Julia Form Generator

Disclaimer

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

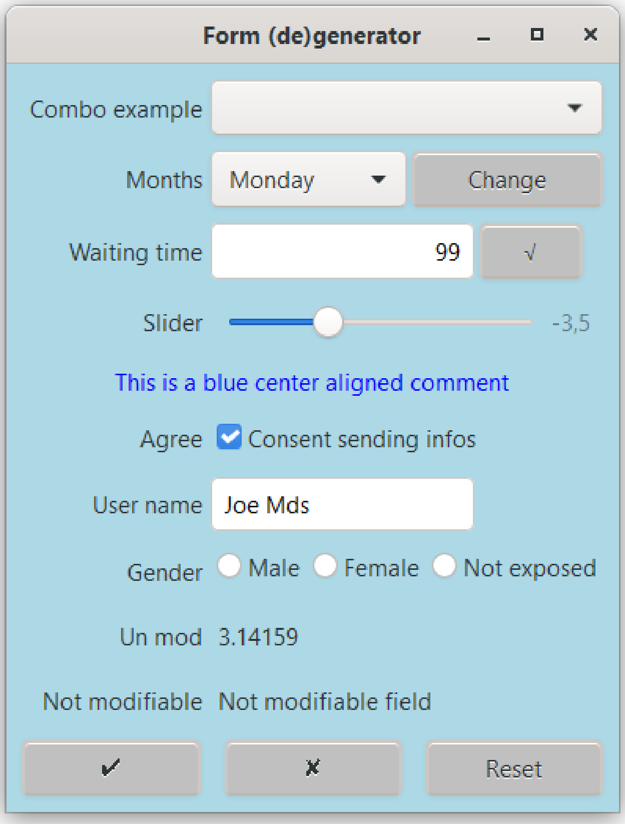
Conventions

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 Julia **Script** which allows build and handle forms; *FormGen* is sufficiently generalized for create a wide set of useful forms from simple message box to complex input forms, based on a list of input controls (some text type, buttons, check boxes, lists, radio buttons, slider, combo box...).



Combo example

Months Monday

Waiting time 99

Slider -3,5

This is a blue center aligned comment

Agree ☑ Consent sending infos

User name Joe Mds

Gender ○ Male ○ Female ○ Not exposed

Un mod 3.14159

Not modifiable Not modifiable field

1.1 Using the form generator

The form builder is contained in `formgen.jl` script, which contains the object function `formGen`.

This function can be invoked for create a new object:

`formGen(`*`control_list`*`,`*`callBack`*`)`

Where *`callBack`* is a function that is called when the form is exited, this function receives a Dictionary of the data contained in the form.

```
using Dates
function answer(data)
    for key in keys(data)
        println(key * "\t" * data[key])
    end
end
include("formgen.jl")
#using Formgen
date = Dates.format(now(), "yyyy-mm-dd HH:MM:SS")
frm = """
Window 'Form (de)generator'  400 200
CMB combo 'Combo example' 25 Alfa|Beta|Gamma|Delta
N wTime 7
S slider 10 -10
Default wTime 99 userName 'Joe Mds' hField '$date' unMod 'Field not
modifiable'
T userName 30
U unMod
H hField"""
formGen(frm,answer)
```

1.2 Data description

Every widget (or control) is characterized by a list of attributes, separated by space(s), in this order: `Type`, `Field Name`, `Field Label`, and `Extra(s)` parameters. The widgets are separated by `newline` character; if `Field Label` or `Extra(s)` contain spaces they must be enclosed in quotation marks (`'`) or double quotes (`"`).

`Extra` can be specific for the widget and in this case they must follow the label field, in general they are a self explanatory constant like `center` or a couple `key value`, for example `call functionName` or `after fieldName`; in case of multiple parameters the order is indifferent; the `key` is case indifferent.

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

The `Type` is indifferent to the case; if `Type` starts with `#` it is a comment which ☞ must also be terminated by `newline`.

