APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Winbuttons

DOCUMENT DATE: **3rd May 2019**
DOCUMENT REVISION: **1.1**

W W W . 4 D S Y S T E M S . C O M . A U

Description

This application note shows how to add and configure a fancy button or win button, one of the widgets available in Workshop. Before getting started, the following are required:

- Any of the following 4D Picaso display modules:

gen4-uLCD-24PT	gen4-uLCD-28PT	gen4-uLCD-32PT
uLCD-24PTU	uLCD-28PTU	uVGA-III

  and other superseded modules which support the ViSi Genie environment
- The target module can also be a Diablo16 display

gen4-uLCD-24D series	gen4-uLCD-28D series	gen4-uLCD-32D series
gen4-uLCD-38D series	gen4-uLCD-43D series	gen4-uLCD-50D series
gen4-uLCD-70D series		
uLCD-35DT	uLCD-43D Series	uLCD-70DT

  Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor. The display module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- 4D Programming Cable / µUSB-PA5/uUSBPA5-II for non-gen4 displays (uLCD-xxx)
- 4D Programming Cable & gen4-IB / 4D-UPA / gen4-PA for gen4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- 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 presented in these recommended application notes.

Content

Application Overview

This application note explains how to add a fancy button or win button to the WYSIWYG screen, how to paste the generated code, how to display the different states, and how to write a code for button touch detection. The procedures for configuring a fancy button to function as a simple (or momentary) button, an on/off (or toggle) button, or as one button among a group of many are also shown. The user can then choose which among these modes of function to implement depending on the nature of the intended application.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Create a New Project

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

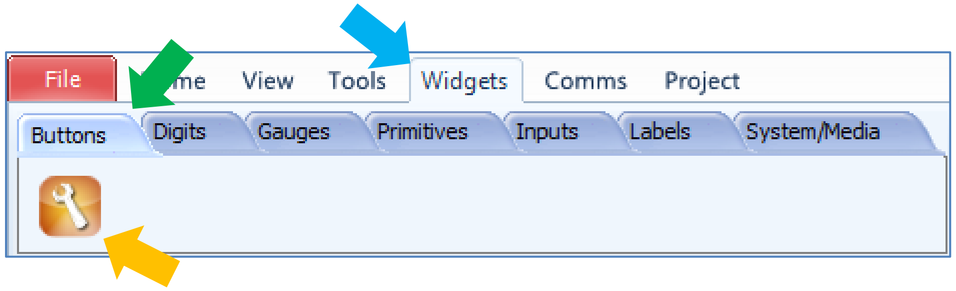
ViSi Getting Started - First Project for Picaso and Diablo16

Design the Project

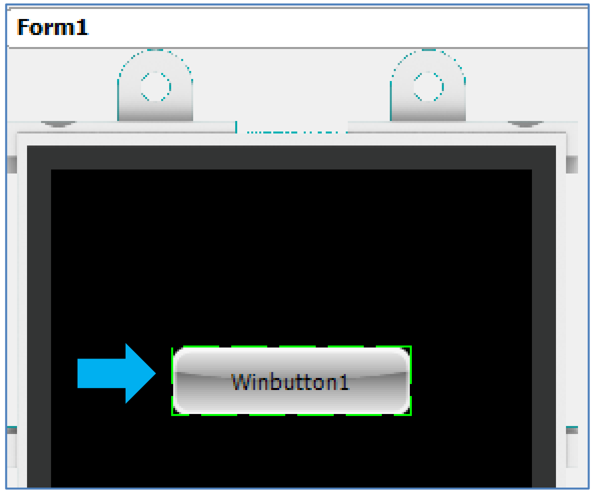
The Simple Button

Create a Fancy Button in the WYSIWYG Screen

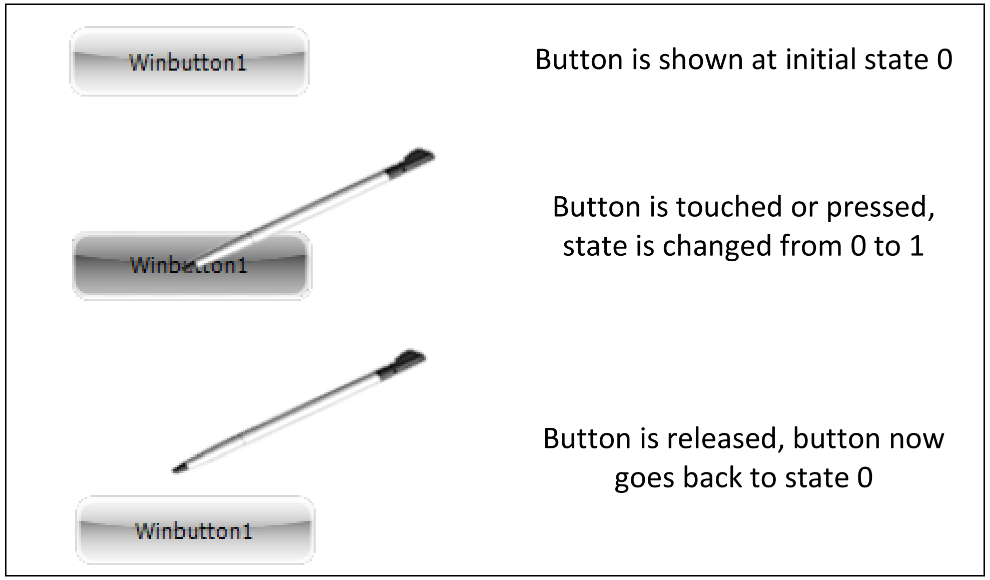
Go to the Widgets menu, select the Buttons pane, and click on the fancy button icon.



Once the fancy button icon is selected, click on the WYSIWYG screen to place it.



