



ViSi-Genie Advanced Buttons

DOCUMENT DATE: **25th April 2020**
DOCUMENT REVISION: **1.2**



Description

This Application Note explores the possibilities provided by ViSi-Genie for the **Button** object:

- Simple button
- Button with an icon
- On/Off button
- Group of buttons
- Menu based on buttons
- Menu based on buttons with messages

This application note requires:

- Workshop4 has been installed according to the document Workshop4 Installation;
- The user is familiar with the Workshop4 environment and with the fundamentals of ViSi-Genie, as described in Workshop4 User Guide and ViSi-Genie User Guide;
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics discussed in these recommended application notes.

Six ViSi-Genie projects are provided as examples to help you along this application note.

Content

Description	2
Content	2
Application Overview	3
Setup Procedure	4
Create a New Project	5
<i>Create a New Project</i>	<i>5</i>
Add a Button Object.....	5
Simple Button	6
Button with Icon	9
On/Off Button	11
Group of Buttons	14
Button-Based Menu	17
Translate a Menu into Forms and Buttons	18
<i>Go from a Button to a Sub-Menu</i>	<i>20</i>
<i>Summarise Forms and Buttons</i>	<i>21</i>
<i>Build the Menu</i>	<i>21</i>
Top Level Form	21
First Level Form	23
Second Level Form	24
Landing Page Forms	24
Button-Based Menu with Messages.....	26

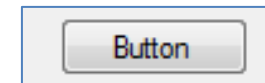
<i>Define the Events for Messages</i>	26
<i>Read the Messages</i>	27
<i>Read the Menu Selected</i>	29
<i>Understand the Messages</i>	30
<i>Prepare a Table with Messages</i>	30
Summary	31
Proprietary Information	32
Disclaimer of Warranties & Limitation of Liability.....	32

Application Overview

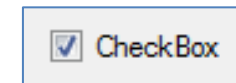
The button is the key element of a graphical user interface.

In ViSi-Genie, the button corresponds to different standard elements:

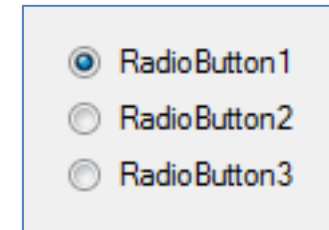
- Used alone, it triggers an action when pressed. This is the simple button.



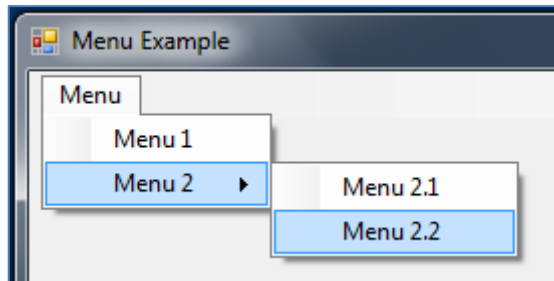
- The button can keep trace of its state, off and on, and provides a feed-back. The equivalent is a check-box.



- Grouped with other buttons, selecting one unselects the others. The equivalent is a group of radio-buttons.



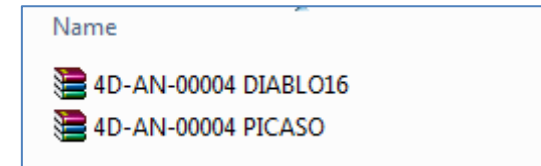
- Finally, combining forms and buttons allow building menus with sub-menus. The equivalent is a menu.



The different sections of this application notes have been redacted on a progressive order. It is thus recommended to read this document sequentially.

Setup Procedure

This application note comes with a zip file which contains two ViSi-Genie projects.



For instructions on how to launch Workshop4, how to open a ViSi-Genie project, and how to change the target display, kindly refer to the section “**Setup Procedure**” of the application note:

ViSi Genie Getting Started – First Project for Picaso Displays

ViSi Genie Getting Started – First Project for Diablo16

Create a New Project

Create a New Project

For instructions on how to create a new ViSi-Genie project, please refer to the section “**Create a New Project**” of the application note

ViSi Genie Getting Started – First Project for Picaso Displays

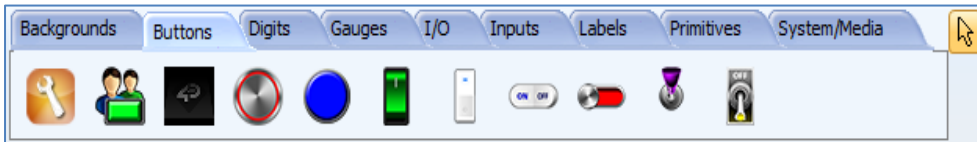
ViSi Genie Getting Started – First Project for Diablo16

Add a Button Object

Select the **Home** menu to display the objects:



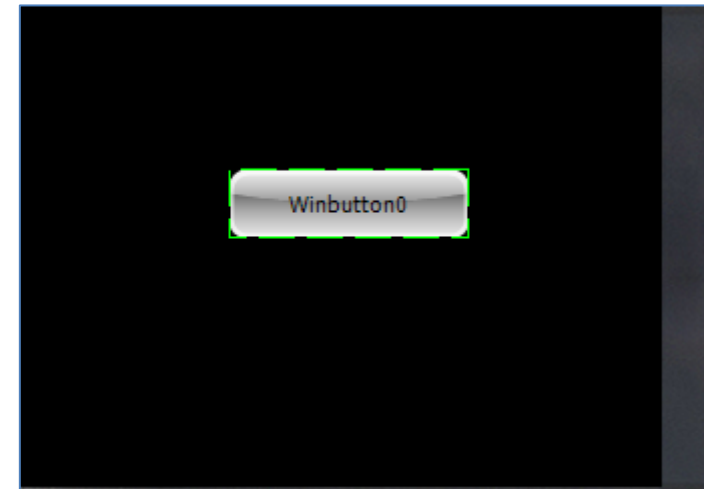
The **Button** object is located on the Buttons pane:



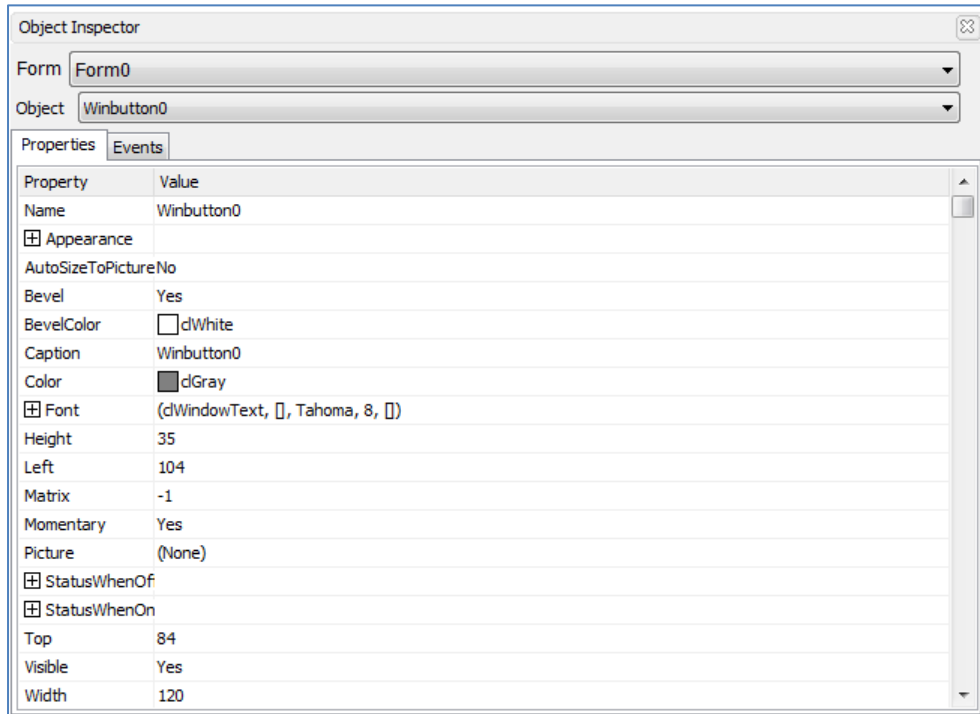
Click first on the Button icon...



