Basic4Android Form Generator

Disclaimer

This SOFTWARE PRODUCT is provided by El Condor "as is" and "with all faults." El Condor makes no representations or warranties of any kind concerning the safety, suitability, lack of viruses, inaccuracies, typographical errors, or other harmful components of this SOFTWARE PRODUCT. There are inherent dangers in the use of any software, and you are solely responsible for determining whether this SOFTWARE PRODUCT is compatible with your equipment and other software installed on your equipment. You are also solely responsible for the protection of your equipment and backup of your data, and El Condor will not be liable for any damages you may suffer in connection with using, modifying, or distributing this SOFTWARE PRODUCT.

You can use this **SOFTWARE PRODUCT freely, if you would you can credit me in program comment:**

El Condor – CONDOR INFORMATIQUE – Turin

Comments, suggestions and criticisms are welcomed: mail to rossati@libero.it

Conventions

Commands syntax, instructions in programming language and examples are with font `COURIER NEW`. The optional parties of syntactic explanation are contained between `[square parentheses]`, alternatives are separated by `|` and the variable parties are in *`italics`*.

Contents table

1 Form generator

Form generator, briefly *FormGen*, is a class module for *Basic4Android* which allows to build and handle forms data; it is sufficiently generalized for a wide use.

1.1 Using the form generator

Before use FormGen the class `fgen` must be instantiate:

```
Sub Globals
...
      Dim fg As fgen      ' instantiate Form Generator class
...
End Sub
...
      fg.Initialize
```

The form is generated by calling the `fg` method:

instantiatedName`.fg(`*activity*`,`*dataDescription*`,`*callingModule*`,`*subHandleAnswer*`,`*subHandleEvents*`)`

where *activity* is the activity which will contains the form, *dataDescription* is a character string containing the form components description, *callingModule* is an activity module or a class module, *subHandleAnswer* and *subHandleEvents* are characters string containing the name of the sub for handle data when the form is closed and the possible sub for handle events or an empty string if you wouldn't handle events.
If the form is only for show data *subHandleAnswer* must be an empty string.

Example:

```
...
Dim parms As String = "Slide,,5,S,3,10 -10;Rdb,Sexe,,10,,M:Male|F:Female;"
...
fg.fg(Activity,Parms,Me,"handleAnswerTest","handleEvents")
...
fg.fg(Activity,Parms,Me,"handleAnswerTest","")
...
```

1.2 Data description

Every view (or widget) is characterized by a list of attributes comma separated, in this order: view Type, Field Name, Field Label, Length, Default Value and Extra. Views are separated by semicolon.

In addition to the views there can be some others information (*Pseudo types*) with different semantics that will be detailed in the paragraphs dedicated to them.

If the view list starts with `'` it is a comment which ☞ must also be terminated by semicolon.

1.2.1 View Type

The Types are indifferent to case.

- Buttons:
  - **B** button;
  - **R** radio button, a set of Radio buttons;
- **CKB** check box;

- Spinners (or Combo boxes):
    - **CMB** spinner;
    - **CMT** is a spinner with Text associated for insert values not in spinner;
    - **F** file and directory;
- Text fields:
    - **C** comment;
    - **N** numeric field;
    - **DN** decimal numeric field;
    - **H** or **HIDDEN** hidden field;
    - **S** seek bar or slider is an extension of the standard control;
    - **QS** qualitative slider;
    - **P** password field, the data entered are masked;
    - **T** text field is the default if the Type is omitted;
    - **U** not modifiable field i.e. a protected field.

1.2.2 View Name

Is the name of the field that, when the form is closed, is returned with the value associated; the name is case-sensitive and it is used to access data and possibly handle the views.

The *name* of comments (type **C**) is meaningless and it is ignored.

1.2.3 View Label

Label of widget or caption of button, if omitted it is used the Field Name.
The *label* of comments (type **C**) is meaningless and it is ignored.