The Object Inspector shows the different properties of the button. By default, a button added to the WYSIWYG screen is a momentary button. In common GUIs the simple button is the equivalent of a momentary button. It is activated when the mouse button is depressed and deactivated as soon as the mouse button is released. In a similar manner, a momentary button is activated when a touch press is detected, and deactivated when the touch is released. To illustrate:

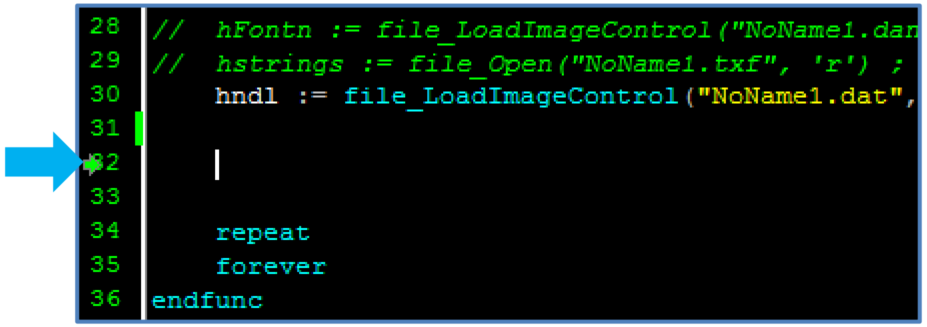

Winbutton1
Button is shown at initial state 0
Winbutton1
Button is touched or pressed, state is changed from 0 to 1
Winbutton1
Button is released, button now goes back to state 0

Note that the button has two states, state 0 and state 1. The button appears darker at state 1, giving the impression of being depressed. ViSi automatically generates these two states for a simple momentary button. All we have to do is create a code for the button to display any of these two states depending on whether it is touched or not.

The physical equivalent of a momentary button is the pushbutton or the tact switch.

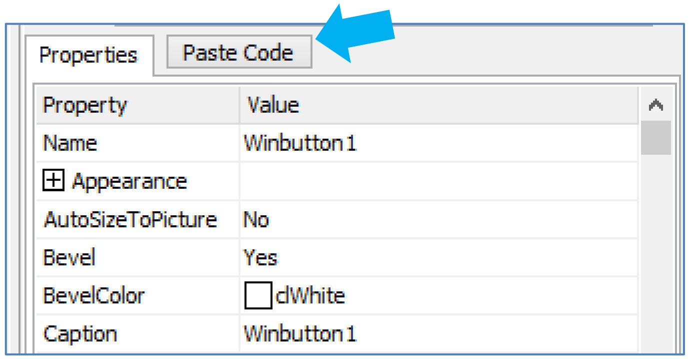
Insert the Fancy Button Code

Go to the code area and place the cursor just after the handle assignment statement (line 32 in this example).



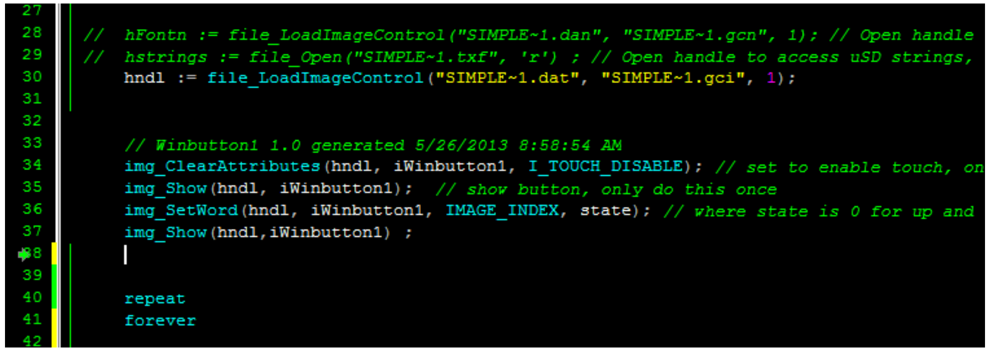
```
28  //   hFontn := file_LoadImageControl("NoName1.dan
29  //   hstrings := file_Open("NoName1.txf", 'r') ;
30       hndl := file_LoadImageControl("NoName1.dat",
31
32       |
33
34       repeat
35       forever
36  endfunc
```

Having selected the fancy button object, go to the Object Inspector and click on the Paste Code button.



Property	Value
Name	Winbutton1
⊞ Appearance	
AutoSizeToPicture	No
Bevel	Yes
BevelColor	☐ clWhite
Caption	Winbutton1

The code will be updated accordingly.



```
27
28  //  hFontn := file_LoadImageControl("SIMPLE~1.dan", "SIMPLE~1.gcn", 1); // Open handle
29  //  hstrings := file_Open("SIMPLE~1.txf", 'r') ; // Open handle to access uSD strings,
30      hndl := file_LoadImageControl("SIMPLE~1.dat", "SIMPLE~1.gci", 1);
31
32
33      // Winbutton1 1.0 generated 5/26/2013 8:58:54 AM
34      img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE); // set to enable touch, on
35      img_Show(hndl, iWinbutton1);  // show button, only do this once
36      img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); // where state is 0 for up and
37      img_Show(hndl,iWinbutton1) ;
38      |
39
40      repeat
41      forever
42
```

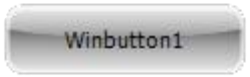
A new block for the button is generated, along with comments for each line. Additional explanations are given below.

The statement in line 34

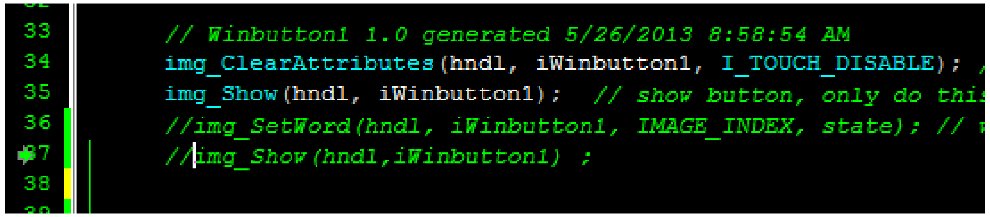
```
img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE);
```

enables the object, iWinbutton1, for touch detection. Not doing this will make the object unresponsive to touch.

The command **img_Show**(hndl,iWinbutton1) displays the button at the default initial state – state 0.