1.2.1 Types

- **Buttons**:
  - **B** button;
  - **R**, **RDB** radio button, a set of Radio buttons;
  - **NAVIGATE**, **NV** button for tell Next or Previous;
- **CHECK**, **CKB** check box;
- **Combo box**:
  - **COMBO**, **CMB** drop down list for select an item;
- **Text** fields:
  - **C**, **COMMENT** comment;
  - **Date** field;
  - **F** or **FILE** input file;
  - **FOLDER**;
  - **H** hidden field.
  - **N** integer number, **PN** positive integer number and **FN** floating number;
  - **S**, **SLIDER** seek bar or slider;
  - **P** or **PSW** password field, the data entered are masked;
  - **T** or **TEXT** text field;
  - **U**, **UN** not modifiable field i.e. a protected field, **UN** is numeric for right alignment.

1.2.2 Field Name

Is the name of the control that, when the form is submitted, it is used by the server to access its value; ☞ the name is case-sensitive.

1.2.3 Field Label

Is the label of control or the caption of button; if omitted it is used the `FieldName` that it is transformed if it has those formats:

- `fieldName` it becomes `Field name`,
- `field_name` it becomes `Field name`.

☞ If there are `Extra(s)` fields the `label` must be present at most as an empty string id est `''` or `""`.

1.2.4 Extra

`Extra field(s)` is (are) used for add information to the control, these will be specified in the relative data description paragraphs.

☞ The order of parameters are indifferent, the parameter name is case indifferent.

Type	B	CKB	CMB	Comment	Date	F	Folder	RDB	S	T	U, UN	Window	Label
After	X												X
Background												X	
Center				X									X
Color				X									X
Default		X	X		X			X	X	X	X		
Filter						X							
Format					X								
Ground												X	
Height				X									
Hint						X	X			X			
Left				X									X
onActivate										X			
onChanged			X										
onEnd												X	
onStart												X	
Reset												X	
Right				X									X
Static												X	
Title					X	X	X						
Value		X	X		X			X	X	X	X		
Vertical								X					
Width	X				X	X	X			X			

Type		Extra field(s)
`Check box`	**`CKB`**	Possible label at right of check box
`Combo box`	**`CMB`**	An item list separated by `comma`: `[`*`key:`*`]`*`value`*`[|[`*`key:`*`]`*`value`*`[|...`
`File`	**`F, FILE`**	Filters
`Date`	**`DATE`**	
`Radio button`	**`RDB`**	An item list separated by `comma`: `[`*`key:`*`]`*`value`*`[|[`*`key:`*`]`*`value`*`[|...`
`Slider`	**`SLIDER, S`**	Start, end and step value, default is `0 100 1`

1.3 Summary by type

1.3.1 Buttons

Buttons can be used both for take different actions on form both for show user caption instead of default `Ok`, `Reset` or `Cancel`.

The syntax is:

```
B name [caption][call functionName] [after textFieldName|comboBoxName]
```

☞ The arguments of the function called are *`name`* and *`widgetName`* (the argument of `after`).

For change the caption of `Ok` or `Reset` or `Cancel` button the syntax is:

```
B [Cancel|Reset|Ok] newCaption ...
```

The Unicode characters are a simple and efficient means to create buttons with pictures:

```
B Cancel \u2718
B Reset \u21B6
```

The default order of buttons is `Ok`, `Cancel` and `Reset`.

Table 1: Some UNICODE characters

Name	Symbol	Julia Code
	↶	`\u21B6`
edit	✎	`\u270E`
delete	✘	`\u2718`
check	✓	`\u2713`
check bold	✔	`\u2714`
email	✉	`\u2709`
cross	✖	`\u2716`
euro	€	`\u20AC`
info	ℹ	`\u2139`
pound	£	`\u00A3`
white square	▢	`\u25A2`
Ballot box	☐	`\u2610`
Ballot box with check	☑	`\u2611`
Square root	√	`\u221a`
Cubic root	∛	`\u221b`
Central ellipsis	⋯	`\u22EF`

Example 1: Buttons after widget

```
Window 'Numbers form'
N naturalNumber
PN positiveNumber 'Waiting time'
FN floatingNumber
S slider '' -10 10
Required naturalNumber positiveNumber
Control positiveNumber <= 1000 'waiting too
long'
B sqrt \u221a after positiveNumber call
roots
B cubicRoot \u221b After floatingNumber call
roots
```

```
function roots(btn,widget)
    local n = formgen.valueOf(widget)
    if btn == "cubicRoot"
        println("Cubic root of ",n, " = ",
formgen.valueOf(widget)^(1/3))
    else
        println("Square root of $n = ",
sqrt(formgen.valueOf(widget)))
    end
end
```

☞ The name `Cancel`, `Reset` and `Ok` are reserved names.