1.2.4 View Length

The length of the view in characters; if it is omitted it is assumed depending on the *type* of node:

	Node type	**Value**
`DN`	Decimal numeric text	9
`F`	File	`30`
`N`	Numeric text	`7`
`QS`	Qualitative slider	`8`
`S`	Slider	`5`
`P`	Password	`16`
`R`	Radio button	`12`
`T`	Text node	maximum from `20` and the length of the possibly default value

1.2.5 Default value

Is the value proposed in form; the form is restored with the defaults values when the `Reset` button is pushed.

1.2.6 Extra(s)

Extra Field is used for add information to the view.

View type	
Check Box	a possible description after the check box
Radio buttons	an item list separated by |
Slider	the slider limits

Qualitative slider	an item list separated by |
Spinner	an item list separated by |
Text Fields	if the default field is empty, is the hint

The possibly second *extra* field of text views is a ToolTip.

1.2.7 Pseudo types

Pseudo fields are flavors for show form; they have a type and the syntax is different from the normal views.

1.2.7.1 After

The pseudo field `after` is useful for insert a button or a check box or a radio buttons at right of a view:

`after,`*`viewNameA`*`,`*`viewNameL`*

where *`viewNameA`* is the view to be placed at right of the view *`viewNameL`*

☞ In the list of widgets *`viewNameL`* must appears before *`viewNameL`*.

```
...
& "N,Age,,5,,age;" _
& "R,Sex,,10,Masculin,M:Man|F:Féminin;" _
& "After,Sex,Age;" _
...
```

1.2.7.2 Check

This pseudo field is used for some controls on fields:

`check,`*`fieldName operator (value|fieldName)[,errorMessage]`*

where operator can be one of =, >, <, <>, >=, <= or `is`; after `is` operator can bee:

`mail|`*`regularExpression`*

```
...
& "T,email,My EMail,20;" _
& "P,password,type password;" _
& "P,repeatPassword,retype password;" _
& "check,email is mail,Incorrect mail form;" _
& "check,password=repeatPassword;" _
& "Check,psw is .{6#44},Password too short;" _
...
```

☞ if *`value`* contains comma or semicolon they must be masked (see paragraph 1.4.1).

1.2.7.3 Defaults

The syntax is:

`defaults,`*`viewName`*`:`*`viewValue`*`[,...]`

it is useful for populate the form.

☞ if *`viewValue`* contains comma or semicolon they must be masked.

1.2.7.4 Background

The pseudo field `Ground` or `Background` create a background in the form or a gradient with a possible effect of transparency; the syntax is:

`[Ground|Background][,`*`fromColor [toColor direction]`*`]`

The colors are in hexadecimal notation with two digit for every component: `RRGGBB`, in this case the color is opaque; the transparency is a hexadecimal value from `00` to `FF` appended before the color, see the examples below:

```
"Ground;"                 ' default FF404040
"Ground,FF;"              ' blue
"Ground,7F0000FF;"        ' half transparent blue
"Ground,FF 8000 LEFT_RIGHT;"    ' gradient blue to green
```

direction can be:

- TOP_BOTTOM or TR_BL (Top-Right to Bottom-Left),
- RIGHT_LEFT or BR_TL (Bottom-Right to Top-Left),
- BOTTOM_TOP or BL_TR (Bottom-Left to Top-Right),
- LEFT_RIGHT or TL_BR (Top-Left to Bottom-Right).

The default *direction* is TOP_BOTTOM.

☞ Without the pseudo field Ground the form has a gray background; for a transparent background use a 00 transparency, for example: Ground,**00**000000;.

1.2.7.5 Required

A list of fields that mus be inserted:

required,*field*$_1$[,*field*$_2$...]]

1.2.7.6 Hidden field

Is the type **H** or **HIDDEN**; the syntax is: [H|Hidden],*fieldName*,*value* es:

```
DateTime.DateFormat = "yyyy-MM-dd HH:mm:ss"
...
"Hidden,TimeStamp," & DateTime.Date(DateTime.Now) & ";"
```

1.2.7.7 Section

section,*sectionName*[,*condition*[,*condition*[,...]]]

This pseudo type is for multi-forms, he must precede its views.

