AJUI Button

User manual

Version Control	Date	Commentary (change)	Author
1.0	19.08.2019	First version	Gary Criblez
1.2	28.11.2019	New State « Focus »	Gary Criblez
1.3	19.12.2019	FAB (Floating Action Button)	Gary Criblez

1 Introduction

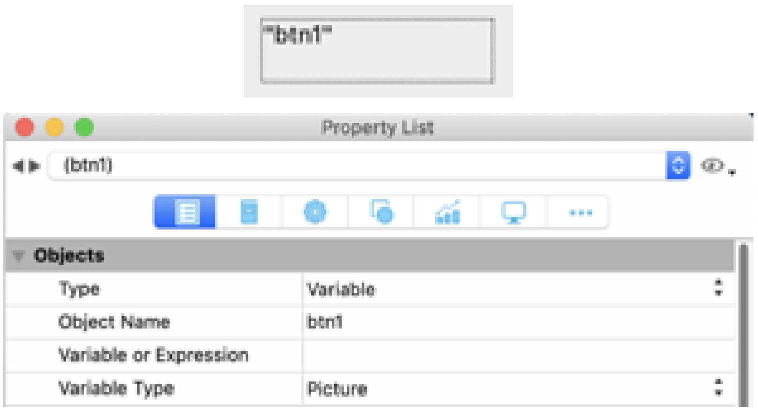
1.1 What is AJUI Button

AJUI Button is a component developed in the 4D language. Its use is intended for 4D developers. It allows you to dynamically generate and display interface buttons in the context of a form.

A button can be defined using an object variable representing an instance of AJUI Button. This variable contains a set of properties and formulas to define the structure and content of a button and generate it in the form.

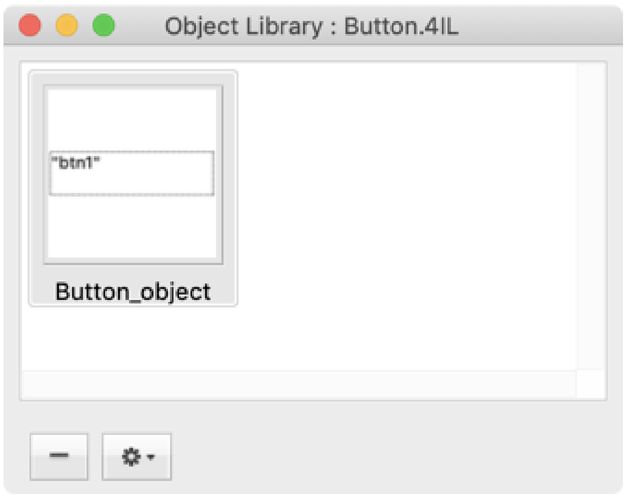
1.2 Button composition

Basically, the button generated by AJUI_Button is a picture type form object variable that you can create yourself in the context of a form.



Type	Variable
Object Name	btn1
Variable or Expression	
Variable Type	Picture

With regard to this point, we provide you with a predefined template of this form object variable containing the appropriate properties for creating an AJUI Button. It is provided in an object library file “Button.4dlibrary” (the library is provided in project version for the v18).



1.3 Button types

AJUI Button offers two main types of buttons.

- Simple button
- Composite button

A simple button is a rectangle whose corners can be rounded or a circle containing text **OR** a picture.

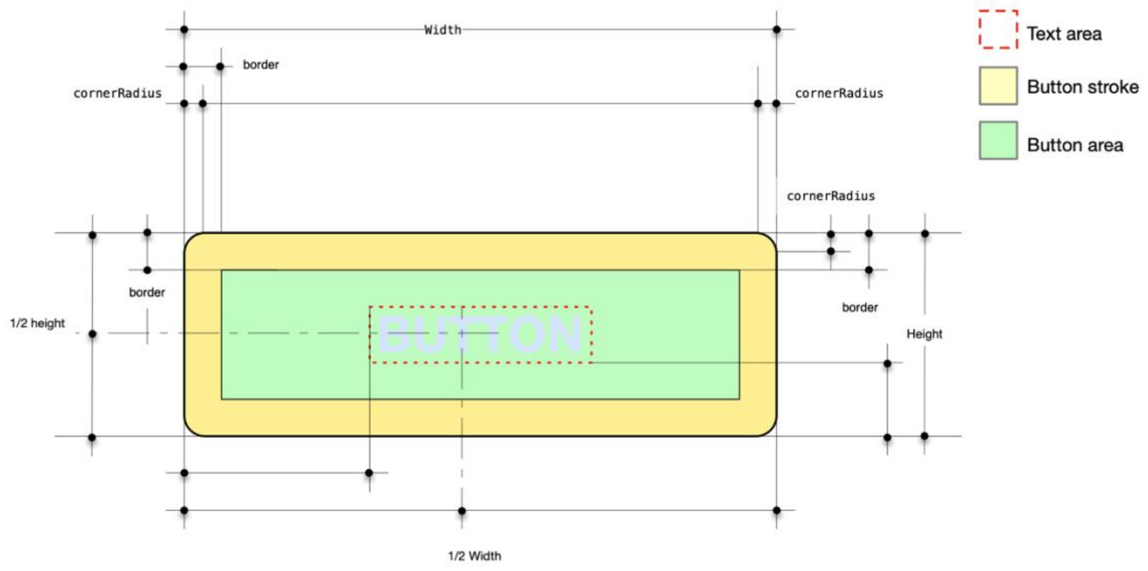
A composite button is a rectangle whose the corners can be rounded or a circle containing text **AND** a picture. The picture and text are placed according to the area allocated to each one, but they can also be padded together. In this case, we speak of a linked composite button. We assign a position to the picture (left, right, top, bottom). The text will be the opposite of this position.

1.3.1 Button Anatomy

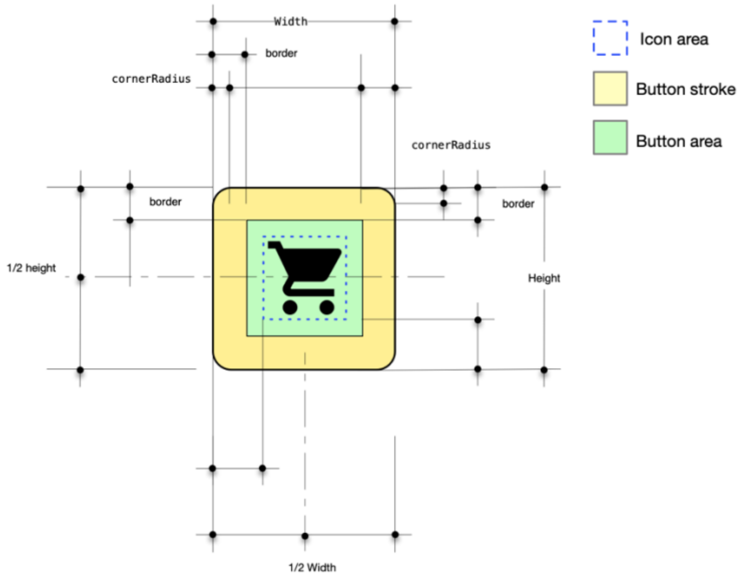
Simple button

A simple button includes a text OR a picture. Its graphic representations and geometric properties are as follows.