Appearance	**State**
	0

Comment out the statements in lines 36 and 37 for now.



```
33        // Winbutton1 1.0 generated 5/26/2013 8:58:54 AM
34        img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE); /
35        img_Show(hndl, iWinbutton1);  // show button, only do thi
36        //img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); //
37        //img_Show(hndl,iWinbutton1) ;
38
```

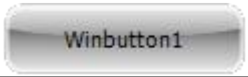
Save the Program

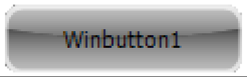
Jump to the section "Run the Program" to learn how to compile and download a program. Now we make a quick test if the simple program works. Save the program with the desired file name first, and then compile and download it. The program in this example is saved as "**FancyButtonTutorial**".

Upon inserting the µSD card, the fancy button at state 0 should be shown on the display module screen as configured in the WYSIWYG screen. Below is the code at this point, with the unnecessary lines excluded.

```
#platform "uLCD-32WPTU"

// Program Skeleton 1.0 generated 5/26/2013 2:17:40 PM

#inherit "4DGL_16bitColours.fnc"

#inherit "VisualConst.inc"

#inherit "FancyButtonTutorialConst.inc"

func main()
    putstr("Mounting...\n");
    if (!(disk:=file_Mount()))
        while(!(disk :=file_Mount()))
            putstr("Drive not mounted...");
            pause(200);
            gfx_Cls();
            pause(200);
        wend
    endif
    gfx_TransparentColour(0x0020);
    gfx_Transparency(ON);
    gfx_Cls();

    hndl := file_LoadImageControl("FANCYB~1.dat",
    "FANCYB~1.gci", 1);

    // Winbutton1 1.0 generated 5/26/2013 2:20:48 PM
    img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE);
    img_Show(hndl, iWinbutton1);
    repeat
    forever
endfunc
```

Change the Button State from 0 to 1

Appearance	State
	0

Appearance	State
	1

To change the state of the fancy button, we use the command:

```
img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 1) ;
```

Here the value of IMAGE_INDEX for iWinbutton1 is set to 1. The parameter IMAGE_INDEX is the current frame of the object. There are two frames for iWinbutton1 – frame 0 and frame 1, each representing the object at its different states. The following code will illustrate this.

```
//set iWinbutton1 to display frame 1
img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 1) ;

//display iWinbutton1
img_Show(hndl, iWinbutton1);

//add a delay
pause(2000);

//set iWinbutton1 to display frame 0
img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 0) ;

//display iWinbutton1
img_Show(hndl, iWinbutton1);

repeat
forever
```

Insert the code above to your main program. When compiled and downloaded, it should display the button at state 1 for two seconds, then at state 0 forever.

Control the Simple Button with Touch

In this section we will configure the button to respond to touch.

Enable Touch Detection

Before using the touch feature, enable it with the function:

```
touch_Set(TOUCH_ENABLE);
```

To disable the feature, use the function:

```
touch_Set(TOUCH_DISABLE);
```

The touch detection feature runs in the background and disabling it when not in use will free up extra resources for the 4DGL CPU cycles.

Check Touch Status

Now that the screen is enabled for touch detection, it needs to be constantly checked for a change in state. The status of a touch response is retrieved by using the following command:

```
touch_Get(TOUCH_STATUS);
```

Using the **touch_Get()** function returns a value depending on the current state. Integers 0 to 3 or their MACRO equivalents are returned based on the following results:

0	=	NOTOUCH
1	=	TOUCH_PRESSED
2	=	TOUCH_RELEASED
3	=	TOUCH_MOVING

Check if the Button is Touched or Not

Of course the button is only a part of the screen. When the screen is touched, we need to know if the point of touch is within the region of interest, which is the button. One way to do this is to use the function, **img_Touched**(handle, index)**.** This function returns back the index if the image (or button) is touched or returns -1 if not.

```
n := img_Touched(hndl, iWinbutton1)
if(n == iWinbutton1)
      print("iWinbutton1 is touched");
if(n == -1)
      print("Touch is outside iWinbutton1);
endif
```

A Simple Example

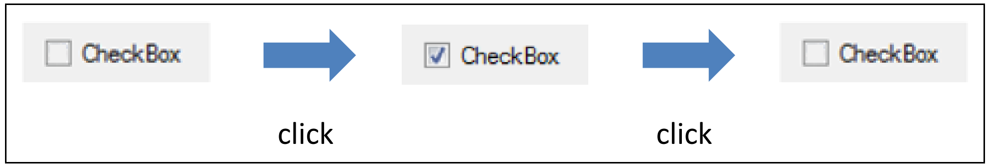
The previous code was improved to create a program that initially displays a button at state 0. When touched or pressed, the button will be displayed at state 1, giving the impression that it is depressed. Upon release, the button goes back to state 0.

Open, compile, and download the accompanying ViSi file **FancyButtonTutorial.4dViSi** to your display module. Note that aside from press and release, the program also has a code for detecting movement. The user is encouraged to analyse and experiment with the code.

The On/Off or Toggle Button

How an On/Off Button Works

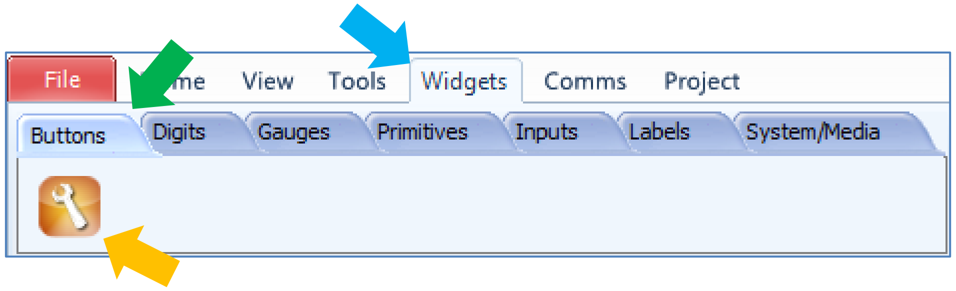
The fancy button can also be configured to function as an on/off button. A mouse click activates it, and another click is needed to deactivate it. In other GUIs, this is equivalent to a check box.