...and then click on the WYSIWYG screen to place it:



Note the Object Inspector of the right part of the screen displays all the properties of the **WinButton0** object:



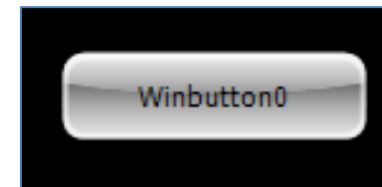
Simple Button

You can load the example...

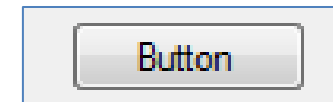
Example: 4D-AN-00004 PICASO - Simple Button or 4D-AN-00004 DIABLO16 - Simple Button

...or follow the procedure described hereafter.

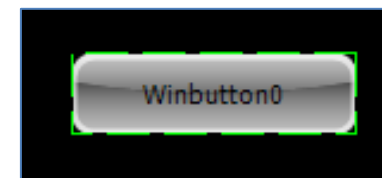
Here is a simple button:



This is the equivalent of the standard button:

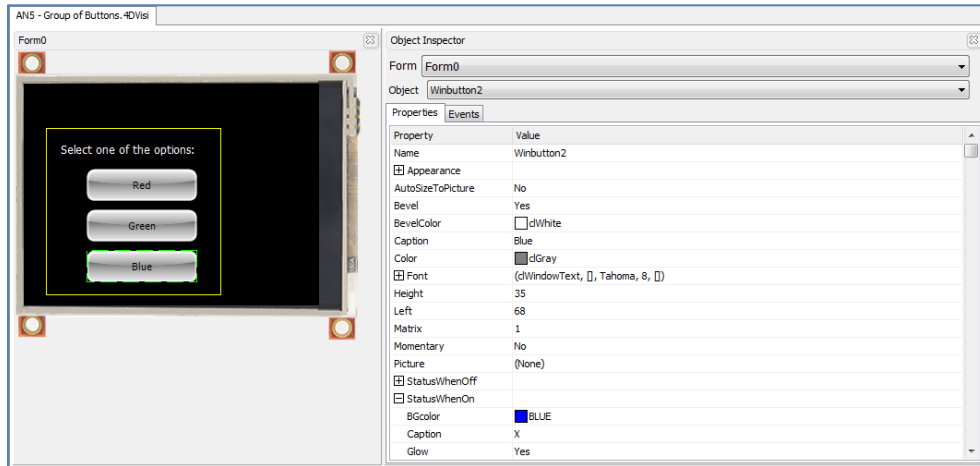


To edit the properties of the button, click on it on the WYSIWYG screen:

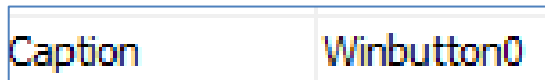


A dotted green line appears.

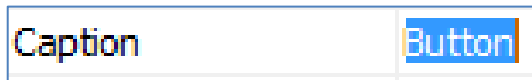
On the right part of 4D Workshop, the Object Inspector displays all the properties of the Winbutton0 button:



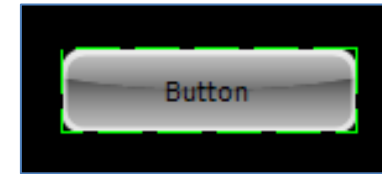
The caption of the button is defined by the **Caption** property. By default, the caption has the same value as the name of the object, here *Winbutton0*:



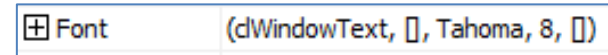
To change the caption, select the line of the property, type in the new caption, here **Button**, and press **Enter**:



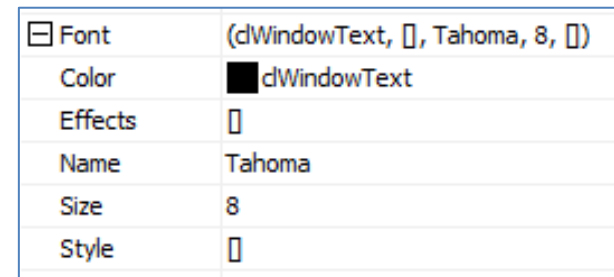
The WYSIWYG screen is updated accordingly:



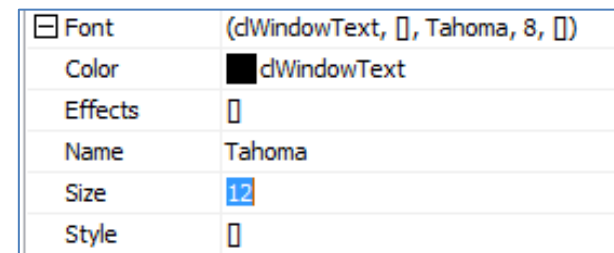
The font, the size and the colour of the caption are under the Font node:



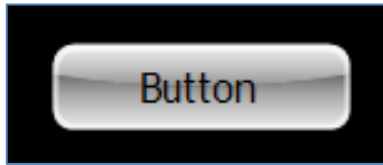
Click on the  to show the details:



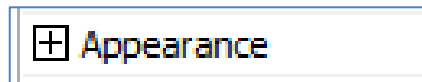
For example, change the font size from 8 to 12:



The button is updated accordingly:



By default, the buttons have a glossy appearance. The appearance properties are under the Appearance node:



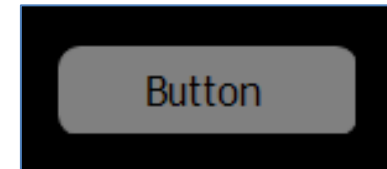
Click on the  to show the details:

[-] Appearance	
Alignment	Center
Layout	Left
PictureAlignment	Right
SimpleLayout	No
Spacing	3

For example, change SimpleLayout from No to Yes to obtain a flat button:

[-] Appearance	
Alignment	Center
Layout	Left
PictureAlignment	Right
SimpleLayout	Yes
Spacing	3

The button is updated accordingly:



Feel free to play with the many options of colours, alignment, bevel to customise the button to your specific needs.

Button with Icon


You can load the example...

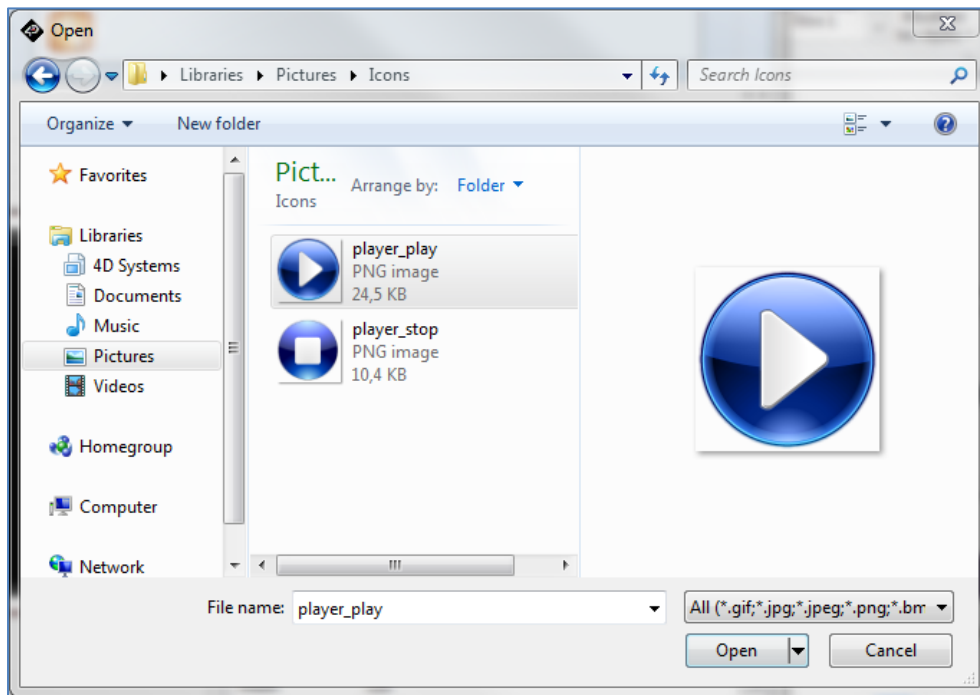
Example: 4D-AN-00004 PICASO - Simple Button – Button with Icon or 4D-AN-00004 DIABLO16 - Simple Button – Button with Icon

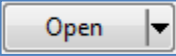
...or follow the procedure described hereafter.