1.3.2 Check box

The extra field of **`CHECK`** or **`CKB`** type can contain a possible description displayed to the right of the check box.

In order to show the check box text (at right) the label must be set even empty, see the example below.

```
CKB Agree '' 'Consent sending info' value on
```

1.3.3 Chek box list

`CKL` type generates a set of check boxes arranged vertically.

The extra field contains a list of field names separated by `|` with syntax: `[`*`key`*`:]`*`value`*`[|` `[`*`key`*`:]`*`value`*`[|...`; the field name of check box is *`key`* if present, otherwise is *`value`*; *`value`* is the description that appears after the check box.

The *Field Name* of the check box list will contain the number of check boxes selected.

Ex.

```
CKL Languages '' 'C:C, C++, C#|JS:JavaScript|Julia|PHP|PYTHON|RUST' default C
Default PHP on
```

☞ check box list accept to set one of the check box to on by the `value/default` parameter, others check boxes can be defaulted to `on` by the pseudo type `Default`.

1.3.4 Combo box

`Cmb` *`fieldName label valueList`* `[onchanged` *`function`*`|submit]`

The *`valueList`* field of **`COMBO`** or **`CMB`** type contains the list of combo items separated by `|`. To obtain in return a key instead of the label, the item must have the form: *`key`*`:`*`value`*.

It is accepted the symbolic names `%days`, `%#days`, `%months` and `%#months` (case insensitive); the symbolic name with `#` returns the number (starting from `1`), `days` starts from `Monday`.

Examples:

```
CMB combo 'Combo example' A:Alfa|B:Beta|Δ:Delta|Γ:Gamma
CMB months 'Months' %Months default June
```

1.3.5 Comment

The label field is the comment shown:

`Comment|C` *`fieldName 'some comment'`* `[color` *`color`*`] [left|center|right]`

`Comment` and `C` are synonym.

☞ *`color`* can be in hexadecimal `RRGGBB` notation (like `#FF0000`) or a symbolic name ☞ (a list of accepted color names is, my be, somewhere in the GTK documentation).

1.3.6 File and Folders

`F|File` *`fieldName label`* `[filter` *`filter`*`(s)] [title` *`title`*`] [default|value` *`defaultValue`*`] [hint` *`hintText`*`]`

`Folder` *`fieldName label`*

```
File FileIn '' filter '*.jpg,*.png,*.gif' title 'Take a graphic file'
```

1.3.7 Hidden field

The *label* field contains the value:

`H|HIDDEN` *`fieldName`* `[`*`value`*`]`

The value can also be set by `Default` pseudo type.

```
date = Dates.format(now(), "yyyy-mm-dd HH:MM:SS")
...
Window 'Text example'
T Text
H Timestamp
H hidden 'A hidden field'
Menu Info Info 'Sandbox $version'
U notModifiable '' 'Not modifiable field'
Default Timestamp '$date'
```

1.3.8 Navigator

`Navigator|NV` *`fieldName function`* `[after` *`fieldName`*`]`

The *`function`* receives the *`fieldName`* and the button name id est `Next` or `Previous`.

Example 2: Use of Navigator

```
frmEvents = """
Window 'Events' background #00C0C0 onStart loadAuthors
Comment c 'Quote' color green height 200 Ground #E0E0E0
Navigator browse browseQuote
COMBO Authors "" onChange loadCite
C Info
"""
                        function browseQuote(name,button)
  global nQuote,Author
  nQ = length(authorQuotes)
  if nQ > 0
    incr = button != "Next" ? -1 : 1
    nQuote = rem(nQuote+incr+nQ,nQ)
    formgen.setValue("c",authorQuotes[nQuote+1])
    formgen.setValue("Info","$Author quotes $nQ quote n° "*string(nQuote+1))
  end
end
```

1.3.9 Radio buttons

`Rdb` *`fieldName label valueList`* `[vertical]`

The *extra* field contains the labels and value of each radio button (see 1.3.3 Chek box list). To obtain a key instead of the label, the item must have the form: *`key`*`:`*`value`*.

```
Rdb Gender '' 'M:Male|F:Female|N:Not exposed'
Rdb Liking '' 'L:Low|M:Medium|H:High' vertical
```

If no radio Buttons are checked the value exists and it is the empty string. More than one set of radio buttons can be present in the form.