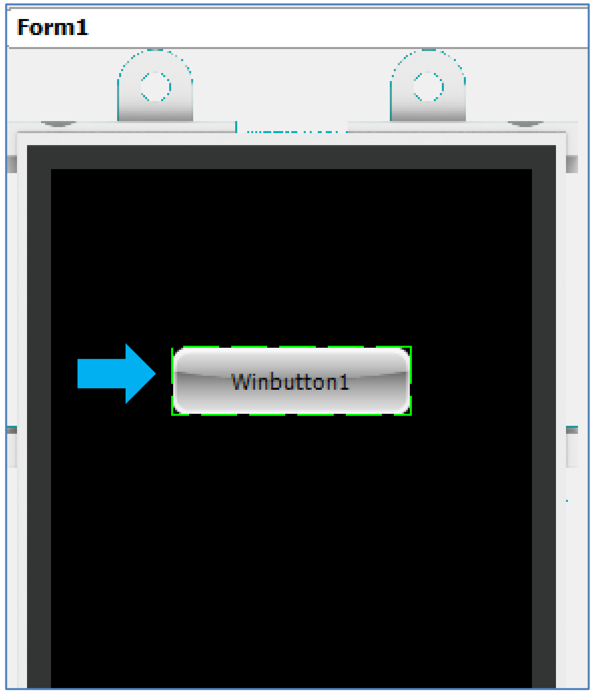
An on/off button responds to touch in a similar manner, as will be shown later. Both the toggle switch and the rocker switch can be thought of as a physical equivalent of an on/off button.

Create a Fancy Button in the WYSIWYG Screen

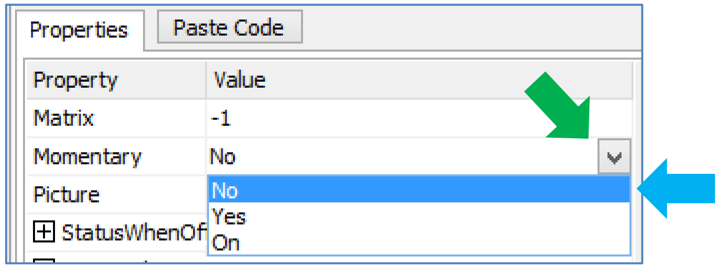
In this section, we will create a program for an on/off button. It is best to create a new program for this, separate from the simple button program developed previously. After having removed the block comment symbols in the program skeleton, go to the Widgets menu, select the Buttons pane, and click on the fancy button icon.



Once the fancy button icon is selected, click on the WYSIWYG screen to place it.



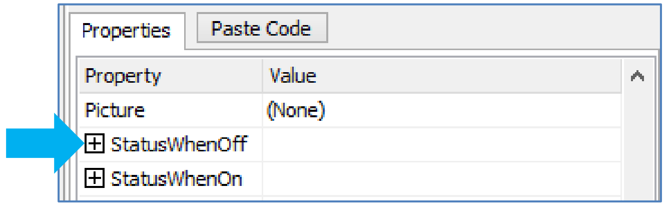
The Object Inspector shows the different properties of the button. Click on the property Momentary and click on the symbol ⌄. A drop down menu appears. Choose No.



Property	Value
Matrix	-1
Momentary	No
Picture	No / Yes / On
⊞ StatusWhenOf	

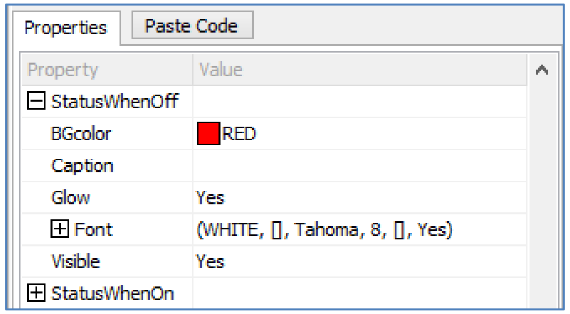
Configure the Off State Appearance

It is possible to add a button caption, the purpose of which is to visually indicate what state the button is to be displayed at. In the Object Inspector click on the symbol ⊞ beside StatusWhenOff.



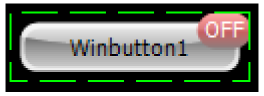
Property	Value
Picture	(None)
⊞ StatusWhenOff	
⊞ StatusWhenOn	

The properties under StatusWhenOff will appear.



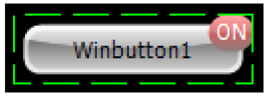
Property	Value
⊟ StatusWhenOff	
BGcolor	🟥 RED
Caption	
Glow	Yes
⊞ Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes
⊞ StatusWhenOn	

Click on the Caption property line, type in the text “OFF”, then press Enter. The WYSIWYG screen will be updated accordingly.



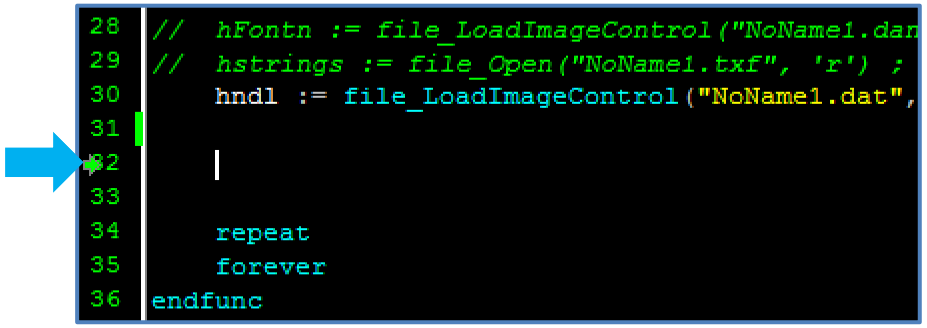
Configure the On State Appearance

Follow the same procedure to create the on state appearance of the button. Use the properties under StatusWhenOn. Type in “ON” for the button caption. Shown below is the on state appearance of the button when you run the program later on.



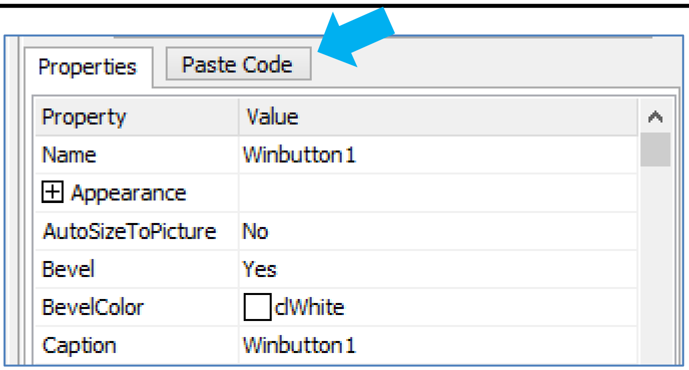
Insert the Code

Go to the code area and place the cursor just after the handle assignment statement (line 32 in this example).