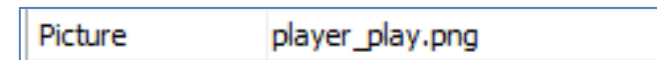
A button can display an icon.



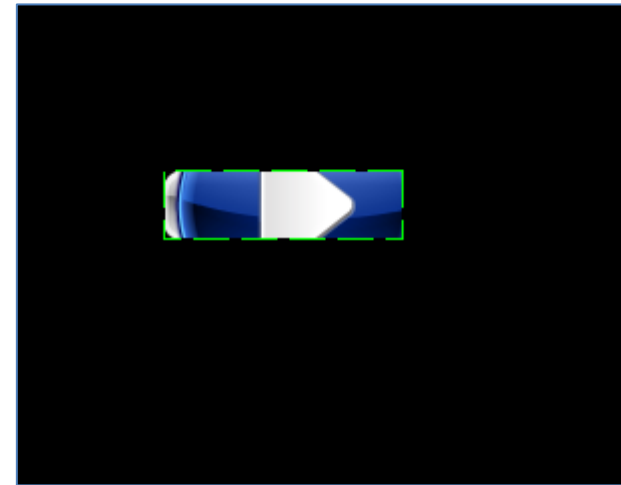
Click on  and the standard Open window asks for a image:



Select the image you want, here *player_play*, and click on . The object property is updated with the name of the image...



...and the WYSIWYG screen shows the new button:



To resize the button, either enter the values for Height and Width...



...or click on the green dotted rectangle and resize the button:



Set **SimpleLayout** to Yes to hide the bevel:

Property	Value
Name	Winbutton0
<input type="checkbox"/> Appearance	
Alignment	Center
Layout	Left
PictureAlignmerRight	
SimpleLayout	Yes
Spacing	3

Final result is:



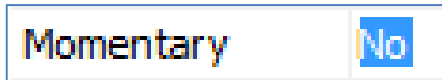
On/Off Button

You can load the example...

Example: 4D-AN-00004 PICASO – On-Off Button or 4D-AN-00004 DIABLO16 – On-Off Button

...or follow the procedure described hereafter.

To convert a momentary button into an on/off button, set the **Momentary** property to **No**:



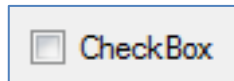
Pressing the button turns it on, pressing again turns it off.

However, the lack of visual feed-back doesn't make this on/off button very easy to use.

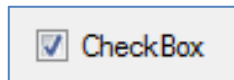
Let's add some visual feed-back:

- Green **OFF** when the button is off
- Red **ON** when the button is on.

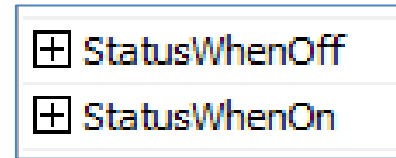
This is the equivalent of the standard check-box, off...





...and on.



The green OFF and red ON captions are set under the **StatusWhenOff** and **StatusWhenOn** properties, respectively:



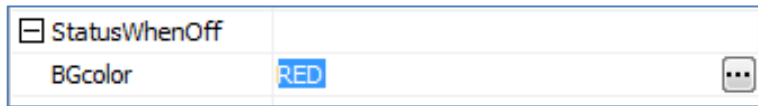
Click on the  to show the details:


<input type="checkbox"/> StatusWhenOff	
BGcolor	 RED
Caption	
Glow	Yes
<input type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes
<input type="checkbox"/> StatusWhenOn	
BGcolor	 RED
Caption	
Glow	Yes
<input type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Let's start with the green OFF under StatusWhenOff:

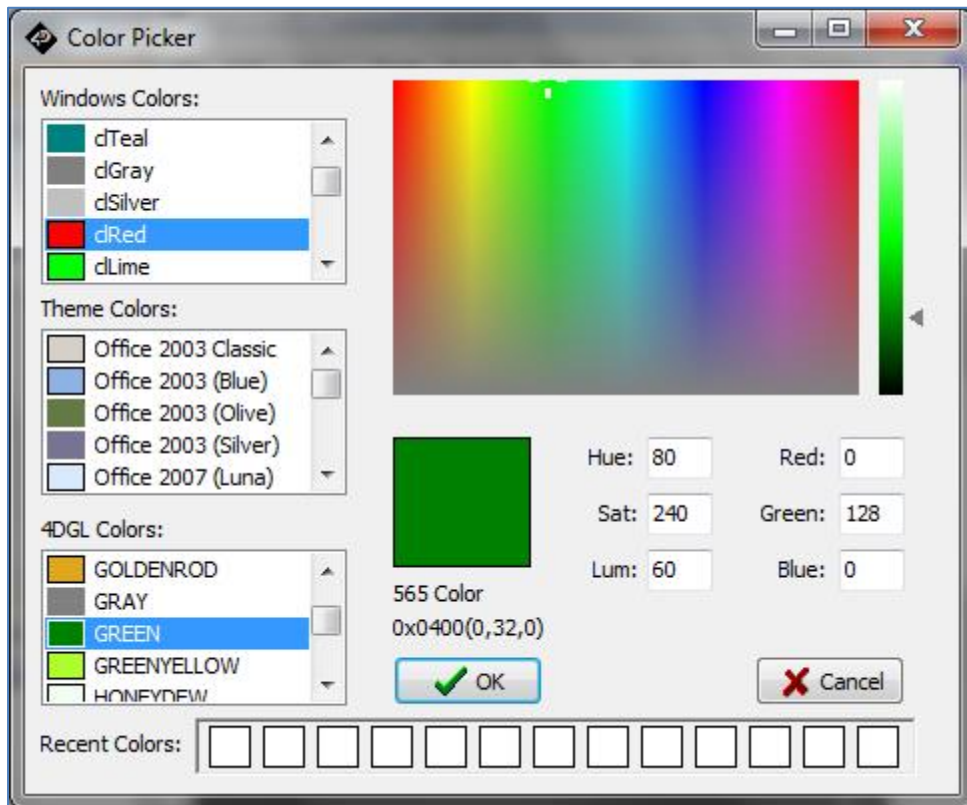
<input type="checkbox"/> StatusWhenOff	
BGcolor	 RED
Caption	
Glow	Yes
<input type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Click on the BGColor line property:



A  symbol appears. Click on it.

A new window **Colour Picker** appears and shows all the colours available:

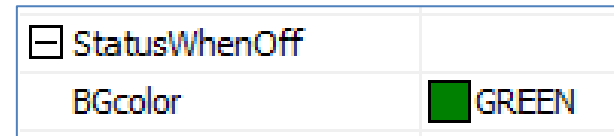


Colours are grouped under Windows, Theme and 4DGL.

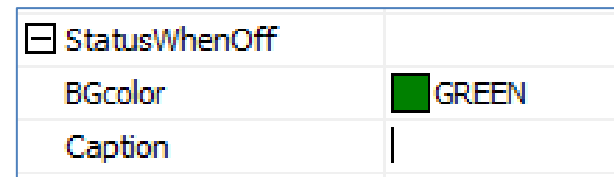
Manual values can also be set for Hue-Saturation-Luminance or Red-Green-Blue components.

Select **Green** under 4DGL Colours and then click **OK**.

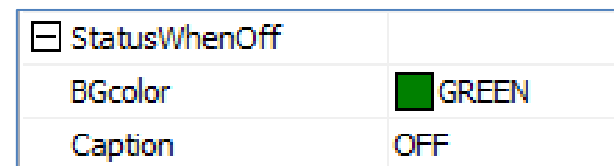
The colour of the status is updated accordingly:



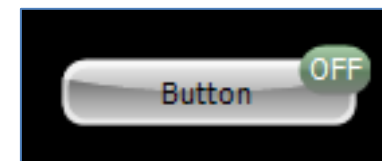
Select the new line, **Caption**



and type in the caption, **OFF**, then press Enter:




The button on the WYSIWYG screen is updated accordingly:




Keep Visible with the value **Yes**, otherwise the caption wouldn't be visible.