Text :



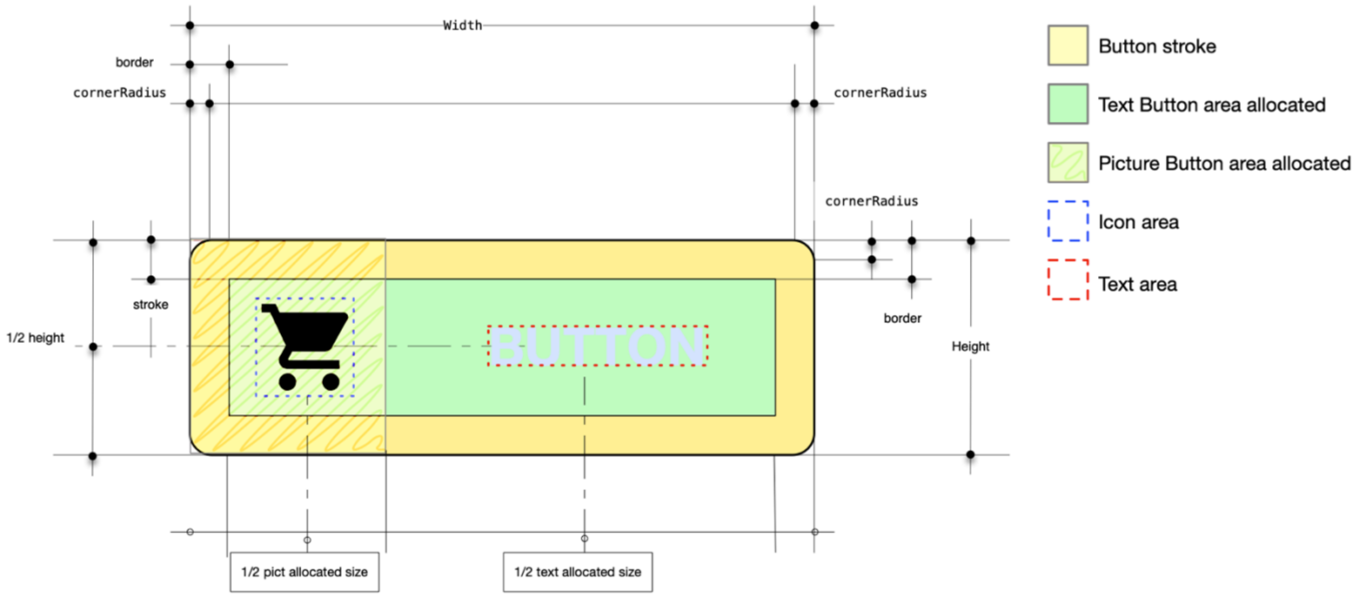
Picture :



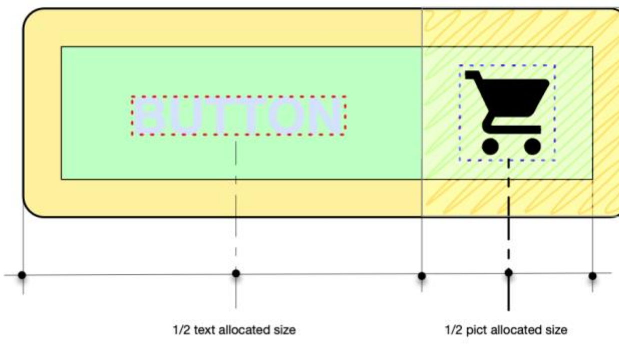
Composite button

A composite button includes a text AND a picture. Its graphic representations and geometric properties are as follows.

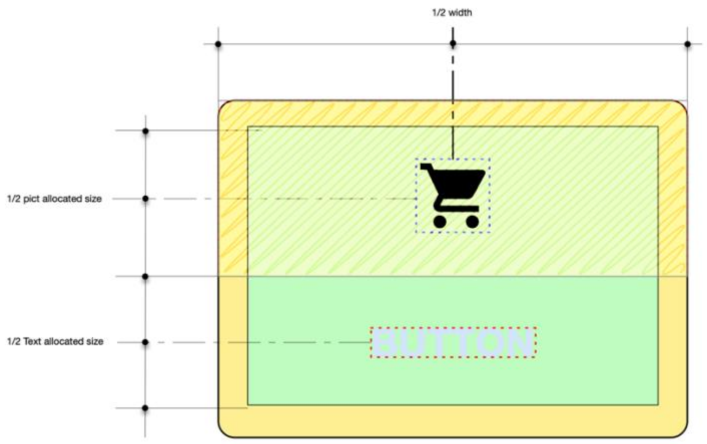
Left position :



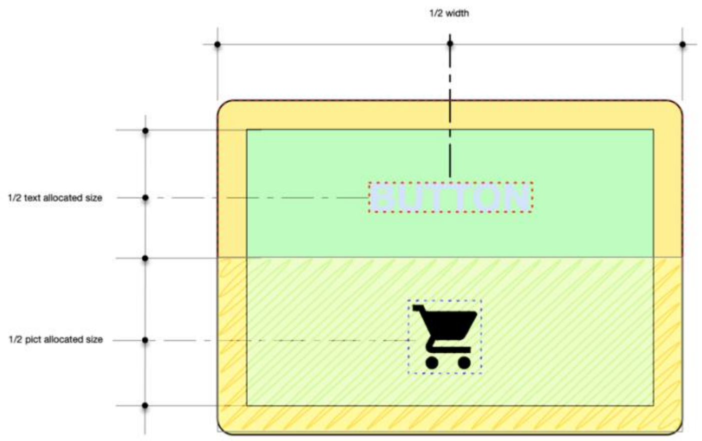
Right position :



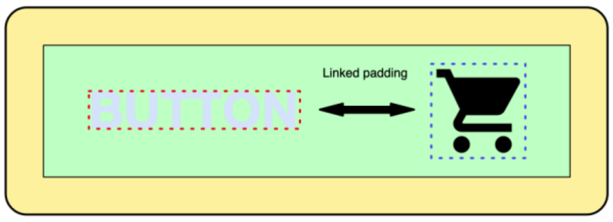
Top position :



Bottom position :



Linked :



1.3.2 Sizing a button

By default, the button generated by the component will match the dimensions defined in the form editor. However, it is possible to redefine them using the member functions of an instance of AJUI Button.