The *condition* has the form: *fieldName operator* [*value*|*fieldName*] where operator can be one of =, >, <, <>, >=.

Sections are are displayed in the order in which they are present, a Section whit condition is displayed only if all condition are verified.

In case of multi section are added two navigation button: fh_Forward and fh_Back with caption respectively --> and <--; the Ok button is present only on the last section.

☞ the views before the first section are repeated on all form.

1.2.7.8 Tab

tab,*tabTitle*[,*defaultIcon*[,*selectedIcon*]]

tab allows to organize the form through a tabHost view; it can be used in sections.

The buttons are visible on all tabs; *defaultIcon* and *selectedIcon* are graphic files that must be present in the DirAssets.

☞ before the first tab there can be ground and title pseudo type, any other view is ignored.

1.2.7.9 Title

Title,*name*,*formTitle*[,*backgroundColor*]

```
TITLE,Title,Send mail parameters,FF202020;
```

If *name* is omitted it is set to fh_title.

1.2.8 Summary by type

1.2.8.1 Default and extra field

Type	Length	Default field	Extra field(s)

B	Ignored	possibly `dis[able]` or `cancel`	Possibly name of CallBack function
C		the data	
CKB		`1` or `check[ed]` = checked	Possible Description at right of check box
CMB		*`value`*	An item list separated by `|:` *`[key:]value`*
F		Initial folder	
S		Initial value	Start and end value, default is `0 100`
T,N,DN,P		Initial value	hint, tooltip
U		Not modifiable text	

1.2.8.2 Buttons

The package adds the standard buttons `Ok`, `Cancel` and `Reset`, if there are sections on all forms except the last the `Ok` button has caption `-->` and all forms except the first has a button with caption `<--` and name `fh_Back`.

The buttons can be used both for take different actions on closing form both for show user caption instead of default `Ok`, this last can be obtained not only for `Ok` button with this syntax:

```
B,[Ok|Cancel|Reset],caption
```

The value of *default* field `dis[able]` is used to start the form with the button disabled.

The *extra* field contains a name of the function called when a CallBack Button is pushed, in the form: *`moduleName.functionName`*. or *`functionName`* if it is in the module which required the creation of the form.

The Ok button is replaced if there is almost one type **B** control in the list not associated, by `AFTER` pseudo type, to some control.

```
...
Dim fg As fgen      ' instantiate Form Generator class
...
Dim frm as String = "N,n1,Integer,10,;n2,DN,Decimal,10;B,Multiply,,10,,Main.CallBack"
fg.fg(Activity,frm,"Main.handleAnswerTest","")
...
Sub CallBack(btnName As String)   ' CallBack event handler
      Dim n1 As Float = fg.iIF(IsNumber(fg.valueOf("n1")),fg.valueOf("n1"),0)
      Dim n2 As Float = fg.iIF(IsNumber(fg.valueOf("n2")),fg.valueOf("n2"),0)
      Msgbox(n1*n2,btnName)
End Sub
...
```

The label of button can be a name of an image that must be in the asset folder or an Unicode character which is a simple and efficient way to create buttons with pictures:the Unicode characters is in the form `#nnnn` or `#xeeee` where *`nnnn`* is a decimal value of the Unicode character and *`eeee`* is s the hexadecimal value of the Unicode character.

```
B,Cancel,#x2718;
B,Reset,#x21B6;
B,Start,#9998,,myHandler,Go;
```

Name	Decimal value	Symbol	Hexadecimal value
edit	#9998	✎	#x270E
delete	#10008	✘	#x2718
check	#10003	✓	#x2713
check bold	#10004	✔	#x2714
email	#9993	✉	#x2709
cross	#10006	✖	#x2716
dollar		$	#x24
euro		€	#x20AC

Table 1: Some UNICODE characters

pound	£	#xA3
white square	☐	#x25a2

1.2.8.3 Check box

For checked box insert into *default* field `check[ed]` or `1`.
The *extra* field can contain a possibly description at right of the check box.
The value returned of check box is a string containing 0 or 1, they must be compared as string:

```
cmd.Append(fg.iIF(fh_Data.Get("Mandatory") = "1","M",""))
```

1.2.8.4 Comment

The comment is contained in the *default* field, it occupies the space of the label and the view.