1.3.10 Slider

The *extra*(s) fields can contains the `start` and `end` values in the form `start end`, for example `-5 5`; if the values are omitted, they are assumed equals to `0 100`. The result can have decimals depending on the difference from `start` and `end` value, see table at right.

☞ The slider has always a value, i.e. the implicit default value is set to the medium point.

☞ Limits must be integers!

start - end	**n. decimals**
`> 99`	`0`
`< 100 and > 10`	`1`
`< 10 and > 1`	`2`

1.3.11 Text fields

`P|NP|N|NF` *`fieldName label`* `[default|value` *`defaultValue`*`] [hint` *`hintText`*`]`

`U|UN` *`fieldName label`* `[default|value` *`defaultValue`*`]`

`H` *`fieldName value`*

`T` *`fieldName label`* `[default|value` *`defaultValue`*`] [hint` *`hintText`*`] [height` *`value`*`] [width` *`value`*`] [onactivate` *`function`*`|submit]`

`TEXT` and **`PSW`** are synonym of **`T`** and **`P`** respectively; numeric fields are **`NP`** (unsigned integer), **`N`** (signed integer) and **`NF`** (signed number with possibly decimals).

The **`U`** and **`UN`** types are a not modifiable texts; ☞ their values can be set by `default` or `value` parameter or set by pseudo type `default`.

The **`H`** type is a hidden text the value is the *label* field or can be set by pseudo type `default`.

☞ If the Text widget contains `height` parameter the widgets is realized by a text area into a scroll view.

☞ texts (`GtkEntry`) minimum-width is hard coded to 150px.

1.4 Pseudo types

1.4.1 Controls and Required

There are two pseudo types for controls fields: `Control` and `Required` or `Req`.
`Control` is used to perform controls on the data. The structure for this command are:

```
Control fieldName is mail|regExpression errorSignal
Control fieldName operand value|fieldName| errorSignal
```

where *`operand`* `:= ==|!=|<|<=|>|>=`

The syntax of Required is: `Req[uired]` *`fieldName`*$_1$`[` *`fieldName`*$_2$`[ ...]]`

☞ If a field has more controls they are in `and`.
Examples:

```
Required naturalNumber waitingTime
Control waitingTime <= 1000 'waiting too long'
```

The possibly `control`(s) are:

Type	**Value**	**Note**
`required`		by pattern `.+`
`Mail (*)`	none	Controlled by pattern `/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/`
`Pattern (*)`	a regular expression	
~~`function`~~	~~a Julia function~~	~~The parameters of function are: the *`form`*, the *`fieldName`* and the *`value`* (see example below), the function must returns `true` or `false`.~~

`(*)` Control aren't executed if the field is empty.

1.4.2 Defaults

The type `Default` is used for populates the form; the syntax is:

```
Default[s] ctrlName value[ ctrlName value[ ...]]
```

- The possible value for check boxes is `on`.
- Values for combo boxes and radio buttons must be the *`key`* (remember that if *`key`* is omitted the *`value`* itself is a *`key`*.
- If the value is constant (like `on`, `off` etc.) it is case insensitive.

When the `Reset` button is pushed the form is restored with the default values.

1.4.3 Event

This pseudo type is used to attach an event handler to a field, the syntax is:

```
Event|On widgetName event call function|submit|'function parameter'
```

Some events are automatically generated:

`Widget`	**`Parameter`**	**Note**
`buttons`	`after` *`field`* `call` *`function`*	
`text`	`onactivate` *`function`*	Not for text view
`combo`	`onchange` *`function`*	

1.4.4 Label

```
Label [left|center|right] [after afterValue] [color color]
```

This pseudo type formats the label presentation; the *`aftervalue`* are characters appended to the label. Example `label color olive after ' :'`

1.4.5 Menu

Add menu in menu bar:

```
Menu father son function|displayData|close|exit
```

Note that the order is order of presentation, in fact every item is a terminal menu item and the program deduces the structure.

```
Menu _File Close Close
Menu _Info About About
```

Note the _ tells that the menu can be accessed by `Alt F` and `Alt I` respectively.