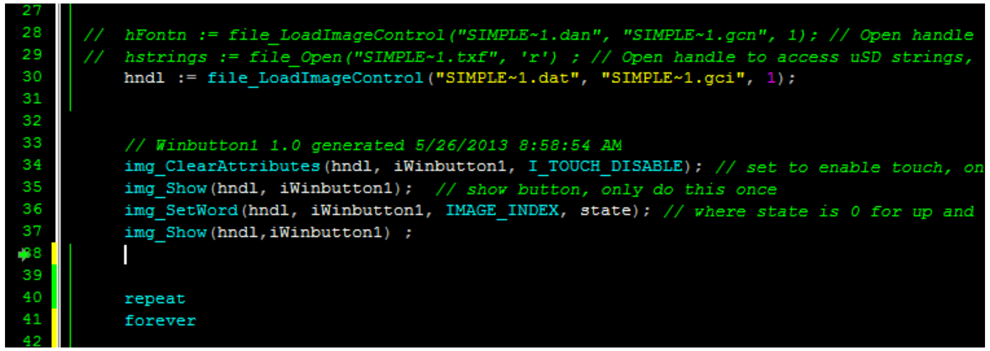
```
28 //   hFontn := file_LoadImageControl("NoName1.dan
29 //   hstrings := file_Open("NoName1.txf", 'r') ;
30      hndl := file_LoadImageControl("NoName1.dat",
31
32      |
33
34      repeat
35      forever
36 endfunc
```

Having selected the fancy button object, go to the Object Inspector and click on the Paste Code button.



Property	Value
Name	Winbutton1
⊞ Appearance	
AutoSizeToPicture	No
Bevel	Yes
BevelColor	☐ clWhite
Caption	Winbutton1

The code will be updated accordingly.

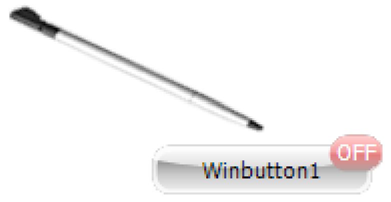


```
27
28 //   hFontn := file_LoadImageControl("SIMPLE~1.dan", "SIMPLE~1.gcn", 1); // Open handle
29 //   hstrings := file_Open("SIMPLE~1.txf", 'r') ; // Open handle to access uSD strings,
30      hndl := file_LoadImageControl("SIMPLE~1.dat", "SIMPLE~1.gci", 1);
31
32
33      // Winbutton1 1.0 generated 5/26/2013 8:58:54 AM
34      img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE); // set to enable touch, on
35      img_Show(hndl, iWinbutton1);  // show button, only do this once
36      img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); // where state is 0 for up and
37      img_Show(hndl,iWinbutton1) ;
38      |
39
40      repeat
41      forever
42
```

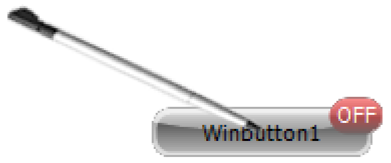
A new block for the button is generated, along with comments for each line. Comment out the statements in lines 36 and 37 if you intend to test run the program at this point. The module will simply display the button at the default initial state (state 0).

On/Off Button States

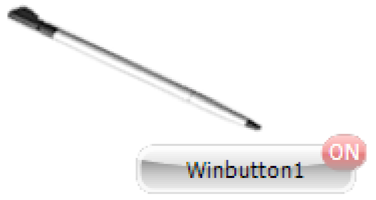
Unlike the simple momentary button which has only two states, the on/off button has four states, which are automatically generated by Workshop. The button cycles between these four states as it is toggled on and off. To illustrate:



Button is initially shown at state 0

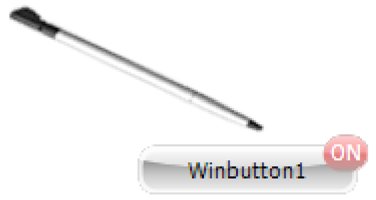


Button is pressed, state is changed from 0 to 1

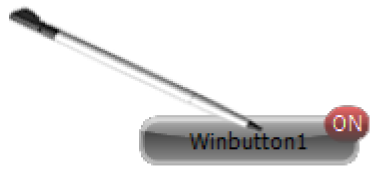


Button is released, state is changed from 1 to 2

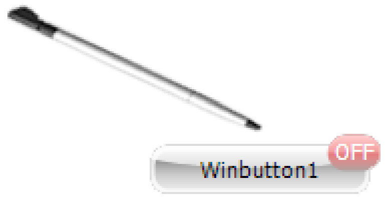
Notice that the button appears dark when pressed and light when released. To continue:



Button is shown at state 2



Button is pressed, state is changed from 2 to 3



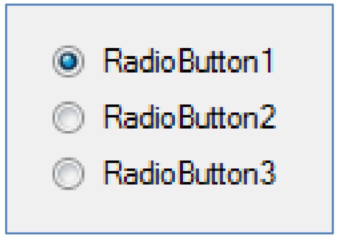
Button is released, state now goes back to state 0

A Simple Example with Touch Detection

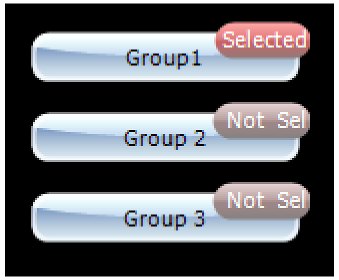
Attached is a simple ViSi program, **ToggleTutorial.4dViSi**, for the user to study and experiment with.