1.2.8.5 File

The initial Folder (and possibly file name) is in *default* field to start, if it is omitted or if it is not a folder, the `File.DirRootExternal` is assumed.

1.2.8.6 Radio buttons

It is possible to have more than one set of radio buttons.
The *length* is the length of the single view; the *extra* field contains the item list separated by `|`. For get a key instead the description, the item must have the form: `key:value`.
The *default* value can be the data showed or the key:

```
Rdb,Status,10,R,Single,M:Married|S:Single|W:Widow;
Rdb,AgeType,10,R,Y,M:Months|Y:Years;
```

1.2.8.7 Slider

The *length* is the length of the text which shows the slider value.

The *extra* field of the type **S** can contains the start and end values in the form `start end`, e.g. `-5 5`; the range is `0 100 if` omitted, if only one value is present, the default value for the second is `100`; the result can have decimals depending on the difference from `start` and `end` value, see table at right; `start` can be greater of `end` e.g.:

```
Slider,,5,S,-3,10 -10
```

abs(start - end)	n. decimals
> 99	0
<100 and > 10	1
<10 and > 1	2
<1 and > 0.1	3
...	...

The qualitative seek bar (type **QS**) returns qualitative values taken from the *extra* field where they have the same syntax of the values of radio buttons:

```
...
QS,Urgency,,8,Green,White|Green|Yellow|Red
...
```

1.2.8.8 Spinners

CMB is a simple spinner, if no member is selected the value returned is an empty string; if the form has only one **CMB** spinner, there is only a cancel button and the form is exited when a spinner item is selected.
The *extra* field of spinners contain the item list separated by `|` (see description in Radio button).
CMT type is a spinner with text associated for insert a possibly value not in spinner.
CMX type is a spinner with text, every choice in spinner is recorded in the text, this is useful for example to compile a list of symptoms.

1.2.8.9 Text fields

For text type (**T**, **P**, **N**, **DN**) the possibly *extra* field is the *hint*; the possibly second *extra* field is the ToolTip.

1.2.9 Returned Values

The data are accessible in the Sub indicated as third parameter of the call, which is called when the buttons `Ok` or `Cancel` or the possible type **B** button is pushed.

The sub has a parameter which is the map which contains the data which are accessible via `Get` or `GetDefault` methods: the key is the field name, besides there is the element with key `fh_button` which contains the name of the button pushed (`Ok` or `Cancel` or the name of the button pushed).

☞ If `Cancel` button was pressed there is only `fh_Button` element.

1.3 CallBack

FormGen works on CallBack not only at the end of form, but also, possibly, for handle events (the fourth parameters of the call), or by a sub associated to a **B** button (the *extra* field). The CallBack function receive the button name.

1.3.1 Handle Events

This function can be used to personalize the form e.g. modify the state of the view, perform controls end so on. The functions receive an array which contains view name and event, besides, for views, contain the view handle, the value and possibly extra field, see below.

Parameters		
	Event	**Note**
event	`Start`	view name = `fh_start`, no view handle and value.
event	`Cancel`	view name = `fh_cancel`, no view handle and value.
event	`End`	view name = `fh_end`, no view handle and value.
event	`Reset`	view name = `fh_reset`
`Buttons`	`CLICK`	
`EditText`	`FOCUS`, `LFOCUS`	Focus and lost focus.
`EditText`	`VREND`	At end of Voice recognition (if is active)
`Check box`	`CHK`	Value = 1 if checked, else 0
`Seek bar`	`SLIDE`	Extra is the handle of text containing the value.
`Spinner text`	`CLICK`	