1.4.6 Style

This pseudo type can set a style to a widget:

```
Style widgetName style
```

Example:

```
frmStyles = """
window 'Style example' ground teal
label right after : color yellow
CMB months '' %Months
CKB Agree '' 'Consent sending infos'
Style Agree 'border:3px outset olive;background-color:silver'
Style months 'border:3px outset olive;border-radius:3px'
"""
```

1.4.7 Window

The type `Window` is used to tell how the form is treated when it is submitted; the syntax is:

```
Window title [reset] [static] [ground|background color] [Start|onStart
function] [End|onEnd function]
```

The `Start` *`function`* is called before the show of form; the `End` *`function`* is called before the close of the form, if the *`function`* returns `false` the form is not close and the callback function is not invoked.

☞ The window is erased by `Cancel` button. The `Ok` button cancels the form unless it has the `static` parameter; if the form has the `reset` parameter it is cleared.

The window handle is exposed with name `fg_window`.

Before the submission is called the function for the `End` event if exists and the data are controlled as indicated in the pseudo type `Control` or `Required` (if they exist), in case of error(s) the form is not submitted and the label of field(s) in error are on red[1] or prefixed by `***` (for Windows systems).

1.5 Data presentation

The data are presented in the order they appears in the parameters list, except for the buttons that appears together the buttons inserted by *FormGen* at the bottom of the form or, possibly, at right of a control.

The buttons inserted automatically (*standard buttons*) are `Ok`, `Cancel` and `Reset` button, they have the name respectively `Ok`, `Cancel` and `Reset`; ☞ there are no buttons if there is only one Combo box,

1.5.1 Styling

Some styling is incorporated into the widgets syntax and furthermore it can be enhanced by the pseudo type `Style` or by the function `fg_set_gtk_style` see paragraph 1.9.6 Set widget style, by means of the functions invoked in the pseudo type events and especially the `Start` event.

---

[1] If the fields label are colored this is not possible.

Another possibly styling is setting some widget properties by means of the `fg_set_gtk_property` see paragraph 1.9.5 Set widget property.

The above functions access the widget by the widget name or the widget handle, the latter is accessed in the `wdgHandles` dictionary by the widget name itself.

Example 3: On start function and some stylings

```
function setOnStart(event)
  TextView = formgen.wdgHandles["TextView"][2]
  ccall((:gtk_text_view_set_monospace,libgtk),Cvoid,
(Ptr{GObject},Cuchar),TextView,true)
  formgen.fg_set_gtk_property("fg_window",:resizable,false)
end
```

1.6 On window closing

When the `Ok` button is pressed and there aren't errors the *`callBackEvent`* function is invoked with a Dictionary of form data, where the key is the *`fieldname`* and the values are `String` or `Float64` depending on their type (☞ however, if the numeric field has not been entered, an empty `String` is returned).

Moreover the Dictionary has the field `fg_Button` that contains the name of the button which has submitted the form or, in case of single combo is the name of the field.

When the `Cancel` button is pressed the *`callBackEvent`* function is invoked with a Dictionary containing only `fg_Button`.
The `Cancel` button clears the form container; the `Reset` button cleans the form and restores the defaults values.
☞ The *close* button on the title bar doesn’t returns nothing.

1.7 Parameters of callback functions

On closing form	Dictionary of the data on form: *`fieldname`* `=>` *`value`*
Buttons after fields	Button name, field name
Navigator	Field name associated, `Next|Previous`
On/Event pseudo type	Field name associated
`onChange` and `onActivate`	Field name associated
`OnStart`	`Start`
`OnEnd`	`End`

1.8 Errors

1.8.1 Comments errors

`Unknown` *`the uninterpreted widget`*	when the widget type was not recognized

1.8.2 Console logged errors

`Default date` *`date`* `incorrect`	Default value of Date type
`Event on field that doesn't exists:` *`fieldName`*	Pseudo type `Event`
`Main doesn't contains` *`functionName`* `function`	Events and button events

1.9 Some functions

`formGen` contains some functions that can be accessed by `formgen.`*`functionName`*`(...)`.

1.9.1 Change the contents of combo box

The function `createOptions` permits to populate or replace a combo box list contents.