Final result looks like:

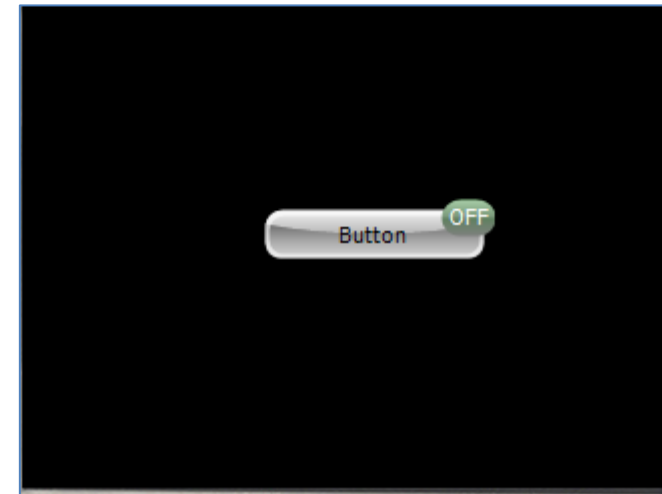
[-] StatusWhenOff	
BGcolor	 GREEN
Caption	OFF
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Proceed the same way with the ON caption under StatusWhenOn:

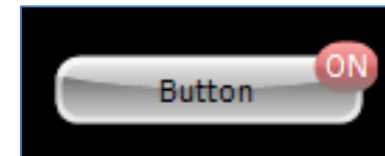
[-] StatusWhenOn	
BGcolor	 RED
Caption	ON
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Build and run the project.

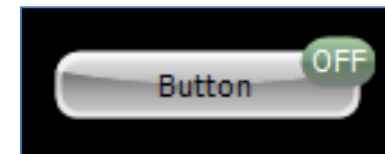
The screen shows the button turned OFF:



Touching the button turns it ON:



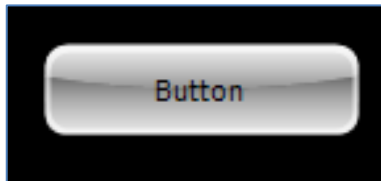
Touching again turns it OFF again:



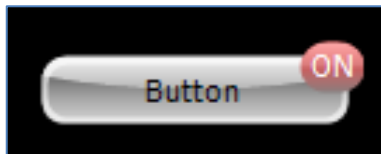
Now, let's get rid of the OFF caption to keep only the ON caption.
Go to the **Visible** property under StatusWhenOff and changed it to **No**:

<input type="checkbox"/> StatusWhenOff	
BGcolor	■ LIME
Caption	OFF
Glow	Yes
<input type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	No

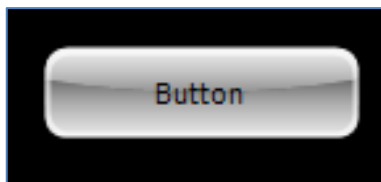
The form displays the button OFF with no caption:



Touching the button turns it ON:



Touching again turns it OFF again:



Group of Buttons

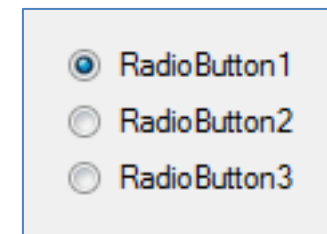
You can load the example...

Example: 4D-AN-00004 PICASO – Group of Buttons or 4D-AN-00004 DIABLO16 – Group of Buttons

...or follow the procedure described hereafter.

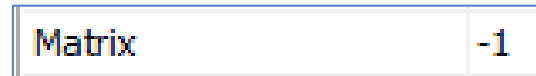
Simple button and on/off button are not the only uses of the buttons. Very often, different buttons are used together to bring a choice among different options. Selecting one option cancels the previous one.

This is the equivalent of the radio-buttons:

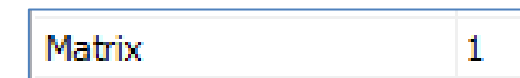


With ViSi-Genie, such a group is called a **matrix**.

To link the buttons together, just change the value of the matrix from **-1** for no matrix...



...to the number of the matrix, here **1**:



It is important that all buttons grouped share the same number of matrix, otherwise pressing on one button will not release the other buttons of the group.

Let's start with the first button, Red.

- Set the caption to **Red**:

Caption	Red
---------	-----


- Enter the number of the Matrix, **1**:

Matrix	1
--------	---

- And set **Momentary** to **No**:



Momentary	No
-----------	----

- Finally, define the **StatusWhenOff** and **StatusWhenOn** parameters as seen in the previous section:

<input type="checkbox"/> StatusWhenOff	
BGcolor	 RED
Caption	
Glow	Yes
<input checked="" type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], No)
Visible	No
<input type="checkbox"/> StatusWhenOn	
BGcolor	 RED
Caption	X
Glow	Yes
<input checked="" type="checkbox"/> Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Do the same for the Green and Blue buttons with their respective values, update the **Caption**, enter the number of the **Matrix** and set **Momentary** to No.

- For the Green button:

[-] StatusWhenOff	
BGcolor	 RED
Caption	
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], No)
Visible	No
[-] StatusWhenOn	
BGcolor	 LIME
Caption	X
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

- For the Blue button:

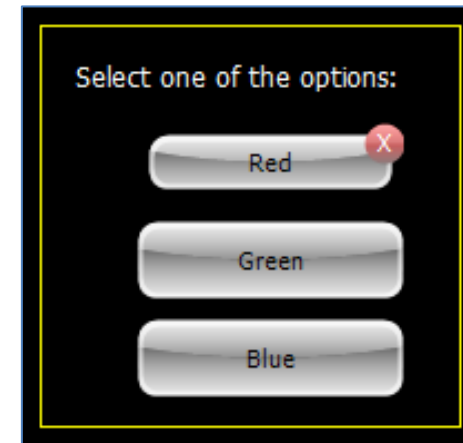
[-] StatusWhenOff	
BGcolor	 RED
Caption	
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], No)
Visible	No
[-] StatusWhenOn	
BGcolor	 BLUE
Caption	X
Glow	Yes
[+] Font	(WHITE, [], Tahoma, 8, [], Yes)

Build and run the project.

The form displays the buttons and no one is selected:



Press on the Red button:



Red is selected.

Press on the Green button:



Red is released and Green is selected.

Press on the Blue button:



Green is released and Blue is selected.

Button-Based Menu

You can load the example...

Example: 4D-AN-00004 PICASO – Menus or 4D-AN-00004 DIABLO16– Menus

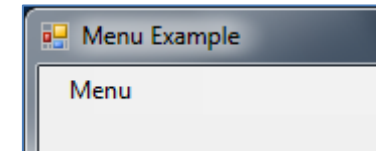
...or follow the procedure described hereafter.

Another use of the buttons is building a menu. A standard menu with sub-menu doesn't fit into a 3.2" or 4.3" screen.

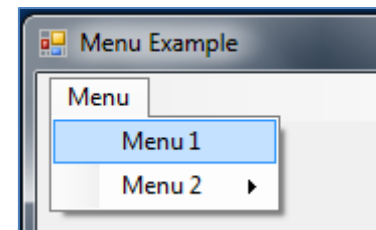
Instead, each menu is going to be shown as a screen, or form, with the sub-menu as buttons, plus a backward button to return to the former level.

Below a menu with two levels:

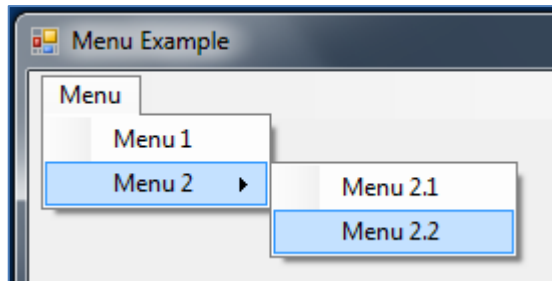
- Top level:



- First level, with the Menu 1 option highlighted:



- Second level, for the sub-menu of Menu 2, and with the Menu 2.2 option highlighted:

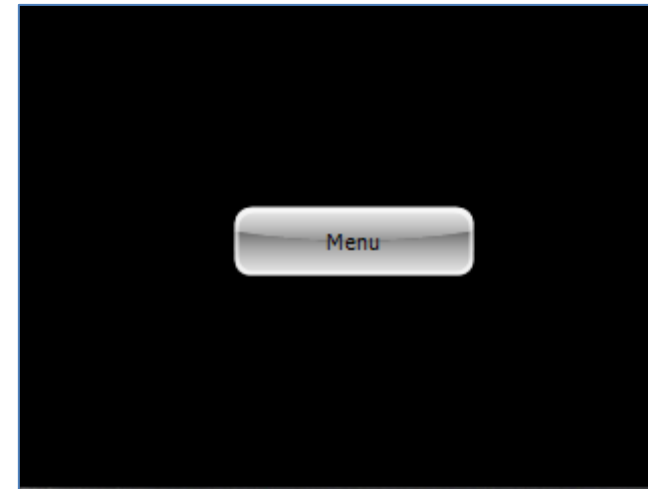


In this example, Menu 1, Menu 2.1 and Menu 2.2 are terminal options. They have no sub-menus associated.