The button `CLICK` event precedes the closing of the form, however, it is possible to inhibit the closing by setting the property `fh_yesToExit` to `False`.

```
Sub handleEvents(parm() As Object)      ' widget events handler
  Dim value As String = ""
  If parm.Length > 2 Then
       If parm.Length > 3 Then value = parm(3)
  End If
  Log("Handle event " & parm(0) & " event: " & parm(1) & " Value: " & value)
       Select parm(0)
         Case "fh_start","fh_reset"
         Case "Message"
              If parm(1) = "VREND" Then
                Dim txt As String = parm(3)
                parm(3) = txt.SubString2(0,1).ToUpperCase & txt.substring(1) & "."
              End If
         Case "Parms"
              fg.fg(Activity,"Prova,,,t","Main.handleAnswer","Main.handleEvents")
         Case "Send"
```

```
            sendMail(parm(0))
            fg.fh_yesToExit = False
      End Select
End Sub
```

Example of sub for handle events

1.3.2 Handle CallBack

The function is called when a type **B** button with call back function (the *extra* field) is clicked; the function receive the name of the button. The values of view can be accessed by the function `valueOf(viewName)`.

```
Sub Globals
...
   Dim fg As fgen    ' instantiate Form Generator class
...
   Dim Finder As String = $"T,Name;CMB,list,List;B,Find,-->,6,,Main.loadNames;"$  _
            & $"B,New;B,Ok,See,,disabled;After,Find,Name;H,app,listProd"$
...
End Sub
...
fg.fg(Activity,"Title,title,Find Products;" & Finder,"Main.handleAnswer","")
...
Sub handleAnswer(fh_Data As Map)
   If fh_Data.Get("fh_button") <> "Cancel" Then
...
    End If
End Sub
Sub loadNames(btnName As String)
   Msgbox(fg.valueOf("app"),"")
End Sub
```

Sample of Callback function

☞ We can also manage the CallBack in the event management function, in this case it is not necessary to specify a dedicated function.

1.4 Remarks

1.4.1 Masking comma and semicolon and insert Unicode characters

If *label* or *default* or *extra* or condition contains commas or semicolons, the function `mask(data)` must be used, commas are replaced by `#44` and semicolons by replaced by `#59`[1]:

```
Dim cmd As StringBuilder
...
instantiatedName.mask(data_to_be_masked)
cmd.Initialize
cmd.Append(fh_Data.Get("Name") & ",")
cmd.Append(fg.mask(fh_Data.Get("Label")) & ",")
cmd.Append(fh_Data.Get("Length") & ",")
cmd.Append(fh_Data.Get("Type"))
cmd.Append("," & fg.mask(fh_Data.Get("Default")) & ",")
cmd.Append(fg.mask(fh_Data.Get("Extra")))
```

1.4.2 Handling Buttons

Form Generator inserts the `Ok` button, the `Cancel` button and the `Reset` button depending on the views contained in the form:

[1] For compatibility are accepted the deprecated `chr(1)` for comma and `chr(2)` for semicolon.

- the `Cancel` button is always present,
- the `Reset` button is present if there are data fields**,**
- the `Ok` button is not present if there is only one spinner (**CMB** type) or some others buttons.

If the form contains only not modifiable views (*type* **U**), there is only the `Cancel` button.

The Forward (`-->`) and Back (`<--`) buttons are always present in case of multi sections.

1.4.3 Data presentation

The data are presented in the order they appears in the parameters list, except for the Type **B** buttons that appears together buttons inserted by *FormGen*, at the bottom of the form.

For view of Type Text, if the length exceed the maximum characters allowed for the line, the view is multi lined; this maximum characters for line depends from the labels width.

With the pseudo type `after` buttons or check box can be placed at right of another view.

1.4.4 Work with views

We can modify the view properties getting the view by the function `getHandle`; therefore for some properties there are specific functions:

- enable view: **`enable`**`(`*`viewName`*`)`
- disable view: **`disable`**`(`*`viewName`*`)`
- get the handle of the view: **`getHandle`**`(`*`viewName`*`)` In case of radio button is the handle of radio button checked.
- Change the value: **`setValue`**`(`*`viewName`*`,`*`value`*`)`
- Get the view value: **`valueOf`**`(`*`viewName`*`)`

Examples:

```
fg.disable("btnGo")
Dim lbl As Label = fg.getHandle("title")
lbl.TextColor = Colors.Green
If fg.valueOf("Consent") = 1 Then fg.enable("btnGo")
fg.SetValue("Slider",0.2)
fg.setValue("Number",400)
fg.setValue("Spinner","Delta")
fg.setValue("UnMod","**********")
fg.setValue("Rdb","Married")
```

1.4.5 Voice recognition

It is possible to use voice recognition for enter text spoken into text field; this is done by the function `startVoiceRec`.