The size of the picture can be defined with the member functions but for consistency, the component will always resize the picture if it exceeds the limits of the button or the allocated size in the case of a composite button.

The text will automatically be shortened and ended with"..." if it exceeds the limits of the button or the allocated size in the case of a composite button.

1.4 States of a button

An AJUI button can take on a different appearance and content depending on five states and the exceptions that will be defined for them.

State « Default »

This is the basic state of a button when a user does not interact with it or when it is not disabled.

State « Hover »

This state occurs when the "On Mouse Enter" event is triggered and ends with the "On Mouse Leave" event. In concrete terms, the user is hovering over the button with the mouse.

State « Active »

This state intervenes on the "On Click" event and ends with the "On Mouse Up. Note that it has priority over the "Hover" state. In practice, the user clicks on the button and the status lasts until it is released.

State « Disable »

This state takes place when the button is deactivated using the object's "Enable" member function. When the button is deactivated, only this state is taken into account.

State « Focus »

This state occurs on the "On Getting Focus" event and ends with the "On Losing Focus”.

1.4.1 Events and Callbacks

As you can see from the states, the component requires that several events be activated on the picture form object in order to be able to manage the different states. Here is the list of these:

- On Load
- On Click
- On Double Click (optionnel)
- On Mouse Enter
- On Mouse Leave
- On Mouse Up
- On Getting Focus
- On Losing Focus

Concerning the two events on click and on double-click, callbacks can be associated with them using the member functions. Basically, the event on the double-click is not necessary unless you use the callback.

Warning : All methods used as callbacks must be shared (method properties) with the component so it can call it. To prevent an error from occurring in such case, the component will check if the method is well shared and if it is not, it will share it itself when executing a callback. In addition, the component will propose to create the callback method if it does not exist when using Setters for callbacks and also before executing a callback.

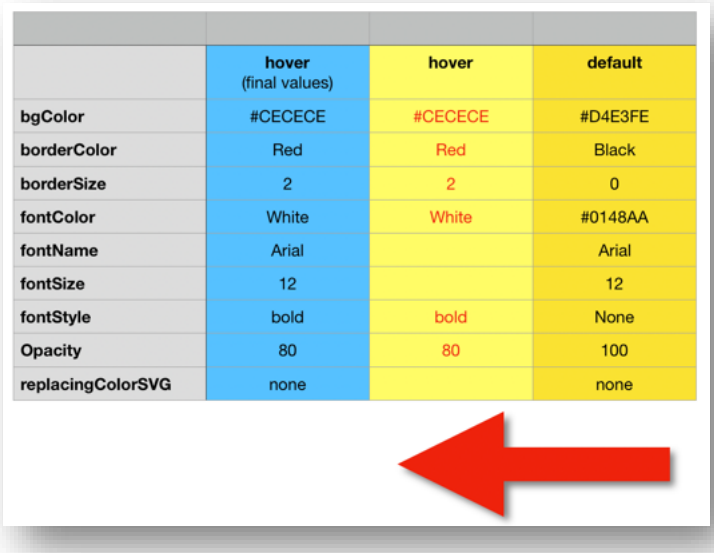
1.4.2 The notion of exception and the constants of the component

In order to be able to modify the structure and content of a button, we have set up an exception mechanism in the component that will apply to all states except "Default" which retains all its basic properties.

To do this, most of the member functions (see the list of properties for details) expect a constant in the first parameter. Each constant represents one of the five states. Here is the list:

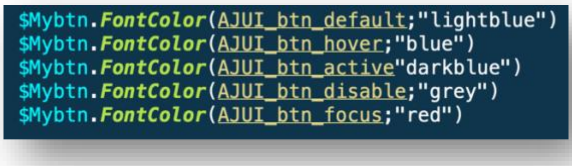
- AJUI_btn_default (no exceptions created)
- AJUI_btn_hover
- AJUI_btn_active
- AJUI_btn_disable
- AJUI_btn_focus

According to the constant that you will pass to a member function, it will create an exception in your AJUI Button instance that will replace the current value (Default) when the state that is linked to it is triggered.



	hover (final values)	hover	default
bgColor	#CECECE	#CECECE	#D4E3FE
borderColor	Red	Red	Black
borderSize	2	2	0
fontColor	White	White	#0148AA
fontName	Arial		Arial
fontSize	12		12
fontStyle	bold	bold	None
Opacity	80	80	100
replacingColorSVG	none		none

Example of assigning a color to the text of the button for each state: :

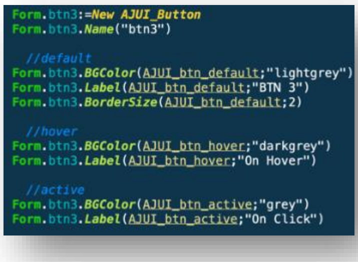


```
$Mybtn.FontColor(AJUI_btn_default;"lightblue")
$Mybtn.FontColor(AJUI_btn_hover;"blue")
$Mybtn.FontColor(AJUI_btn_active"darkblue")
$Mybtn.FontColor(AJUI_btn_disable;"grey")
$Mybtn.FontColor(AJUI_btn_focus;"red")
```

1.5 Life cycle of a button

1.5.1 Phase 1 : initialization and configuration

The first step is to define the properties of the button that will be displayed. To do this, the component provides a "New AJUI_Button" method generating a base object representing a default definition. Then the user is free to use the member functions attached to the object to customize its properties as he likes.



```
Form.btn3:=New AJUI_Button
Form.btn3.Name("btn3")

  //default
Form.btn3.BGColor(AJUI_btn_default;"lightgrey")
Form.btn3.Label(AJUI_btn_default;"BTN 3")
Form.btn3.BorderSize(AJUI_btn_default;2)

  //hover
Form.btn3.BGColor(AJUI_btn_hover;"darkgrey")
Form.btn3.Label(AJUI_btn_hover;"On Hover")

  //active
Form.btn3.BGColor(AJUI_btn_active;"grey")
Form.btn3.Label(AJUI_btn_active;"On Click")
```

This step is very important, because all the other steps will be based on what has been defined in the object.

1.5.2 Phase 2 : Calculations and button generation

The second phase that is the most significant for the component is the creation of the button based on its definition. During this phase, it will be in charge of retrieving the object's properties in order to perform a series of calculations and controls in order to generate/assign the different elements that will compose this set called the button. It is also in this phase that exceptions related to the state are processed. If a callback is assigned and the event corresponds to it, it will be executed at the end of the generation.

1.5.3 Phase 3 : Showing and Hiding

At the end of the generation, the picture is assigned to the picture form object that has been associated with it. The button display is managed using the "Show" and "Hide" member functions or using 4D methods to manage form objects.