Translate a Menu into Forms and Buttons

With Visi-Genie, this whole menu including all its sub-menus is going to be translated into 3 forms, with each form containing the options of the corresponding sub-menu:

- Top level:



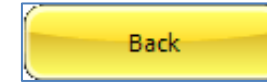
- First level, with two options:



- Second level with the 2 options for the Menu 2 sub-menu:

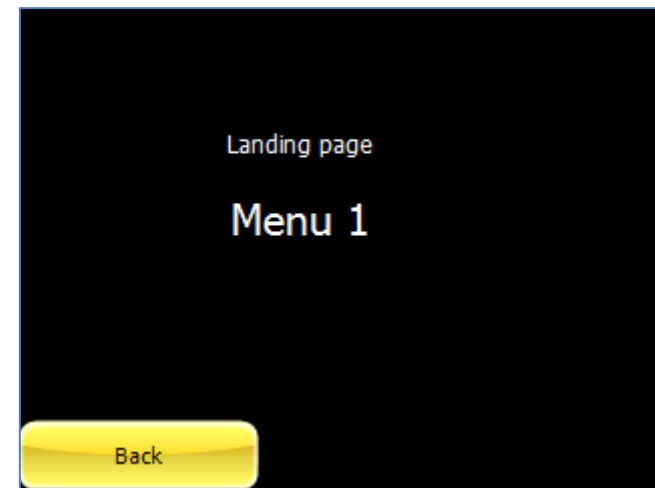


For better ergonomics, a yellow Back button is included:



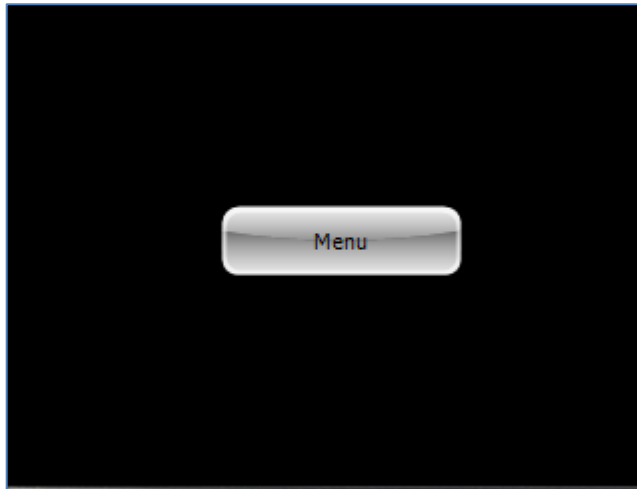
The Back button returns to the previous node. Above, it goes to the first level.

Each terminal option, Menu 1, Menu 2.1 or Menu 2.2, ends to a landing page. Here the example for Menu 1:

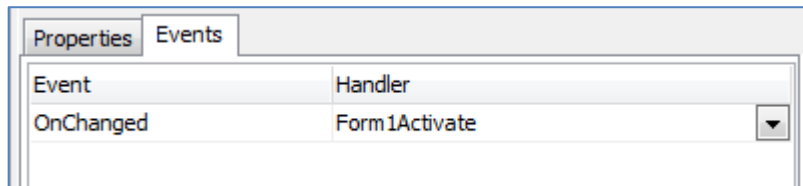


Go from a Button to a Sub-Menu

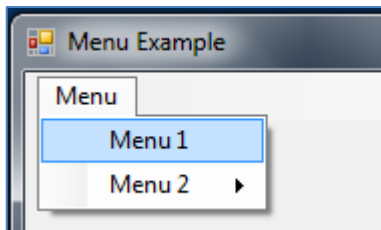
Each form includes the options of the menu as buttons.



The Button object generates the event **onChanged** when pressed:



When pressed, the button associated with an option opens the form which contains the sub-menus of the option selected, just like the classic way:



Here, pressing on the **Menu** button opens the forms with the **Menu 1** and **Menu 2** options:



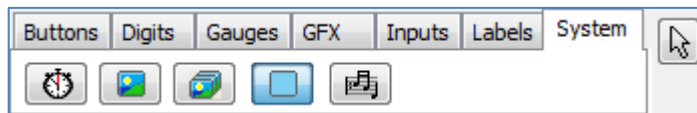
Summarise Forms and Buttons

Here is the summary of the forms and the buttons, with their associated actions:

Form	Button	Action = Command
Form 0 = top level	Menu	Open first level = activate Form 1
Form 1 = first level	Menu 1	Go to Menu 1 landing page = activate Form 3
	Menu 2	Open second level of Menu 2 = activate Form 2
	Back	Return to top level = activate Form 0
Form 2 = second level of Menu 2	Menu 2.1	Go to Menu 2.1 landing page = activate Form 4
	Menu 2.2	Go to landing Menu 2.2 landing page = activate Form 5
	Back	Return to first level = activate Form 1
Form 3 = Menu 1 landing page	Back	Return to last menu = activate Form 1
Form 4 = Menu 2.1 landing page	Back	Return to last menu = activate Form 2
Form 5 = Menu 2.2 landing page	Back	Return to last menu = activate Form 2

Build the Menu

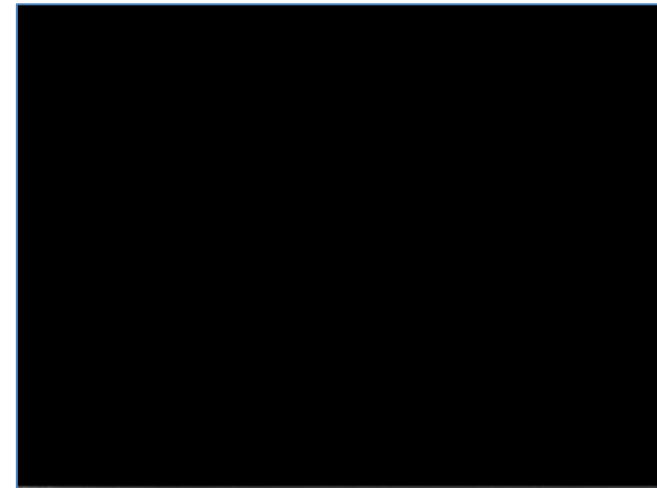
First, create as many forms as levels and landing pages, 6 in the example.
To create a form, select the System pane of objects...



...click on the Form object...

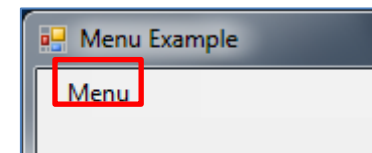


...an empty form appears on the WYSIWYG screen:



Top Level Form

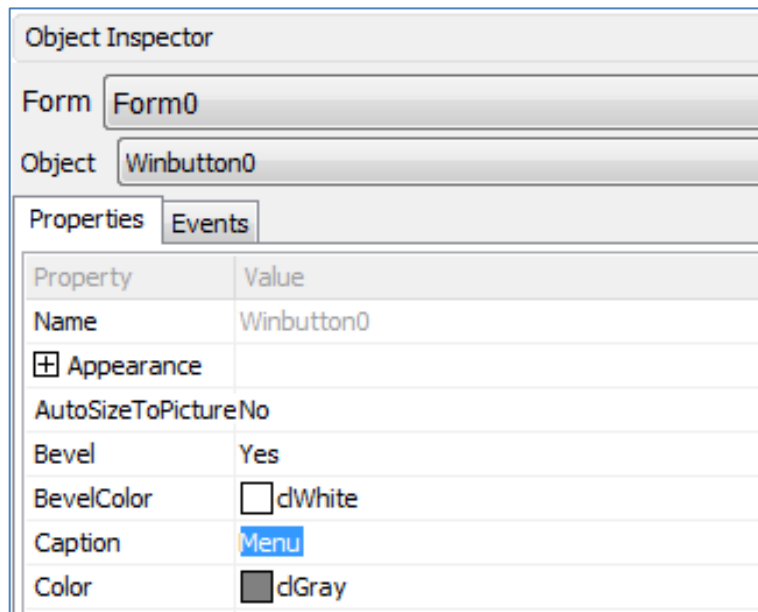
Form 0 corresponds to the top level of the menu:



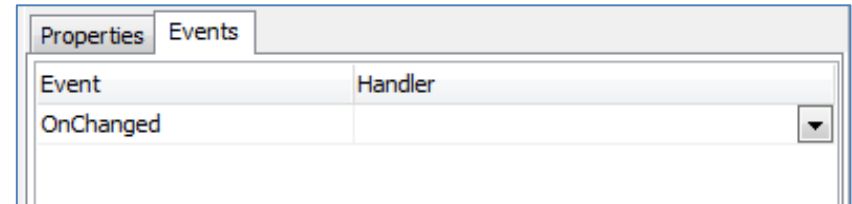
Add a button to the form, **WinButton0**:




Select **WinButton0** and rename it to **Menu**:

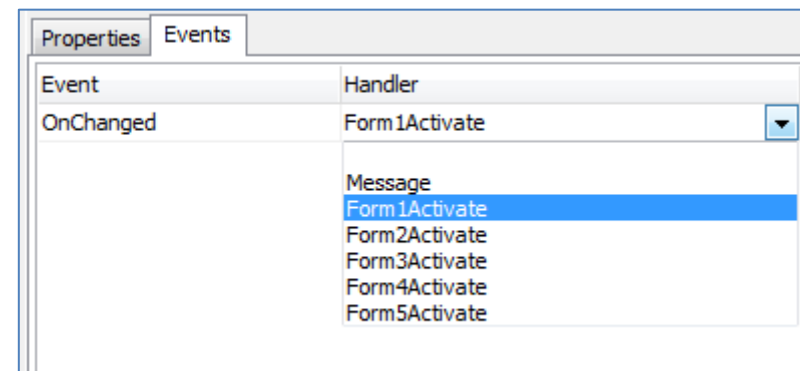


The Button object generates the event **onChanged** when pressed. Select the Events pane on the Object Inspector and click on the line **onChanged**:



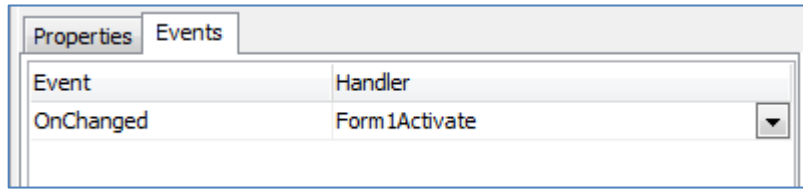
A  symbol appears. Click on it.

A drop-down list appears with a list of commands:



Five of the six forms we created before appear. **Form0** is not listed as **WinButton0** belongs to that form.

Here, the **Form1Activate** command stands for the activation of the **Form1**, which corresponds to the first level of the menu, with Menu 1 and Menu 2 options.



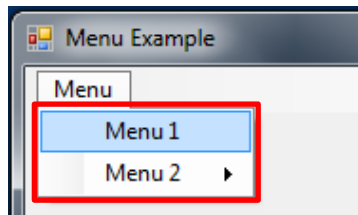
When **WinButton0** Menu is pressed, the first level of the menu is displayed by **Form1**:



Form1 is empty for the moment.

First Level Form

Form1 correspond to first level of the menu, with two options, Menu 1 and Menu 2:



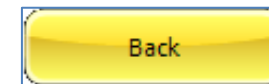
Add two buttons, call them Menu 1 and Menu 2 and define the following events:

Button	Event	Command
Menu 1	onChanged	Form3Activate
Menu 2	onChanged	Form2Activate

The WYSIWYG screen displays:



Optionally, add the the yellow Back button...



...and define its event:

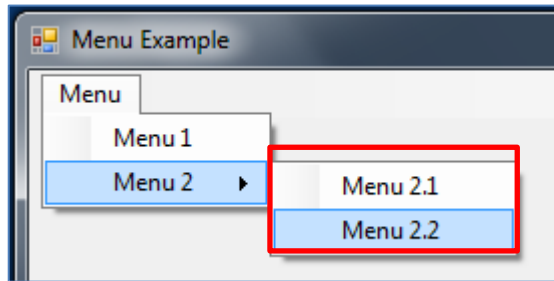
Button	Event	Command
Back	onChanged	Form0Activate

The WYSIWYG screen displays:



Second Level Form

Pressing Menu 2 calls the second level menu, with Menu 2.1 and Menu 2.2 options:



Proceed with **Form2** with the following buttons:

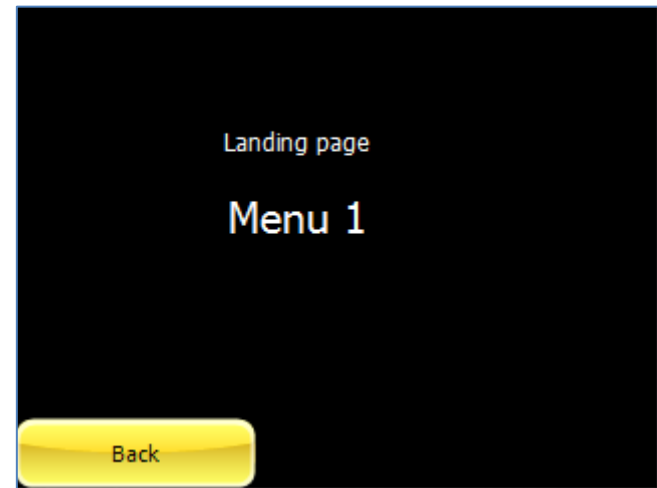
Button	Event	Command
Menu 2.1	onChanged	Form4Activate
Menu 2.2	onChanged	Form5Activate
Back	onChanged	Form1Activate

Final result is:



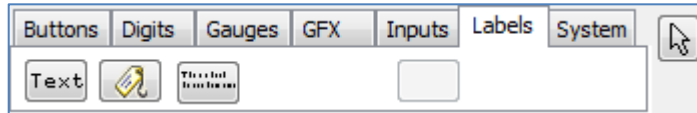
Landing Page Forms

The following forms **Form3**, **Form4** and **Form5**, are landing pages:



Include two StaticText objects, one with *Landing page* caption and the other with the menu number, here *Menu 1*.

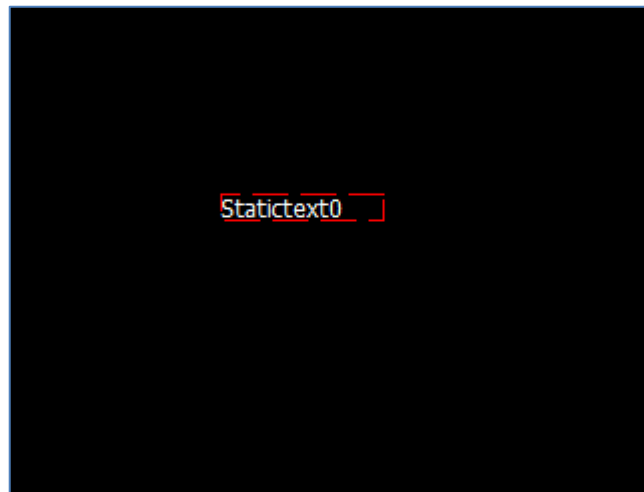
To add a **StaticText** object, select the Labels panes...



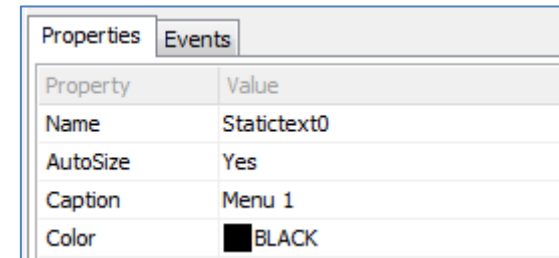
...click on the **StaticText** object...



...then place it on the WYSIWYG screen...