Voice recognition is deactivated, for enable uncomment:

```
'Dim VR As VoiceRecognition
'VR.Initialize("VR")
Sub startVoiceRec()
  'VR.Listen 'calls the voice recognition external activity
End Sub
```

In the event `VREND` we can access the text and modify (see the above example).

```
Sub handleEvents(parm() As Object)      ' widget events handler
          Dim value As String = ""
          If parm.Length > 2 Then
                    If parm.Length > 3 Then value = parm(3)
```

```
          End If
          Select parm(0)
             ...
             Case "Message"
                If parm(1) = "VREND" Then
                  Dim txt As String = parm(3)
                  parm(3) = txt.SubString2(0,1).ToUpperCase & txt.substring(1) & "."
                End If
             ...
       End Select
End Sub
```

1.5 Others functions and utilities

FormGen module contains, may be, useful functions; some are just seen above at paragraph 1.4.4 Work with views.

1.5.1 Add values to spinner

```
splitKeyValue(name As String, data As String) As List
```

Where `name` is the spinner name, `data` is a string of items in the form: `[key:]value`. separated by `|`.
The `splitKeyValue` populates the `mapValues` map; the returned List is used to populate the spinner.

```
Dim cmb As Spinner = fg.getHandle("listProp")
cmb.clear
cmb.addall(fg.splitKeyValue("listProp",fg.implode("|",listExpl)))
```

1.5.2 Dialog method

☞ This method is sperimental, it is based on deprecated `doEvents()`, and don't works on B4A 7.

```
Dialog(act As Activity,title As String,Default As String) As String
```

This method asks an input string modal:

```
...
Dim fg As fgen ' instantiate Form Generator class
...
fg.Initialize
...
name = fg.Dialog(Activity,"Get Name","Unknown")
```

1.5.3 iIf

`iIf` function return an object depending on a test:

```
Dim Number As Int = fg.valueOf("Number")
Log(Number & " is " & fg.iIf((Number Mod 2) = 0,"even","odd"))
```

1.5.4 Right and Left string functions

Right and Left functions returns a pieces of string stripped by some extent:

```
Log(fg.Left(fg.Right("Condor Informatique",12),6))    ' Inform
```

1.5.5 Get filename components

`pathinfo`[2] returns a map containing information about a file ex:

```
Dim fileInfo As Map = pathinfo("/www/htdocs/index.html")
```

- `dirname` `/www/htdocs`
- `basename` `index.html`
- `extension` `html`
- `filename` `index`

1.5.6 Get the widget Handle

```
Dim cmb As Spinner = fg.getHandle("listProp")
```

[2] This is similar to PHP `pathinfo` function.

```
cmb.clear
cmb.addall(fg.splitKeyValue("listProp",fg.implode("|",listExploiteurs)))
```

☞ the handle of a not modifiable field (**U**) is a label handle.

☞ For Combo Text (**CMT**) and Combo Extended Text (**CMX**) the handle is relative to the text associated with the spinner, for access to the spinner:

```
Dim cmbTxt As View = fg.getHandle("Commune")
Dim aWdg() As Object = fg.widgetRef.Get(cmbTxt.Tag)
Dim cmb As Spinner = aWdg(3)
...
```

1.5.7 Extend Log function

`toText` is a function for speedy logs; it has two parameters, the first is a string containing some text and a formatting command (`%` or `%n` or `%n.d`), the second is an array of object. Every `%` is replaced by an element of the second parameter. The possible *n* is a length of output, the possible *d* is the number of decimals to show.

```
Dim aaa() As Object
aaa = Array As Object (3,3.14,"vintun",False,True, 18723)
Dim t As String = "% %6.3 %11 %1 aaaa %3 %12.2"
Log(fg.toText(t,aaa))
```