2 Component methods

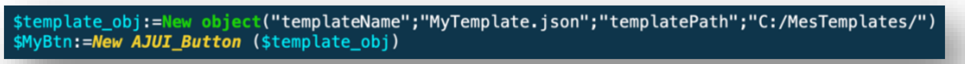
2.1 New AJUI_Button

New AJUI_Button ({ template_obj}) -> button_obj

This method returns an object variable that represents an instance of AJUI Button. It contains all the properties and their default values as well as the formulas (member functions) to manipulate them. It is possible to pass an object as a parameter to it in order to import an AJUI Button template (JSON file). The object expects as properties:

- *templateName* : Corresponds to the name of the JSON file to import (template). If the file is not found, the method will return a new instance of AJUI Button.

- *templatePath* (optional): You can specify a path to retrieve the file otherwise, the component will search in the default folder located in the resources (…/Resources/AJUI_Button_Templates/).

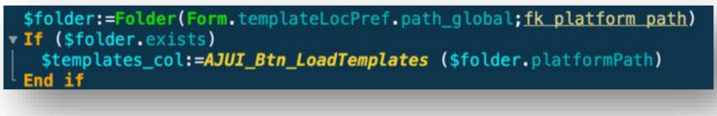


```
$template_obj:=New object("templateName";"MyTemplate.json";"templatePath";"C:/MesTemplates/")
$MyBtn:=New AJUI_Button ($template_obj)
```

2.2 AJUI_Btn_LoadTemplates

AJUI_Btn_LoadTemplates ({folderPath}) -> templates_name_col

Utility method to get all JSON files from a folder into a collection. You can pass in parameter the path of the folder containing the templates. If this is not the case, the method will search in the default path (…/Resources/AJUI_Button_Templates/).

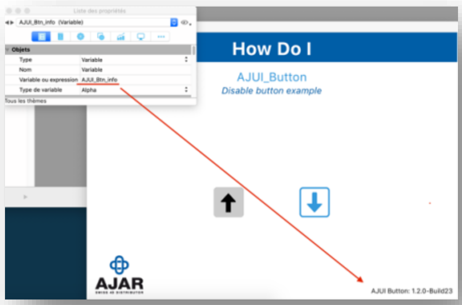


```
$folder:=Folder(Form.templateLocPref.path_global;fk platform path)
If ($folder.exists)
  $templates_col:=AJUI_Btn_LoadTemplates ($folder.platformPath)
End if
```

2.3 AJUI_Btn_info

AJUI_Btn_info ( ) -> version_txt

This method returns a string representing the version number of the component.



Liste des propriétés

AJUI_Btn_info (Variable)

Objets

Type: Variable

Nom: Variable

Variable ou expression: AJUI_Btn_info

Type de variable: Alpha

Tous les thèmes

AJUI Button: 1.2.0-Build23

3 List of properties

A button has a set of properties that will define its representation and interactions with the host form. In this chapter, we will review the different existing properties accessible by a formula that acts as **Setter** and also as **Getter** if no parameter is passed to them (the parameter corresponding to the state remains mandatory). All formulas can be called up at the first level of the object.

Warning : concerning the formulas requiring the state of the button, this one is mandatory, only the second parameter is optional to realize a **Getter** in this case.

For color-related properties, you can give them a color code in hexadecimal (e. g. 0x00FFFFFFFF) or as CSS strings (e. g. "black", "#0A509E").

It is recommended not to manually modify properties whose values are not defined by formulas because they are managed internally by the component.

3.1 Global

Global properties at the second level of the object : **btn.**global

Name	Type	Description	By default	Formula
composition	string	Button composition : text, picture, composite.	text	Composition
name	string	Name of the button and therefore of the picture form object.	Empty string	Name
onClickCB	string	Name of the callback method on the event « On Click ». You can pass in the second parameter an object that contains the callback parameters.	Empty string	OnClick
onDoubleClickCB	string	Name of the callback method on the event « On Double Click ». You can pass in the second parameter an object that contains the callback parameters.	Empty string	OnDoubleClick
type	string	Button type (shape) : rectangle, circle.	rectangle	Type

Global – Formulas and parameters

Formula name	Parameter(s)
Composition ({composition})	- Button composition (string)
Name ({name})	- Button name (string)
OnClick ({callback {; parameters}})	- Callback on click (string) - Parameters (object)
OnDoubleClick ({callback {; parameters}})	- Callback on double-click (string) - Parameters (object)
Type ({type})	- Button type (string)

3.2 Box

Properties related to the container on the second level of the object : **btn.**box

Name	Type	Description	By default	Formula
bgPrimaryColor	string	Main button color. It is possible to define an opacity rate (%). (E.g.: black:80 - black with 80% opacity)	#4d9ad4	BGColor
cornerRadius	longint	Value of the radius applied to the four corners of the buttons.	5	CornerRadius
height	longint	Button height. The value -1 takes the original height of the picture	-1	Height
resize	boolean	If true, it will take into account the values passed in height and width. If false, it takes into account the dimensions of the picture object in the form editor.	False	Resizable
width	longint	Button width. The value -1 takes the original width of the picture	-1	Width

Box – Formulas and parameters

Formula name	Parameter(s)
BGColor (state {; color})	- Button state (constant) - Primary color (string)
CornerRadius (state {; size})	- Button state (constant) - Corner radius (longint)
Height (state {; height})	- Button state (constant) - Button height (longint)
Resizable ({resize})	- Resize form object (boolean)
Width (state {; width})	- Button state (constant) - Button width (longint)

3.3 Border

Border properties at the second level of the object : **btn.border**

Name	Type	Description	By default	Formula
color	string	Border color. It is possible to define an opacity rate (%).	#0a509e	BorderColor
size	longint	Border size.	0	BorderSize

Border – Formulas and parameters

Formula name	Parameter(s)
BorderColor (state {; color})	- Button state (constant) - Border color (string)
BorderSize (state {; size})	- Button state (constant) - Border size (longint)

3.4 Text

Properties related to text elements on the second level of the object : **btn.**text

Name	Type	Description	By default	Formula
fontColor	string	Text color. It is possible to define an opacity rate (%).	white	FontColor
fontName	string	Name of the font to use. Several fonts can be entered by separating them with a comma without spaces. The component will search for the first usable font in relation to your OS.	Arial, Helvetica, MS Sans Serif	FontName
fontSize	longint	Text size.	16	FontSize
fontStyle	string	Text style (Bold, Italic, Underline, Strikethrough).	Empty string	FontStyle
label	string	Text to be displayed.	Label	Label
marginHorizontal	longint	Horizontal margin applied to the text to add a gap when it is aligned to the left or right. If the text is centered, it can be compressed. The margin applies to each side of the text.	0	TextMarginHorizontal