`createOptions(ComboName,newData)`

ex. `formgen.createOptions("Unit","in:Inch|ft:Feet|yd|Yard|Mile")`

1.9.2 Create Calendar

`formgen.fg_create_calendar(receiverFunction[,value|tomorrow][,format|Y-m-d])`

```
g = GtkGrid()
...
g[1,1] = formgen.fg_create_calendar(receiveDate,"tomorrow","E d U Y")
g[1,3] = formgen.fg_create_calendar(receiveDate)
g[2,3] = formgen.fg_create_calendar(receiveDate,"2023-01-30")
...
```

The function returns the calendar handle.

In the absence of the *`value`*, the date shown is today's.

`macro defaultDateFormat() return "Y-m-d" end`

1.9.3 Create Menu

This function can be used independently of the generation of a form:

`menuHandle = formgen.fg_create_menu(menuList)`

where *`menuList`* is an array wehre the items have the form *`father son`* `[`*`function`*`|`*`message`*`]`.

The function returns a `GtkMenuBar` object.

Example 4 Menu

```
...
menu = [["File", "Parameters", "showParameters"],
["File", "End", "btnClose_Action"],
["Help", "Show commands", "btn_About"],
["Help", "About", "El Condor\nwww.condorinformatique.com"]]
g = GtkGrid()
g[1,1] = formgen.createMenu(menu)
win = GtkWindow( "toolbar", 200, 100)
push!(win, g)
showall(win)
...
```

1.9.4 Create Textview

`[handlers] = fg_create_textview(;width=0,height=0)`

Where *`handlers`* is an array containing the handlers of:

- Text buffer,
- text view,
- scroll vindow.

Example 5 Create Text View and iconic button

```
using GTK3_jll
const libgtk = libgtk3
using Gtk, Gtk.ShortNames
include("formgen.jl")
g = GtkGrid()
global fg_win = GtkWindow( "Try Text view")
btn = GtkButton("\u22EF")
sc = GAccessor.style_context(btn)
```

```
pr = CssProviderLeaf(data="#$name {background:#C0C0C0;border-width:2px}")
push!(sc, StyleProvider(pr), 600)
pr = CssProviderLeaf(data="#$name:hover {background:#EEEEEE}")
push!(sc, StyleProvider(pr), 600)
pr = CssProviderLeaf(data=".text-button {font: 24px \"sans\"}")
push!(sc, StyleProvider(pr), 600)
id = signal_connect(btn, "button-press-event") do widget, event
   println(get_gtk_property(textBuf, :text, String))
   println(get_gtk_property(textBuf2, :text, String))
end
g[1,2] = btn
handlers = formgen.fg_create_textview(height=200,width=350)
textBuf2 = handlers[1]
ccall((:gtk_text_view_set_monospace,libgtk),Cvoid,
(Ptr{GObject},Cuchar),handlers[2],true)
set_gtk_property!(textBuf2,:text,"Inserted text")
g[1,1] = handlers[3]
push!(fg_win,g)
showall(fg_win)
```

1.9.5 Set widget property

```
fg_set_gtk_property(widget::String, name::Core.Symbol, value::Any)
fg_set_gtk_property(widgetName, property, value)
fg_set_gtk_property(widget::Gtk.GtkWidget, name::Core.Symbol, value::Any)
fg_set_gtk_property(widgetHandle, property, value)
```

1.9.6 Set widget style

```
fg_set_gtk_style(widget, style[,value])
```

Example 6: Set widget style

```
commentStyle = "* {border:3px inset;margin:5px;padding:3px;background:cyan}"
formgen.fg_set_gtk_style(formgen.wdgHandles["c"],commentStyle)
```

2 History

0.1.1	January 2023	Started

3 Technical notes

3.1 Software versions

- Gtk Gtk3
- Julia REPL 1.8.1, 1.9.0 Window 11
- Julia 1.8.5 Ubuntu in Virtual Box

3.2 Events

A form is created with some events added:

`activate`	Menu, Enter key of Text
`button-press-event`	`Ok`, `Cancel` and `Reset` buttons
`changed`	Combo boxes
`clicked`	`After` buttons
`key-press-event`	For numeric texts
`key-release-event`	For numeric texts
`toggled`	Radio buttons

Moreover events can be added by the `Event` or `On` pseudo type.