The result is:

```
3  3.140 vintun      f aaaa tru    18,723.00
```

1.5.8 Implode

```
implode(separator As String, obj As List) As String
implodeFrom(separator As String, obj As List, From As Int, At As Int) As String
```

`implode` function returns a string containing all elements of an array or list with separator between each element:

```
Log(fg.implode("<br>",aaa))
```

The risult is:

```
set<br>3<br>3.14<br>vintun<br>false<br>true<br>18723
```

1.5.9 TypeOf Object type

`typeOf` is a function which return a type of object, stripping `java.lang` from what is returned by `GetType` Basic4Android function.

```
Select TypeOf(values(j))
   Case "Integer","Double"
      ...
```

1.5.10 Sand box

The Sand box is a demo of *FormGen* and a tool for testing the forms. The initial form, not created by *FormGen,* has a text box and some buttons:

- Add button generate a form for create a view.
- Test button generate a form starting from what is contained in the text box.
- Button for generate samples of selected type.
- Sample button generate a sample form with some views and a CallBack button.

2 Technical notes

2.1 Library

Form generator need Core library and Phone library (if use Voice recognition).

2.2 Maps

Name	Key	Value	Note
allButtons	*ButtonName*	Array(*Caption*,*CallBack*)	String array
after	*viewAfter*	*viewBefore*	
checks	Field name	control on view	array of comma separated checks (1)
Defaults	Field name	value	#44
fh_Data	Field name	value	map of data
fh_FieldsType	Field name	Field type	
hiddensMap	Field name	value	map of hidden fields
limits	*fieldName*Min, *fieldName*Max, *fieldName*Delta, *fieldName* (*QS type*)	value items (*QS type*)	Slider limits; for qualitative slider are list of values
mapValues	Field name & value	key	For Radio buttons and Combo box (spinner)
required	Field name		for required fields
tabsMap	Page name	Array (3)	
tipsMap	Field name	tip	
widgetRef	Field name	Id, widgetType, label, extraField1	(2)

1) `checks` array of bi dimensional array:
   - the check condition *`fieldName operator (value|fieldName)`*
   - possibly error message.

2) `widgetRef` contains a reference to all view and TITLE pseudo field
   1. key = *`fieldname`*, value = `array(`*`Id, widgetType, label, extraField`*`)`
   2. `Id` is a widget `ID`, for RadioButtons is a panel container
   3. *`extraField`* is:
   4. for button a possibly sub for CallBack,
   5. for seekbar (type **S**) is the ID of text containing the value,
   6. for file (type **F**) is the file path,
   7. for spinner text (type **CMT** and **CMX**) is the ID of the spinner associated.
   8. `for RadioButton is a radiobutton selected.`

3) `tabsMap` contains data for tab
   - background `color` e.g. `FF 8000 LEFT_RIGHT`,
   - possibly *`defaultIcon`*,
   - possibly *`selectedIcon`*,
   - title array (*`Name`*, *`title`*, *`background color`*)

2.3 Lists and arrays

- `Elem` array of field descriptions.
- `rdbList` the data array contains normalized RadioButtons data description.
- `Sections` two dimension array (number of sections, 3):
  - section widgets,
  - section name,
  - check condition(s).

3 History

Version 0.3.1.1

- Button with `cancel` in default field for do not show `Cancel` button

Version 0.3.1

- function `implode` accept in addition to the `Arrays` also `Lists`,
- a form with only a single spinner exits when an element is selected.
- pseudo field GROUND added

Version 0.4.x

- the ground is set to black, if no ground is present,
- added `checks` pseudo field for add controls,
- add comments,
- the fields describing the view are rearranged, the type is the first item, this permits a neatest evolution of the software,
- eliminated the mandatory (M) character for the fields that must exists, this is replaced by the pseudo type `required`,
- added the pseudo type `defaults` and `after`,
- choice of the caption for buttons `Ok`, `Cancel` and `Reset`,
- pseudo type `Hidden`.