Text – Formulas and parameters

Formula name	Parameter(s)
FontColor (state {; color})	- Button state (constant) - Text color (string)
FontName (state {; font})	- Button state (constant) - Font name (string)
FontSize (state {; size})	- Button state (constant) - Font size (longint)
FontStyle (state {; style})	- Button state (constant) - Font style (string)
Label (state {; label})	- Button state (constant) - Button label (string)
TextMarginHorizontal (state {; margin})	- Button state (constant) - Horizontal margin (longint)

3.5 Picture

Properties related to picture elements on the second level of the object : **btn.**picture

Name	Type	Description	By default	Formula
colorToReplace	string	Current color of a SVG picture. Value of the tag "fill".	Empty string	ColorToReplace
height	longint	Picture height. The value -1 takes the original height of the picture.	-1	PictureHeight
opacity	longint	Picture opacity rate.	100	PictureOpacity
path	string	Folder path from the "resources" folder. You can also use "#" plus the name of your picture file as a path. This will only work with an imported AJUI Button template and the picture must be located in the same folder as the template.	Empty string	PicturePath
ratio	real	Enlarges/reduces the picture size. It is taken into consideration instead of the properties height and width. The value 1 disables the consideration of the ratio (1 = current picture size).	1	PictureRatio
replacingColor	string	Color to replace the current color in an SVG picture.	Empty string	ReplacingColor
scale	longint	Resizing the picture according to the size of the button according to a percentage. Overrides all other properties that change the size. The value zero disables this operation. The value must be between 0 and 100	0	PictureScale
width	longint	Picture width. The value -1 takes the original width of the picture	-1	PictureWidth

Picture – Formulas and parameters

Formula name	Parameter(s)
ColorToReplace(state {; couleur})	- Button state (constant) - Old color (string)
PictureHeight (state {; height})	- Button state (constant) - Picture height (longint)
PictureOpacity (state {; percent})	- Button state (constant) - Opacity of the picture (longint)
PicturePath (state {; path})	- Button state (constant) - Path of the picture file (string)
PictureRatio (state {; ratio})	- Button state (constant) - Ratio of the picture size (real)
ReplacingColor (state {; couleur})	- Button state (constant) - New color (string)
PictureScale (state {; percent})	- Button state (constant) - Scale of the picture (longint)
PictureWidth (state {; width})	- Button state (constant) - Picture width (longint)

3.6 Composite

Properties related to composite elements at the second level of the object : **btn.composite**

Name	Type	Description	By default	Formula
activeSecondColor	boolean	Active second color.	false	CompActiveSecondColor
bgSecondaryColor	string	Secondary color for composite. It is possible to define an opacity rate (%) (example: black:80 - black with 80% opacity).	#9fddf9	BGSecondaryColor
linked	boolean	Binds the text with the picture and applies a padding between them (secondary color is not active in this case).	false	CompLinked
padding	longint	Padding between the picture and the text if linked.	10	CompPadding
picturePosition	string	Position of the picture in the button (only applicable for composites). Available position : *left, right, top, bottom*	left	CompPicturePosition
pictSizeAllocation	longint	Space (in px) allocated to the picture in proportion to the text. This is used to resize the picture if it exceeds the button limits. This value will be applied to the height or width depending on the position of the image in relation to the text.	30	CompPictSizeAllocation
textAlign	string	Text align in the composite button (*left, center, right*).	center	CompTextAlign

Composite – Formulas and parameters

Formula name	**Parameter(s)**
CompActiveSecondColor ({activate})	- Activate second color (boolean)
BGSecondaryColor (state {; color})	- Button state (constant) - Secondary color (string)
CompLinked ({linked})	- Bind text and picture (boolean)
CompPadding (state {; padding})	- Button state (constant) - Padding between text and picture (longint)
CompPicturePosition (state {; position})	- Button state (constant) - Image position (string)
CompPictSizeAllocation (state {; percent})	- Button state (constant) - Size allocated to the picture (longint)
CompTextAlign (state {; align})	- Button state (constant) - Text align (string)

4 Member functions not related to a property

List of member functions not attached to a property.

Draw ( ) : Generates the button that corresponds to the current state.

Enable (isActive) : Makes the picture form object active/inactive and switches the button to "disable" state if inactive. Expects a Boolean as a parameter. Automatically launches the "Draw" method.

Export (templateName { ; TemplateFolderPath}) : Allows you to create a template of your AJUI Button object in a file in JSON format. The formula requires in the first parameter the name associated with the template (file name). A second optional parameter allows you to specify the folder path, otherwise it will be exported to the default folder (.../Resources/AJUI_Button_Templates/).

Hide ( ) : Hide the button.

RemovePropertyException (formula ; state) : allows you to delete an exception related to a property. The name of the formula related to the property must be set as the first parameter. The second parameter corresponds to the state.

RemovePropertyException (formula) : Allows you to delete all exceptions related to a property. The name of the formula related to the property must be set as the first parameter.

ResetAllStates ( ) : Allows you to delete all the exceptions.

ResetState (state) : Allows you to delete all the exceptions of a state. You must pass the constant corresponding to the state in the parameter (does not apply to the default state).

Show ( ) : Display the button.

Note that the Enable, Hide and Show member functions are actually wrappers for both 4D methods "OBJECT SET VISIBLE " and "OBJECT SET ENABLE". You can use them directly if you like.

5 Getting started

5.1 Installation

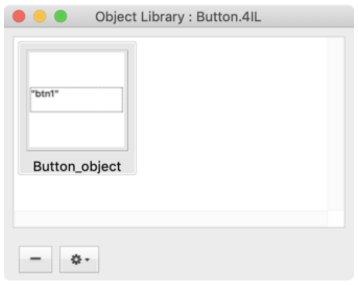
AJUI Button must be placed in the component folder of your application.

5.2 Principles of use

In this section, we describe the sequence of operations to be performed in order to generate a button in the context of the main form.

5.2.1 Prerequisites

AJUI Button needs a picture object that will serve as a model. This object is provided with the "Button.4IL" library that you can open from your application (/File/Open/Object Library...) and you just have to drag and drop the "Button_object" object on your form. In the case of the v18 of 4D, you must use the project version of the library (only project mode)



You can duplicate this object to create as many buttons as necessary. Each button should have a specific name that you can define according to your imagination.

If you create the button by yourself, do not forget to activate the events required to operate the button (see the chapter on events).