Button Matrix – a Group of 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 radio buttons:



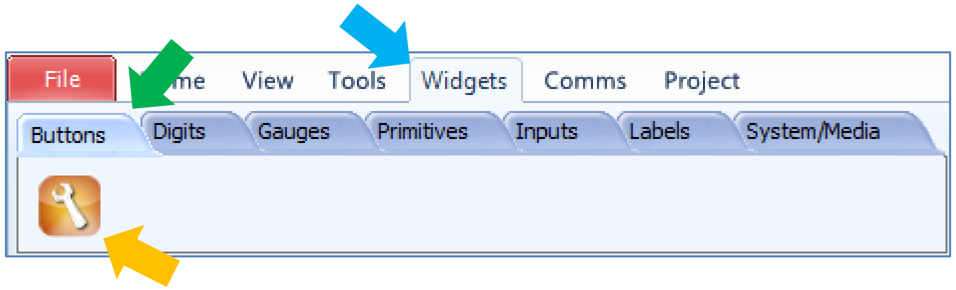
In ViSi, this can be shown as:



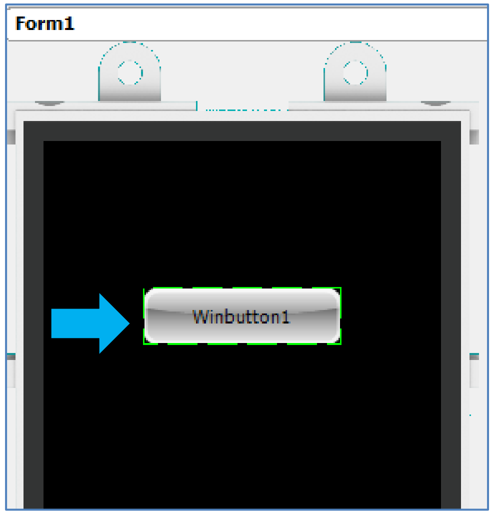
Actually, the three buttons are on/off buttons, each of which is programmed to show different states depending on the location of touch.

Create a Fancy Button in the WYSIWYG Screen

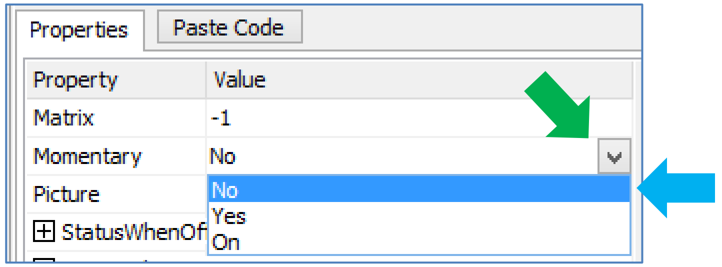
We will now create a program for a group of buttons. Again, it is best to create a new program for this, separate from the ones developed previously. After having removed the block comment symbols in the program skeleton, go to the Widgets menu, select the Buttons pane, and click on the fancy button icon.



Once the fancy button icon is selected, click on the WYSIWYG screen to place it. The screen will be updated accordingly.



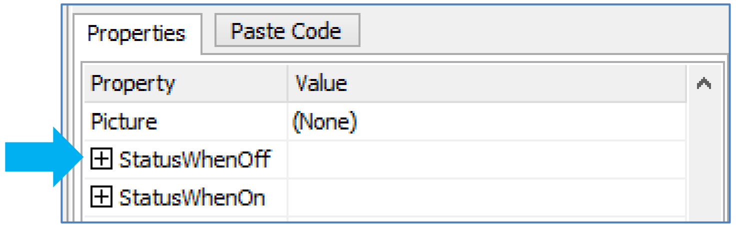
The Object Inspector shows the different properties of the button. Click on the property Momentary and click on the symbol ⌄. A drop down menu appears. Choose No.



Property	Value
Matrix	-1
Momentary	No
Picture	No / Yes / On
⊞ StatusWhenOf	

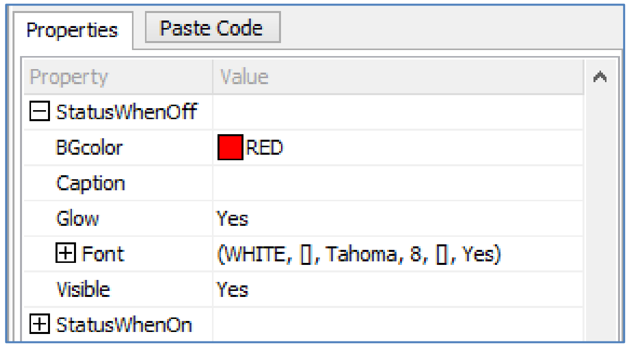
Configure the Off State Appearance

Now add a button caption, the purpose of which is to visually indicate if the button is selected or not. In the Object Inspector click on the symbol ⊞ beside StatusWhenOff.



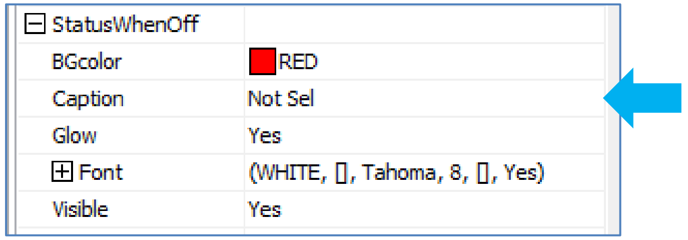
Property	Value
Picture	(None)
⊞ StatusWhenOff	
⊞ StatusWhenOn	

The properties under StatusWhenOff will appear.



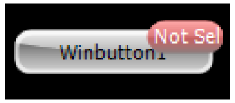
Property	Value
⊟ StatusWhenOff	
BGcolor	🟥 RED
Caption	
Glow	Yes
⊞ Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes
⊞ StatusWhenOn	

Click on the Caption property line, type in the text "Not Sel", then press Enter.



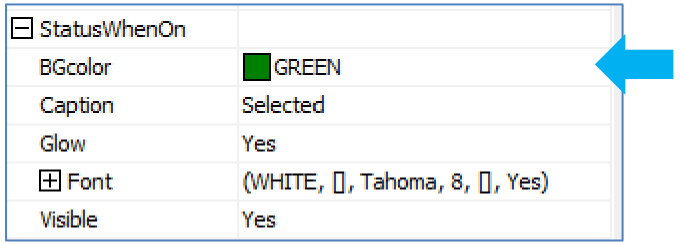
⊟ StatusWhenOff	
BGcolor	🟥 RED
Caption	Not Sel
Glow	Yes
⊞ Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

The WYSIWYG screen will be updated accordingly.