Version 0.5.x

- Dialog method,
- qualitative slider,
- check improved by comparison operators,
- multi form management,
- Graphic button and button with Unicode characters.

Version 0.5.3

- Eliminated BC type

Version 0.6.2

- added the C (comment) type

Version 0.7.0

- Error messaging has been improved,
- added Tab pseudo type,
- minor corrections.

3.1 Differences of version 4.x from previous version

3.1.1 Buttons

- Default field of type **B** buttons can contain the disable command, instead of the value returned when the button is clicked; the value returned is the button name.
- Removed the option `cancel` in default field of button for do not show `Cancel` button.

3.1.2 View list

The fields are rearranged, the type is the first item, this permits a neatest evolution of the software.

3.1.3 Others

- Removed functions `stripFile`. it has substituted by `pathinfo` that returns directory, file name and extension.
- Eliminated the mandatory (M) character for the fields that must exists, this is replaced by the pseudo type `required`,

3.2 Differences of version 0.5.3 from previous version

Eliminated the BC button, the call back is implicit if the *extra* field is not empty.

The Tip *pseudo type* has been eliminated, the tip is on the second *extra* field of text views.

3.3 Differences of version 0.6.x from previous version

The form of calling has changed:

```
instantiatedName.fg(activity,dataDescription,callingModule,subHandleAnswer,sub
HandleEvents)
```

The *callingModule* is an activity module or a class module *subHandleAnswer* and *subHandleEvents* are the name of subs concerned. This change has become necessary to be able to call between modules.

4 Annexes

4.1 Introduction to regular expressions

A regular expression is a string of characters used to search, check, extract part of text in a text; it has a cryptic syntax and here there is a sketch with a few examples.
The regular expression can be prefixed by modifiers such as **`(?i)`** to ignore the case.
The expression is formed with the characters to search in the text and control characters, among the latter there is a `\` said *escape* used to introduce the control characters or categories of characters:

- **`\` escape character**, for special characters (for example asterisk) or categories of characters:
  - **`\w`** any alphabetical and numerical character, **`\W`** any non alphabetical and numerical character,
  - **`\s`** *white space* namely. tabulation, line feed, form feed, carriage return, and space,
  - **`\d`** any numeric digits, **`\D`** any non digit,
- **`.`** any character,
- **quantifiers**, they apply to the character(s) that precede:
  - **`*`** zero or more characters
  - **`+`** one or more characters
  - **`?`** zero or one character (means possibly)
  - **`{n}`**, **`{n,}`** and **`{n,m}`** respective exactly *n* characters, almost *n* characters and from *n* to *m* characters.

`(...)` what is between parentheses is memorized,
`?=`*`pattern`* checks if *`pattern`* exists,
`[a-z]` any letter from `a` to `z` included,
`[`*`a`*`|`*`b`*`]` `a` or `b`,
`\b` word boundary,
`$` (at the bottom),
`^` (at start).

4.1.1 Examples

`^\s*$`	Empty set or white spaces
`aa+`	Find a sequence of two or more `a`, like `aa`, `aaa`,....
`(\w+)\s+(\w+)\s+(\w+)`	Find and memorize three words
`(\-[a-z])`	Find and memorize `minus` followed by one alphabetic character
`.(jpg|jpeg)$`	Controls file type jpg or jpeg
`^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$`	Control of mail address
`^\d+$`	Only integers
`((?=.*\d)(?=.*[a-z]+)(?=.*[\W]).{6,12})`	`(?=.*\d)` almost a digit from `0-9` `(?=.*[a-z])` almost one lowercase character `(?=.*[\W]+)` almost one special character `.` match anything with previous condition checking `{6,12}` length at least 8 characters and maximum 20
`^[-+]?\d{1,2}(\.\d{1,2})?$`	**Numeric values** `[-+]?` the sign is possible `\d{1,2}` one or two digits `(\.\d{1,2})?` It is possible to have a decimal point followed by one or two digits
`[aAbBcCdDeEfF\d]{8}`	8 hexadecimal digits

5 Alphabetical Index