3.2.1 Blocking event

`signal_handler_block(`*`widgetHandler`*`,`*`eventId`*`)`

Example 7: Block event

```
wdgEventsId[field[2]] = signal_connect(wdgHandles[field[2]], "changed")
      do widget
      callFunction = field[4]
      eval(Meta.parse("Main.$callFunction(\"" * field[2] *"\")"))
end
...
cmb = wdgHandles[name]
if haskey(wdgEventsId,name) signal_handler_block(cmb,wdgEventsId[name]) end
```

3.2.2 Unblocking event

`signal_handler_unblock(`*`widgetHandler`*`,`*`eventId`*`)`

3.3 Structures and variables

name	**key**	**value**
`wdg`	`index`	Array of data of widgets
`wdgAfter`	*`fieldName`*	
`wdgControls`	*`fieldName`*	Array of controls
`wdgCombos`	*`fieldName`*	Dictionary that contains dictionary of combo and radio buttons items *`value => key`*
`wdgCombosIndex`	*`fieldName`*	Array of *`values`* (for preserve order)
`wdgDate`	*`fieldName`*	
`wdgDefaults`	*`fieldName`*	Default value
`wdgEvents`	`index`	

`wdgEventsId`	*`fieldName`*	
`wdgHandles`	*`fieldName`*	Widget *`handle`*, also `fg_win` the window form
`wdgHiddens`	*`fieldName`*	*`value`*
`wdgLabelHandles`	*`fieldName`*	*`label handle`*
`wdgRequired`	*`fieldName`*	
`wdgTypes`	*`fieldName`*	*`type`*

```
rdbHandlers = Dict{String,Any}()  # key = fieldNameValue
```

3.4 Widget names

```
fg_lScale
fg_scale
fg_window
```

3.5 Form styling

See

- Supported CSS Properties https://docs.gtk.org/gtk3/css-properties.html
- Overview of CSS in GTK https://docs.gtk.org/gtk3/css-overview.html

3.5.1 Set general style

`provider = fg_setCSS(style)`

Example 8: General styling

```
CSSStyles = """
#fg_button {margin:5px;background-image: image(#ccc)}
#fg_button:hover {background-image: image(#eee)}
#fg_button:active {background-image: image(#ddd)}
#fg_lButton {font: 24px \"sans\";min-width:20px;margin-left:3px;background-
image: image(#ccc)}
#fg_lButton:hover {background-image: image(#eee)}
#fg_lButton:active {background-image: image(#ddd)}
#fg_navigButtons {min-width:10px;margin-left:3px}
#fg_scale {min-width: 180px;}
#fg_lScale {min-width: 140px;}
"""
...
screen_provider = fg_set_CSS(CSSStyles * labelsStyle)
```

3.5.2 Reset general style

`fg_setCSS(provider)`

3.5.3 CSS names

Widgets	Name	Default	Note
Monocharacter buttons	fg_lButton		
labels	fg_label	margin:5px	
comments	fg_comment		
textsArea	fg_textArea		Text with height or width

3.5.4 Setting widget properties

Some properties can be set by the Gtk3 function, others by the CSS statements accepted by the Gtk3 version. In the paragraphs below the properties and CSS set in *formGen* or into *sandbox*.

The property list of a widget can be found in Julia interactive by creating the widget, for example:

```
GtkEntry()
GtkLabel("")
```

3.5.4.1 Check box

Check (active) the check box

```
ckb = GtkCheckButton(field[4])
set_gtk_property!(ckb,:active,uppercase(value) == "ON" ? true : false)
```

3.5.4.2 Combo box

```
cmb = GtkComboBoxText()
str = Gtk.bytestring(GAccessor.active_text(wdgHandles[name]))
return get_gtk_property(wdgHandles[name], :active, Int) > 0 ? wdgCombos[name]
[str] : ""
```

3.5.4.3 Entry

```
txt = GtkEntry()
```

Right alignment

```
set_gtk_property!(txt,:xalign,1)
```

Widget dimension

```
d = Dates.format(Dates.now(),dtFormat)
set_gtk_property!(txt,:max_width_chars,length(d))
```

3.5.4.4 Label

Right alignment

```
alignmentConst = Dict{String,Any}("CENTER"=>0.5,"LEFT"=>0,"RIGHT"=>1)
label = GtkLabel("")
set_gtk_property!(label, :xalign, alignmentConst["RIGHT"])
```

3.5.4.5 Slider

Creation:

```
slider = GtkScale(false, 1:11)
```

whit `true` instead of `false` the slider is disabled.