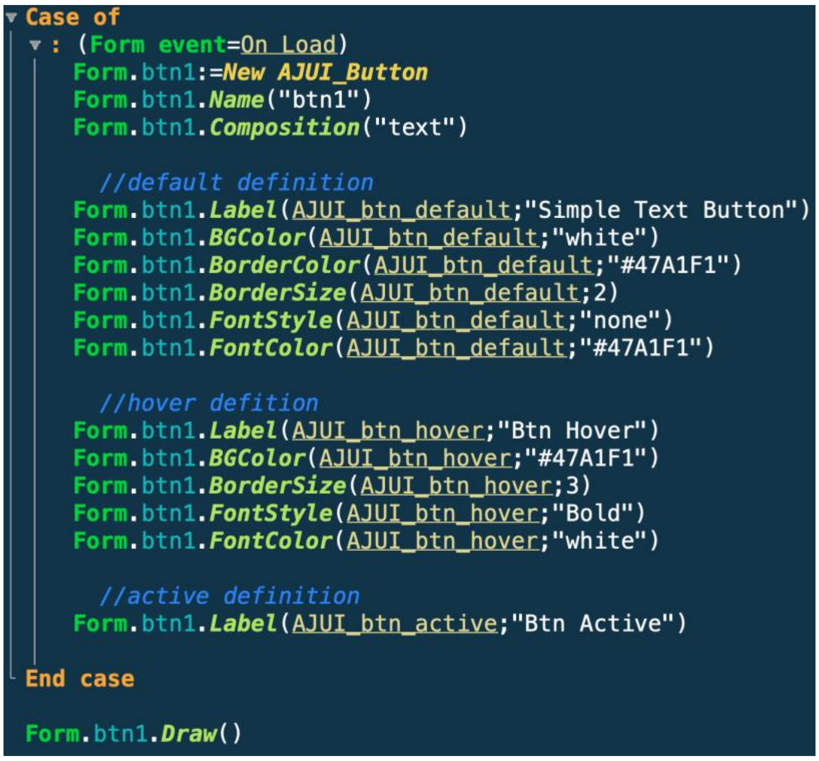
5.2.2 Basis for creating a button

The first thing to do when creating a button is to open the method of your picture form object and create an instance with the "New AJUI_Button" method. Then, associate the name of your button to the instance with the "Name" member function. Now the button is created and will use the default values, but you should try to change it.

To do this, simply call the property member functions described in the corresponding chapter. Test the different compositions and types. Also change property values for different states using constants. Finally, encapsulate everything in an "on load" event and launch the "Draw" function.

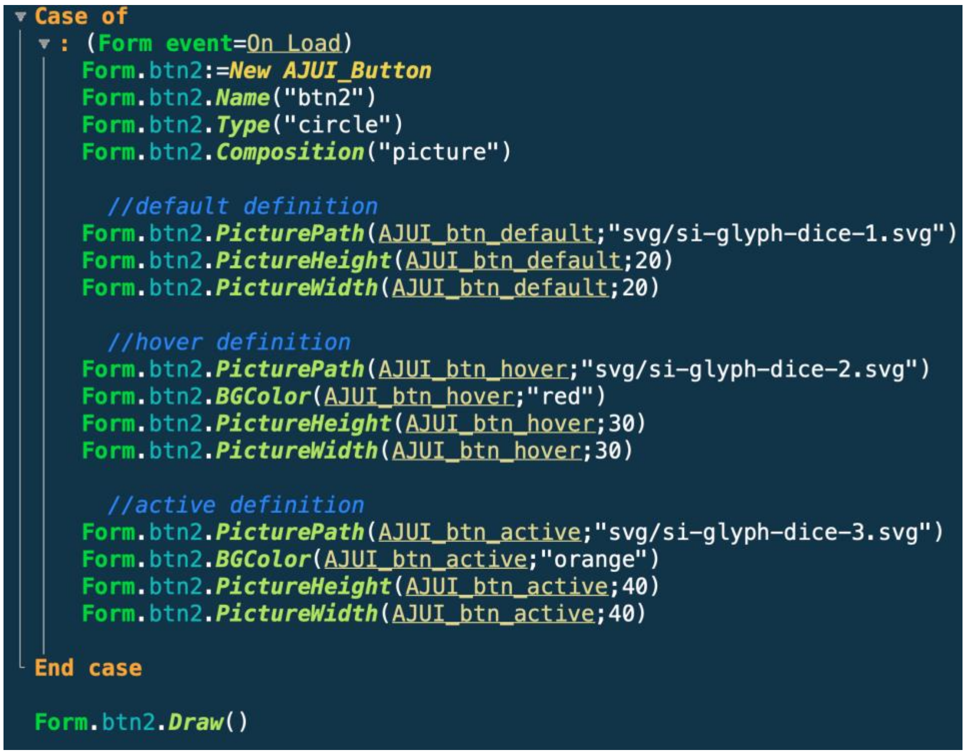
In the following chapter, some examples are presented and we strongly recommend that you study the source code of the "AJUI_ButtonLab" application used as a demo for the component.

Simple button with text



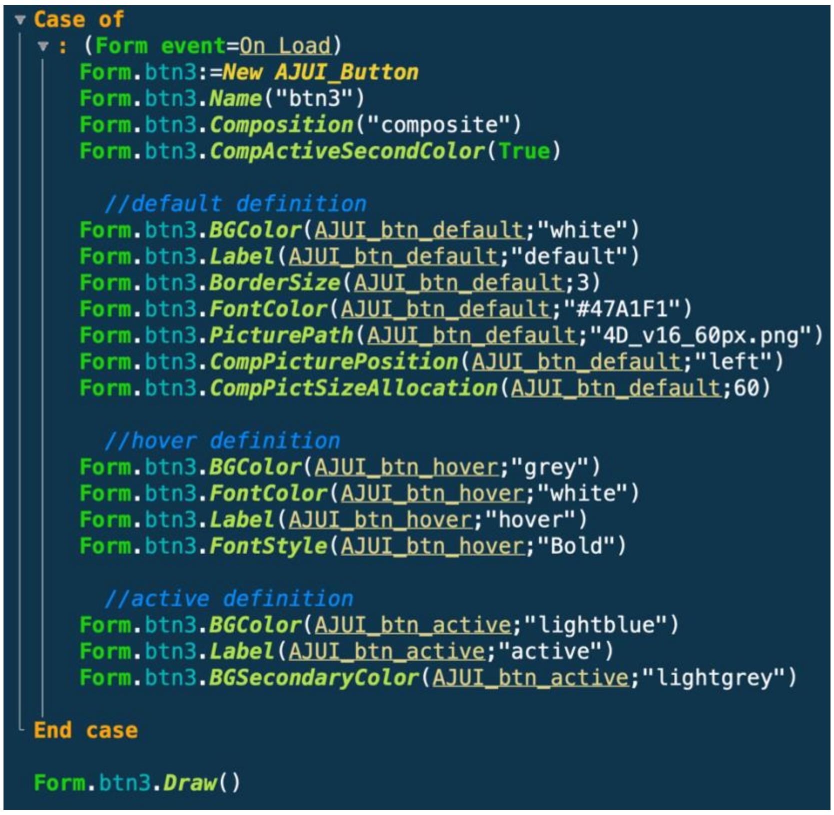
```
Case of
  : (Form event=On Load)
    Form.btn1:=New AJUI_Button
    Form.btn1.Name("btn1")
    Form.btn1.Composition("text")

      //default definition
    Form.btn1.Label(AJUI_btn_default;"Simple Text Button")
    Form.btn1.BGColor(AJUI_btn_default;"white")
    Form.btn1.BorderColor(AJUI_btn_default;"#47A1F1")
    Form.btn1.BorderSize(AJUI_btn_default;2)
    Form.btn1.FontStyle(AJUI_btn_default;"none")
    Form.btn1.FontColor(AJUI_btn_default;"#47A1F1")

      //hover defition
    Form.btn1.Label(AJUI_btn_hover;"Btn Hover")
    Form.btn1.BGColor(AJUI_btn_hover;"#47A1F1")
    Form.btn1.BorderSize(AJUI_btn_hover;3)
    Form.btn1.FontStyle(AJUI_btn_hover;"Bold")
    Form.btn1.FontColor(AJUI_btn_hover;"white")

      //active definition
    Form.btn1.Label(AJUI_btn_active;"Btn Active")

End case

Form.btn1.Draw()
```