...finally, enter its caption:



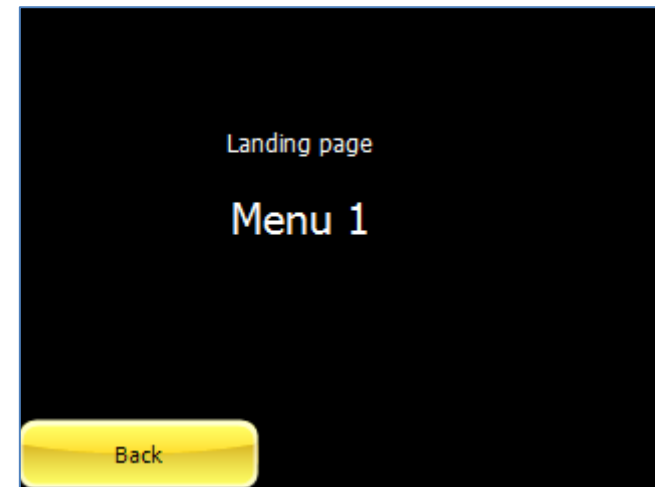
Each landing page is going to have its specific Menu label, Menu 1, Menu 2.1 or Menu 2.2.

Finally, add one yellow Back button with the following definition:

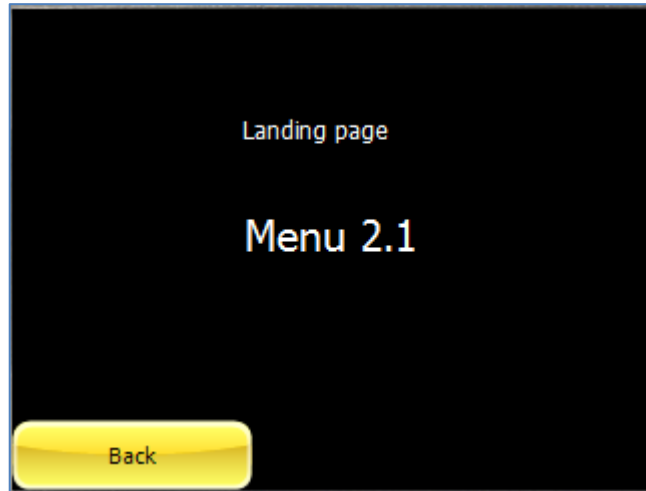
Form	Button	Event	Command
Form3	Back	onChanged	Form1Activate
Form4	Back	onChanged	Form2Activate
Form5	Back	onChanged	Form2Activate

The landing pages look like:

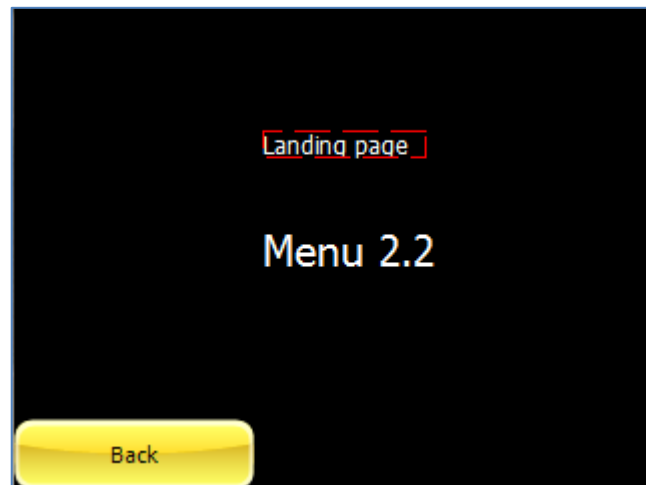
- **Form3** as landing page for Menu 1



- **Form4** as landing page for Menu 2.1



- **Form5** as landing page for Menu 2.2



Save, build and upload the project.

Congratulations, you have completed a menu with sub-menus!

Button-Based Menu with Messages

You can load the example...

Example: 4D-AN-00004 PICASO – Menus with messages or 4D-AN-00004 DIABLO16 – Menus with messages

...or follow the procedure described hereafter.

Instead of displaying landing pages, we want now the screen to send back to the micro-controller connected to the screen the menu selected.

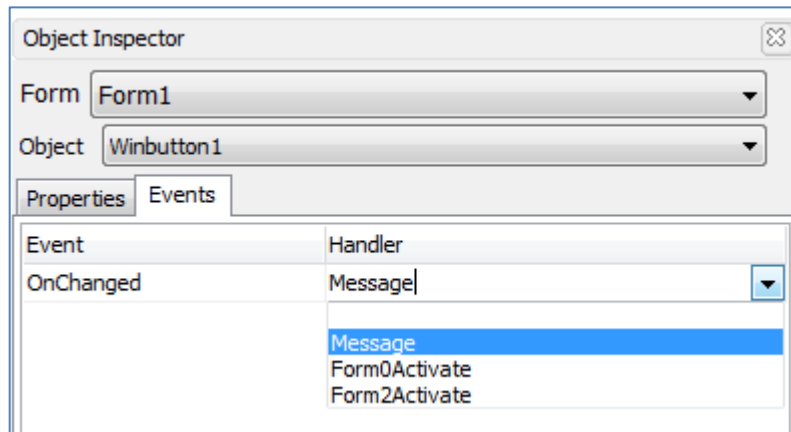
Instead of associating the command activate a form to the **onChanged** events, the buttons will send a message.

Define the Events for Messages

To do so, delete all the landing page forms, **Form3**, **Form4** and **Form5**, as they are no longer required.

Then define **Message** as handler for the **onChanged** event only for the buttons associated with terminal options, more specifically **WinButton1** for Menu 1, **WinButton3** for Menu 2.1 and **Winbutton4** for Menu 2.2.

On **Form1** for the first level menu, **WinButton1** or Menu 1 has the event **onChanged** associated with the action **Message**:



Note that the deleted forms no longer appear. **Form1** is the active form and thus isn't listed.

Do the same for **Form2** corresponding to the second level menu:

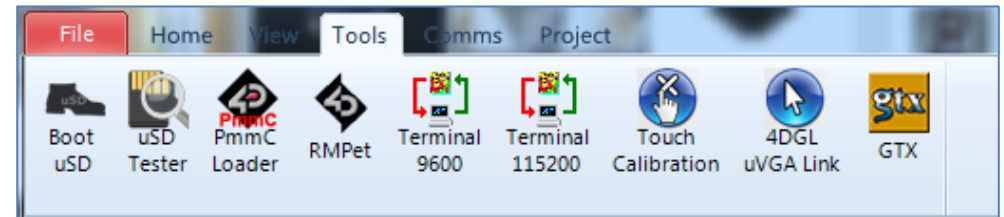
Form	Button	Event	Command
Form1	Menu 1	onChanged	Message
Form2	Menu 2.1	onChanged	Message
	Menu 2.2	onChanged	Message

Read the Messages

Save, build and upload the project.

In order to read the messages, we are going to use the built-in debugger.

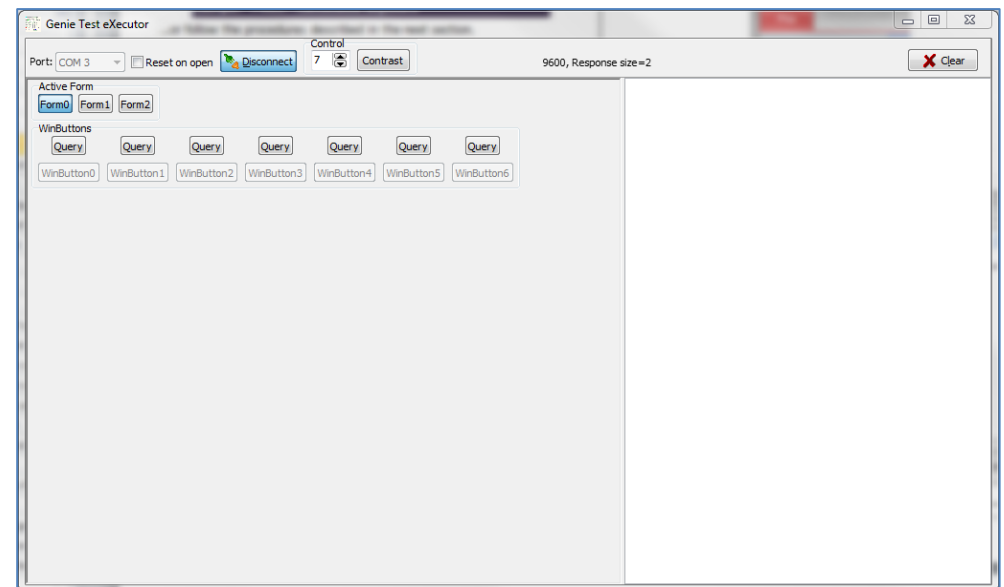
To debug the project, select on the **Tools** menu...