Exposed value at slider right:

```
  set_gtk_property!(slider,:value_pos, 1)
delta = sliderLimit[2] - sliderLimit[1]
  digits = 1/10^(floor(log10(delta-2))) < 1 ? floor(log10(delta-2)) : 0
  set_gtk_property!(slider,:digits, digits)
```

3.5.4.6 Window

Settable Property

```
set_gtk_property!(win,:opacity,0.5)
set_gtk_property!(win,:title,"backgroundcolor:teal")
```

background color

```
sc = GAccessor.style_context(win)
pr = CssProviderLeaf(data="* {background:#C00000;border-width:2px;margin-left:5px}")
```

also

```
pr = CssProviderLeaf(data="* {background:teal;border-width:2px;margin-left:5px}")
push!(sc, StyleProvider(pr), 600)
```

3.5.4.7 `Button`

```
sc = GAccessor.style_context(ok)
pr = CssProviderLeaf(data="* {background:#C0C0C0;border-width:2px;min-width:80px}")
push!(sc, StyleProvider(pr), 600)
```

3.6 Julia weird behavior

In version 1.8.1, 1.8.3, 1.8.6

```
julia> "A:Alfa|B:Beta|Γ:Gamma|Δ:Delta"[1]
'A': ASCII/Unicode U+0041 (category Lu: Letter, uppercase)

julia> "A:Alfa|B:Beta|Δ:Delta|Γ:Gamma"[1]
'A': Unicode U+0391 (category Lu: Letter, uppercase)

greek = "A:Alfa|B:Beta|Γ:Gamma|Δ:Delta" A:Alfa|B:Beta|Γ:Gamma|Δ:Delta
println(greek)                          a:alfa|b:beta|γ:gamma|δ:delta
println(lowercase(greek))               A:Alfa|B:Beta|Δ:Delta|Γ:Gamma
```

```
greek = "A:Alfa|B:Beta|Δ:Delta|Γ:Gamma" α:alfa|β:beta|δ:delta|γ:gamma
println(greek)
println(lowercase(greek))
```

3.7 Operating system differences

3.7.1 Windows

Problems on Window related to the Gtk interface, the `Gtk.CssProviderLeaf` function is very slow, see below.

```
Window Visual studio Julia 1.8.1 ************************
@time Gtk.CssProviderLeaf(data="#name {background:#C0C0C0;border-width:2px}")
2.046077 seconds (6 allocations: 96 bytes)
GtkCssProviderLeaf()

Window Julia 1.9.0 ************************************
@time Gtk.CssProviderLeaf(data="#name {background:#C0C0C0;border-width:2px}")
GtkCssProviderLeaf: 0.003820 seconds (1 allocation: 64 bytes)
"#name {background:#C0C0C0;border-width:2px}"

Ubuntu in Virtual Box ***********************************
@time Gtk.CssProviderLeaf(data="#name {background:#C0C0C0;border-width:2px}")
  0.000136 seconds (6 allocations: 96 bytes)
GtkCssProviderLeaf()
```

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 few examples.

The regular expression is contained between `//` and can be followed 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, unless it starts with `?:` (see in Examples).

`(?:...)` a non-capturing group,

`?=`*`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*$`	Empty set or white spaces
`(\w+)\s+(\w+)\s+(\w+)`	Find and memorize three words
`(\-[a-z])`	Find and memorize `minus` followed by one alphabetic character
`.(jpg|jpeg)$`	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
`(?=.*\d)(?=.*[A-Z])(?=.*[a-z]).{6,12}`	At most one digit, one capital letter, one minuscule and

	from 6 to 12 characters
`r"\s*(?:ground|background)\s+(\S+)"i`	Extracts the items of any non whitespaces **`(\S+)`** after the parameter name.

`re = r"^\"(.*)\"$\|^'(.*)'$"`
`replace(s,re => s"\g<1>\g<2>")`
	Strip string delimiter (`'` or `"`)

`^(\d{4})\D?(\d\d?)\D?(\d\d?)$\|^(\d{4})(\d\d)(\d\d)$`	Check date in form `AAAAMMDD` or `AAAA?MM?DD` where `?` is a possible separator and month and day can be one character

5 Indexes

5.1 List of Examples