Simple button with picture



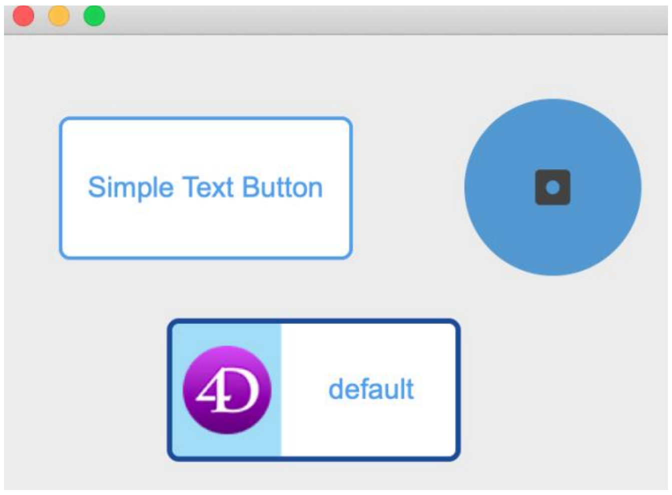
```
Case of
  : (Form event=On Load)
    Form.btn2:=New AJUI_Button
    Form.btn2.Name("btn2")
    Form.btn2.Type("circle")
    Form.btn2.Composition("picture")

      //default definition
    Form.btn2.PicturePath(AJUI_btn_default;"svg/si-glyph-dice-1.svg")
    Form.btn2.PictureHeight(AJUI_btn_default;20)
    Form.btn2.PictureWidth(AJUI_btn_default;20)

      //hover definition
    Form.btn2.PicturePath(AJUI_btn_hover;"svg/si-glyph-dice-2.svg")
    Form.btn2.BGColor(AJUI_btn_hover;"red")
    Form.btn2.PictureHeight(AJUI_btn_hover;30)
    Form.btn2.PictureWidth(AJUI_btn_hover;30)

      //active definition
    Form.btn2.PicturePath(AJUI_btn_active;"svg/si-glyph-dice-3.svg")
    Form.btn2.BGColor(AJUI_btn_active;"orange")
    Form.btn2.PictureHeight(AJUI_btn_active;40)
    Form.btn2.PictureWidth(AJUI_btn_active;40)

End case

Form.btn2.Draw()
```

Composite button

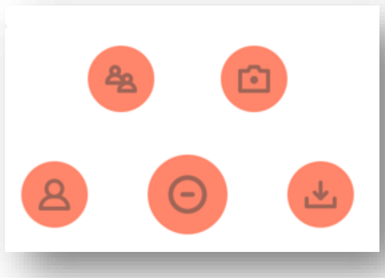


```
Case of
  : (Form event=On Load)
    Form.btn3:=New AJUI_Button
    Form.btn3.Name("btn3")
    Form.btn3.Composition("composite")
    Form.btn3.CompActiveSecondColor(True)

      //default definition
    Form.btn3.BGColor(AJUI_btn_default;"white")
    Form.btn3.Label(AJUI_btn_default;"default")
    Form.btn3.BorderSize(AJUI_btn_default;3)
    Form.btn3.FontColor(AJUI_btn_default;"#47A1F1")
    Form.btn3.PicturePath(AJUI_btn_default;"4D_v16_60px.png")
    Form.btn3.CompPicturePosition(AJUI_btn_default;"left")
    Form.btn3.CompPictSizeAllocation(AJUI_btn_default;60)

      //hover definition
    Form.btn3.BGColor(AJUI_btn_hover;"grey")
    Form.btn3.FontColor(AJUI_btn_hover;"white")
    Form.btn3.Label(AJUI_btn_hover;"hover")
    Form.btn3.FontStyle(AJUI_btn_hover;"Bold")

      //active definition
    Form.btn3.BGColor(AJUI_btn_active;"lightblue")
    Form.btn3.Label(AJUI_btn_active;"active")
    Form.btn3.BGSecondaryColor(AJUI_btn_active;"lightgrey")

End case

Form.btn3.Draw()
```

Result



6 FAB (Floating Action Button)



FABs are buttons floating above the user interface of a form. They act as call to action buttons, intended to represent a single action that users perform most on that particular form screen.

The FAB function we propose will allow you to create and manage this type of floating buttons.

This function allows you to designate an instance as the main button that will contain a collection of other secondary buttons.

Once the roles have been defined, the main button will be assigned to a callback internal to the component allowing it to display or hide the other secondary buttons associated with the collection. When displayed, the secondary buttons will be distributed around the main button reproducing a circle shape. The size, direction and distribution of the secondary buttons on this circle can be defined via the properties linked to the FAB functionality.