...and launch the **Codeless Executive Test Instrument**:



A new screen appears, with the form and objects we have defined previously:

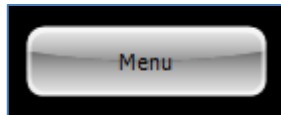


Let's start!

The screen shows the main menu:



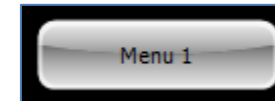
- ...press Menu...



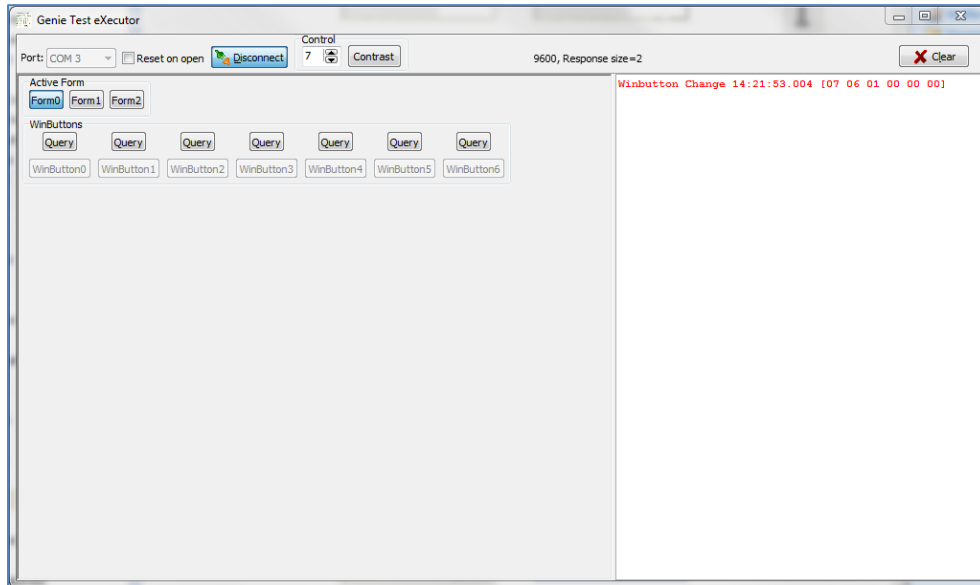
- ...the screen is refreshed with the top level menu...



- ...then press Menu 1



- ...the debugger shows now:



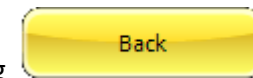
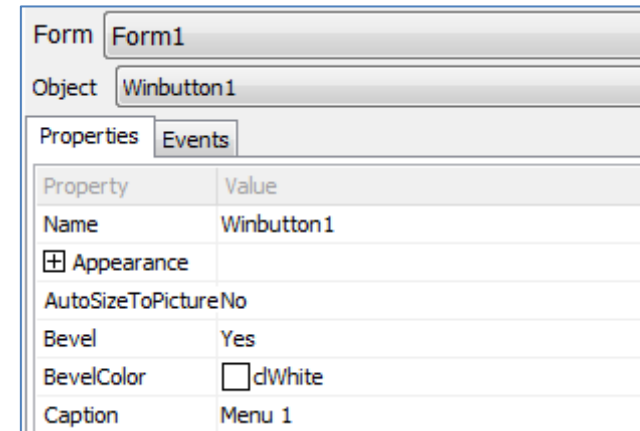
Read the Menu Selected

The white area on the right of the **Codeless Executive Test Instrument** window, displays the messages sent by the screen:

Winbutton Change 14:21:53.004 [07 06 01 00 00 00]

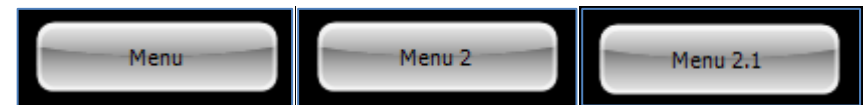
07 06 01 00 00 00 is the message sent by **WinButton1**. **06** stands for WinButton object type and **01** for number 1.

The **WinButton1** button corresponds to the first Button object, Menu 1:



Go back to the main menu by pressing

Now, from the main menu, press the sequence



The debugger displays the message:

Winbutton Change 14:23:16.965 [07 06 03 00 00 02]

07 06 03 00 00 02 is the message sent by **WinButton3**. **06** stands for WinButton object type and **03** for number 3.

All codes are in hexadecimal.

The **WinButton3** button corresponds to Menu 2.1:

Form	Form2
Object	Winbutton3
Properties	Events
Property	Value
Name	Winbutton3
[-] Appearance	
AutoSizeToPictureNo	
Bevel	Yes
BevelColor	<input type="checkbox"/> dWhite
Caption	Menu 2.1

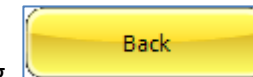
Understand the Messages

As **WinButton4** button corresponds to Menu 2.2...

Form	Form2
Object	Winbutton4
Properties	Events
Property	Value
Name	Winbutton4
[-] Appearance	
AutoSizeToPictureNo	
Bevel	Yes
BevelColor	<input type="checkbox"/> dWhite
Caption	Menu 2.2

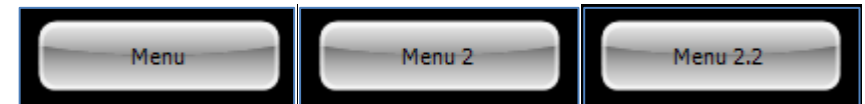
...we might expect a message like **07 06 04 00 00 05**, with **06** for WinButton object type and **04** for number 4.

Let's check it!



Go back to the main menu by pressing  twice.

Now, from the main menu, press the sequence



The debugger displays the message...

Winbutton Change 14:24:43.514 [07 06 04 00 00 05]

...just as expected.

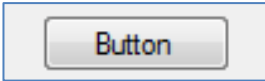
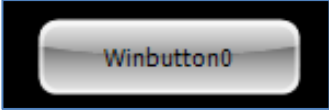
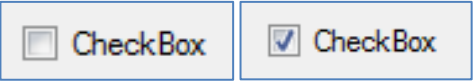

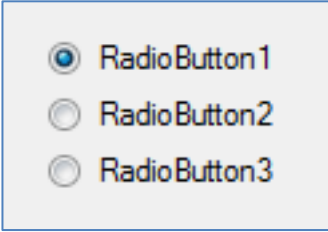
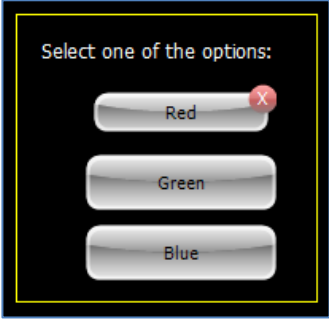
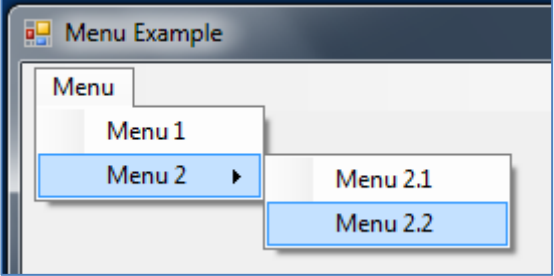

Prepare a Table with Messages

A good idea is to prepare a table with the different messages:

Form	Button	Caption	Event	Message
Form1	WinButton1	Menu 1	onChanged	07 06 01 00 00 00
Form2	WinButton3	Menu 2.1	onChanged	07 06 03 00 00 02
	WinButton4	Menu 2.2	onChanged	07 06 04 00 00 05

This table is very useful for interpreting the messages sent by the screen. Note that the other buttons doesn't need to send messages, as they manage the user interface and display another form. Only the terminal options need to send messages.

Summary

Element	Standard	ViSi-Genie	Comment
<p>Button</p>			<p>Simple WinButton</p>
<p>Check-box</p>			<p>Momentary set to No StatusWhenOff and StatusWhenOn defined</p>
<p>Radio-Button</p>			<p>Matrix with number Momentary set to No StatusWhenOff and StatusWhenOn defined</p>
<p>Menu</p>			<p>One Form per menu One WinButton per option, buttons grouped per sub-menu Message for terminal options Back button recommended</p>

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expresses or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.