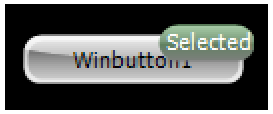
Configure the On State Appearance

Follow the same procedure to create the on state appearance of the button. Use the properties under StatusWhenOn.
Type in "Selected" for the button caption. Also, change the background colour to green.

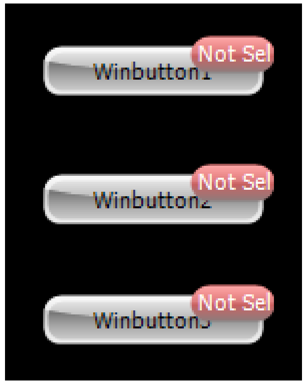


⊟ StatusWhenOn	
BGcolor	🟩 GREEN
Caption	Selected
Glow	Yes
⊞ Font	(WHITE, [], Tahoma, 8, [], Yes)
Visible	Yes

Shown below is the on state appearance of the button. This will be shown when you run the program.

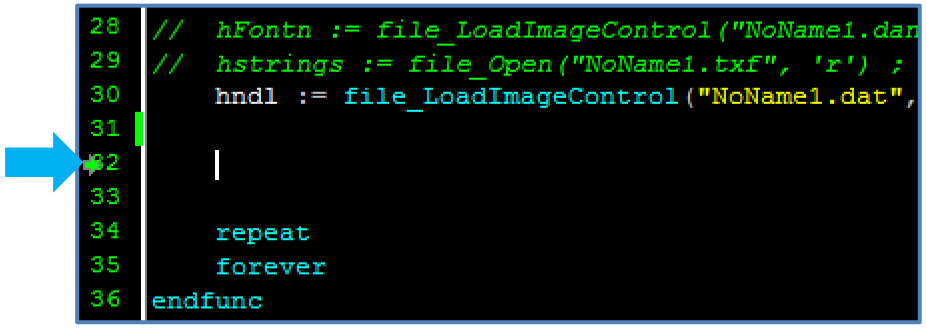


Now create two more instances of this button. After having created three identical buttons, your WYSIWYG screen will look like as shown below:



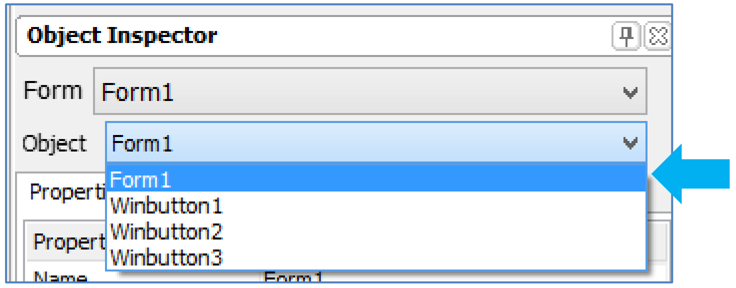
Insert the Code

Go to the code area and place the cursor just after the handle assignment statement (line 32 in this example).



```
28  //   hFontn := file_LoadImageControl("NoName1.dan
29  //   hstrings := file_Open("NoName1.txf", 'r') ;
30       hndl := file_LoadImageControl("NoName1.dat",
31
32       |
33
34       repeat
35       forever
36  endfunc
```

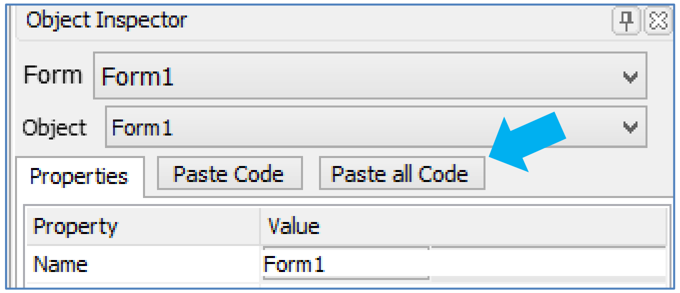
Now click on any part of the form outside the buttons, or choose Form1 as the current object in the Object Inspector.



Object Inspector

Form	Form1
Object	Form1

Now click on the Paste all Code button.