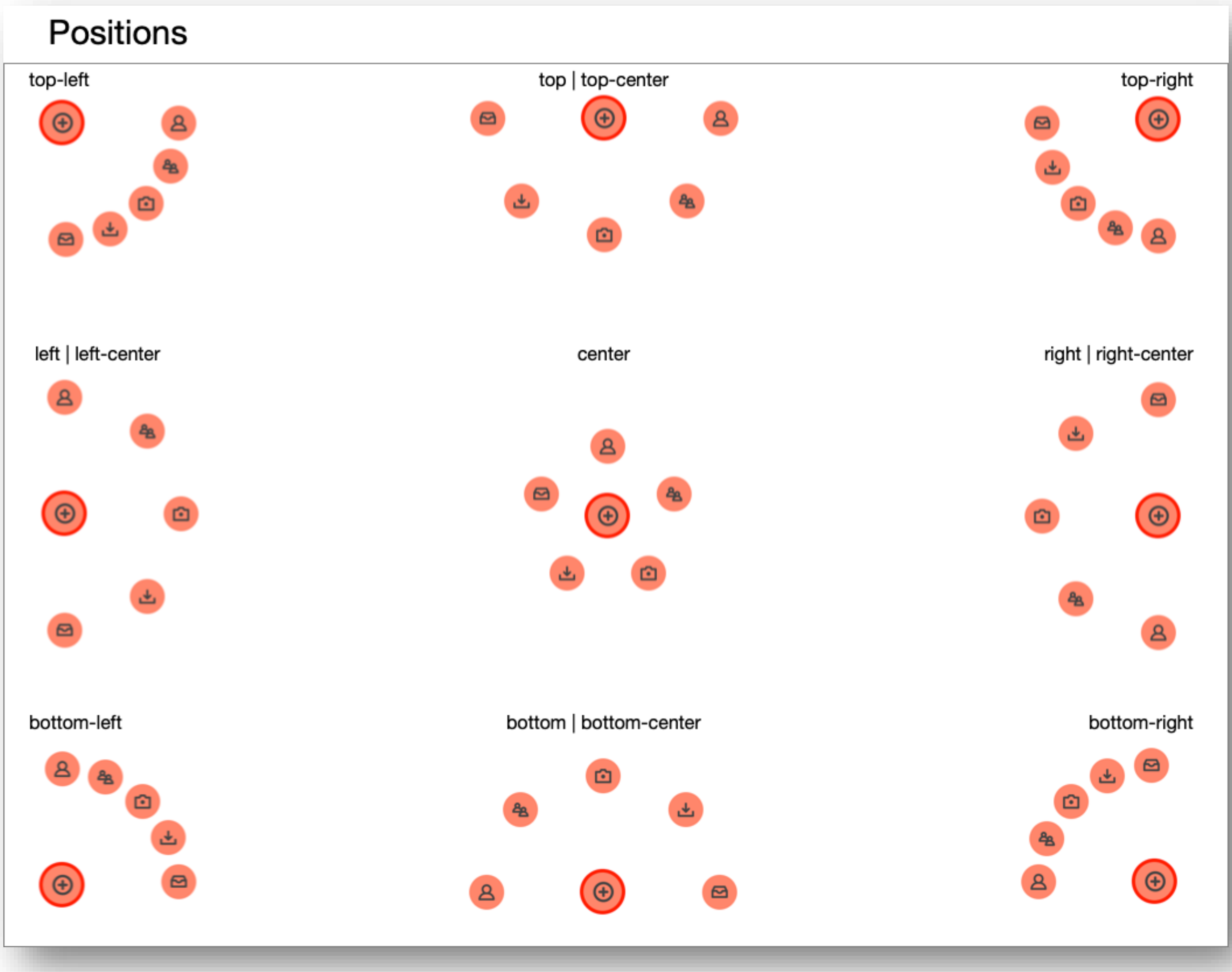
It is possible to apply an animation when the secondary buttons are shown or hidden.

6.1 Properties list

Name	Type	Description	Default value
mainBtn	boolean	If true, the button is designated as the main button of an FAB system. Its only function is to show and alternately hide the secondary buttons. Any callbacks defined (*OnClick* and *OnDoubleClick*) to this button will be ignored.	False
btnCol	collection	Collection of the names of the secondary buttons (name of the image form object) that will compose the FAB system. Their positions in the collection can be defined via an add function (FABAdd).	Empty collection
clockwise	boolean	If true, the arrangement of the buttons is clockwise, if not, it is the opposite.	True
startAngle	longint	Angle from which the first secondary button will be positioned. The values 0 to 359 are allowed. The value is in degrees. The starting point of the angle values (value 0) is located at 3 o'clock.	270
angleToTravel	longint	Sets the relative size of the angle where the secondary buttons will have to be spaced starting from the start angle (startAngle) and based on the direction (clockwise). Values from 0 to 360 are allowed. The value is in degrees.	180
Radius	longint	Space between the secondary buttons and the main button. The value is in pixel	100
mainPosition	text	Allows you to use a button positioning predefined by the component (see sub-chapter "predefined positions"). Using a predefined position will automatically update the startAngle and angleToTravel properties. Warning: the member function linked to mainPosition acts as a shortcut to enter predefined values for these two parameters only when it is called. It is not taken into account each time the FAB is used.	Empty string
animation	boolean	Allows you to activate the FAB animation	False

6.2 Predefined positions

Here are the 9 predefined positions and their identifiers that you can enter in the member function "FABMainPosition".



6.3 Member functions

List of member functions linked to the FAB functionnality.

- **FABIsMainButton** ({ $active }) : Getter/Setter for the property « mainBtn ».
- **FABIsClockwise**({ $clockwise }) : Getter/Setter for the property « clockwise ».
- **FABStartAngle** ({ $angle }) : Getter/Setter for the property « startAngle ».
- **FABAngleToTravel** ({ $angle }) : Getter/Setter for the property « angleToTravel ».
- **FABRadius** ({ $radius }) : Getter/Setter for the property « radius ».
- **FABMainPosition** ({ $namePosition }) : Getter/Setter for the property « mainPosition ».
- **FABAdd** ($nameBtn { ; $position }) : Adds the name of a button to the collection. You can define its position in the list as an optional second parameter, otherwise it will be added at the end. If the name of a button already exists in the list, the method will act as a update method for the position but will not create a duplicate.
- **FABRemove** ($nameBtn) : Removes from the collection the name of the button in parameter.
- **FABClearCollection** ( ) : Removes all items in the collection.
- **FABIsAnimated** ( ) : Activate/inactivate animation.

6.4 Setting up

The first thing to do to implement the FAB function is to create each of the buttons that will compose it.

Once done, you will define an instance as the main button using the "FABIsMainButton" member function. This button will now have the sole function of showing and hiding the secondary buttons on the "On Mouse Up" and/or "On Double Click" events. Any callbacks defined (OnClick and OnDoubleClick) for this button will be ignored.

Then, you have to add to this instance the names of the other buttons with the "FABAdd" function, you can define their position in the collection. You can also activate the FAB animation.

When loading your form, the main button will automatically hide the secondary buttons. For a practical example of implementation, we advise you to look at the HDI_FAB form code in the component lab.

7 Conclusion

The purpose of this document was to introduce you to the theoretical principles of the component as well as the different methods, formulas and properties available to you to manage the buttons.

As for the practical elements presented, they are intended to allow you to get off to a good start and to understand how the component works.

If you need help to implement the AJUI Button component in your application. You want to modify or extend its functionalities for a specific use. You want to have the source code of the AJUI Button component in order to perpetuate its use in your application with future versions of 4D. Feel free to contact us to discuss it.