,.bil,bjl).
The conversion uses the BalConverter class from Erel.



When you click on the Variables menu item the middle panel shows the variables list, buttons for adding a variable, loading all variables and saving all variables. The name

of the template file is displayed and below that a webview contains the template code with indications for the placeholders.



Steps to take to generate a template with the variables:

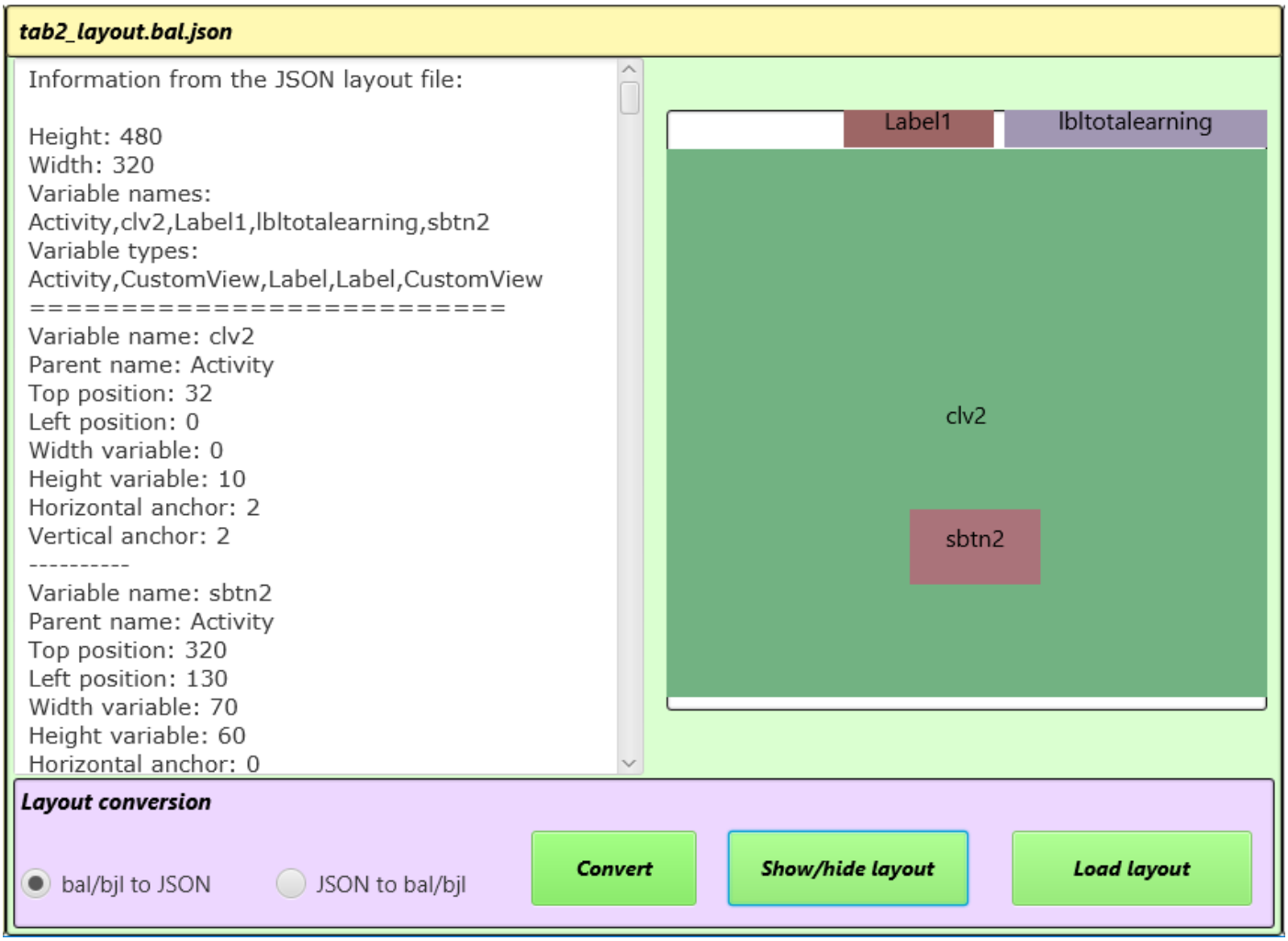
1. Select a template file
2. Insert placeholders where needed
3. Save the template file in the templatefiles folder and reselect the file
4. Click on the Variables menu item
5. Click on the Load button
6. Fill in the information for the placeholders.
   For a loop when there is more than one placeholder provide a list (seperator = ,) and make sure the number of items in each list are the same.
   In the loop variable you indicate which variables belong to the loop.
7. Click on the Generate menu item
8. Save the generated file in the generatedfiles folder
9. Check the result by selecting the newly generated file

The application creates 4 folders in
C:\Users\<yourname>\AppData\Roaming\pmtemplates
generatedfiles, layoutfiles, templatefiles, varfiles

Layout viewer

Steps to take to view a layout file:

1. Click on the Load layout button to select a .bal or .bjl file and click on open in the dialog (Note: the conversion doesn't support a .bil file yet)
2. Select the newly added .bal or .bjl file in the layout files list. The text in the middle mentions that it's a binary file format.
3. Click the radiobutton to be used (default is bal/bjl to JSON)
4. Click on the Convert button to start the conversion process
5. A message appears that the conversion was succesfull. Click OK.
6. Select the added .json file (same filename but the extension .json is added)
7. The JSON file information is displayed in the middle.
8. Click on the Show/hide layout button to see a representation of the layout.



And that's it. You can find the examples in the zip-file.
Happy coding!