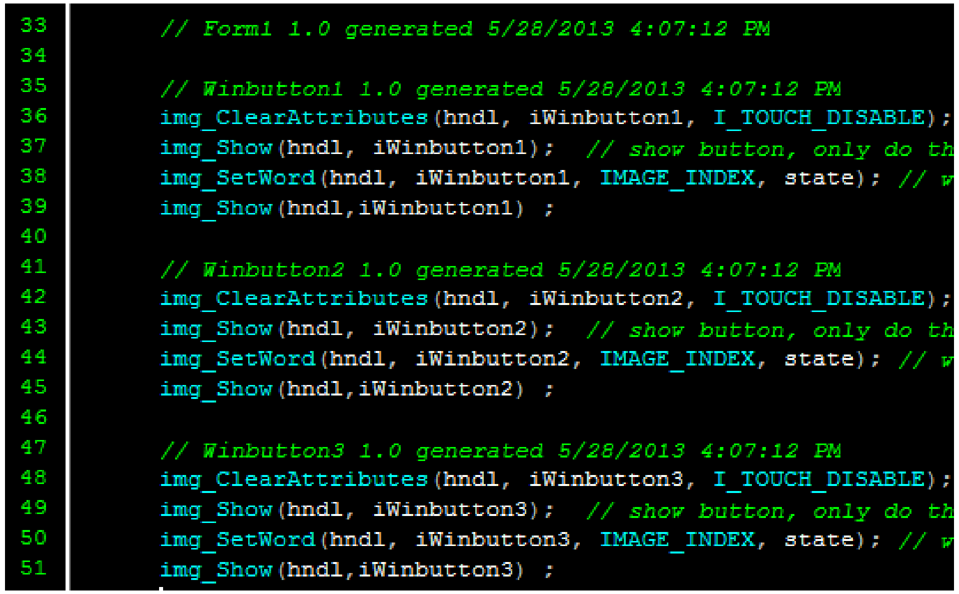


Object Inspector

Form	Form1
Object	Form1

Property	Value
Name	Form1

The code will be updated accordingly. The codes for all the three buttons under Form1 are inserted.



```
33        // Form1 1.0 generated 5/28/2013 4:07:12 PM
34
35        // Winbutton1 1.0 generated 5/28/2013 4:07:12 PM
36        img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE);
37        img_Show(hndl, iWinbutton1);  // show button, only do th
38        img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); // w
39        img_Show(hndl,iWinbutton1) ;
40
41        // Winbutton2 1.0 generated 5/28/2013 4:07:12 PM
42        img_ClearAttributes(hndl, iWinbutton2, I_TOUCH_DISABLE);
43        img_Show(hndl, iWinbutton2);  // show button, only do th
44        img_SetWord(hndl, iWinbutton2, IMAGE_INDEX, state); // w
45        img_Show(hndl,iWinbutton2) ;
46
47        // Winbutton3 1.0 generated 5/28/2013 4:07:12 PM
48        img_ClearAttributes(hndl, iWinbutton3, I_TOUCH_DISABLE);
49        img_Show(hndl, iWinbutton3);  // show button, only do th
50        img_SetWord(hndl, iWinbutton3, IMAGE_INDEX, state); // w
51        img_Show(hndl,iWinbutton3) ;
```

Note that each button has a unique **index** name – **i**Winbutton1 for Winbutton1, **i**Winbutton2 for Winbutton2, and **i**Winbutton3 for Winbutton3. As discussed previously, the functions **img_SetWord(**handle, **index**, offset, word**)** and **img_Show(**handle, **index)** are used to change and show the state of an object. When dealing with multiple objects, make sure that the specified index is correct.

If you wish to test run the program now, comment out lines 38, 39, 44, 45, 50, and 51. As an alternative, analyse the code and declare **state** as a variable. Assign it a value of 0, 1, 2, or 3 at the beginning part of the code. When you run the program, the module will now display the three buttons at the state specified by the variable, **state**.

Index Values

The parameter **index** is an integer, the value of which starts at zero. In this example, Winbutton1 has an index value of 0 since it is the first object created. Winbutton2 has an index value of 1 since it is the second object created, and so on. The statement

```
img_Show(hndl, 0)
```

brings about the same result as the statement

```
img_Show(hndl, iWinbutton1)
```

Therefore, iWinbutton1 is equal to zero. The table below lists the index names and values for the three buttons.

Object	Index name	Index value
Winbutton1	iWinbutton1	0
Winbutton2	iWinbutton2	1
Winbutton3	iWinbutton3	2

If another object is added to the program, say a user LED, Workshop assigns it an index value of 3, it being the fourth object created.

Object	Index name	Index value
Userled1	iUserled1	3

To show the user LED, use the statement

```
img_Show(hndl, iUserled1);
```

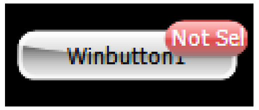
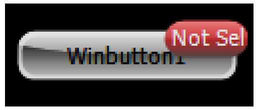
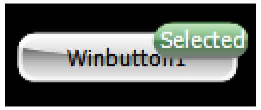
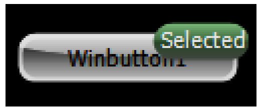
or

```
img_Show(hndl, 3);
```

A Simple Example

After knowing how index values are assigned to objects, it is now possible for the user to program a set of buttons to behave as a group, wherein selecting a button deselects the others. Attached is a simple program, **GroupTutorial.4dViSi,** for a group of three buttons. Note how the index values are assigned to or replaced with different variables.

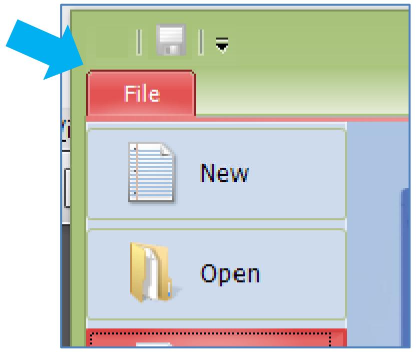
Remember that an on/off button has four states generated by Workshop. In this example, the states are as follows:

	State 0 Off state or not selected state Button appears **light**
	State 1 Off state or not selected state Button appears **dark**
	State 2 On state or selected state Button appears **light**
	State 3 On state or selected state Button appears **dark**

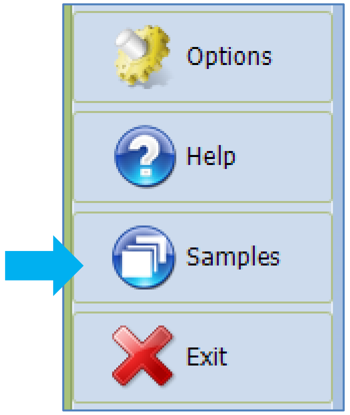
As discussed earlier, the dark appearance of a button can be used to indicate that it is pressed, the light appearance to indicate that is not pressed. To simplify the program however, only states 0 and 2 are used in **GroupTutorial.4dViSi**. The user can add more buttons to the group after having understood how the program works.

Combination of Buttons

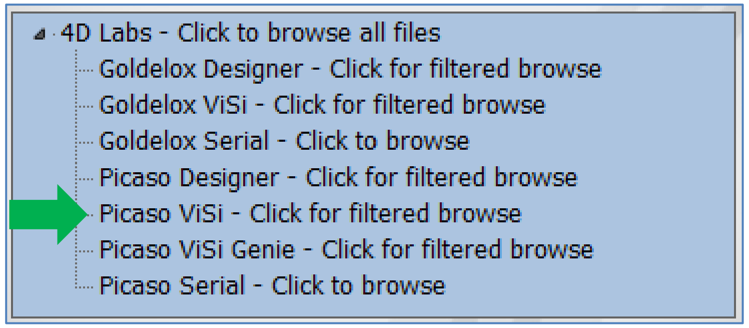
In summary, three kinds of buttons are discussed in this application note - the simple (or momentary) button, the on/off (or toggle) button, and the group of buttons. A ViSi program for each kind is also provided. The codes are heavily commented to help explain how the program works. Now, Workshop comes with many sample programs, one of which is for a combination of these buttons. To open it, click on the File menu.



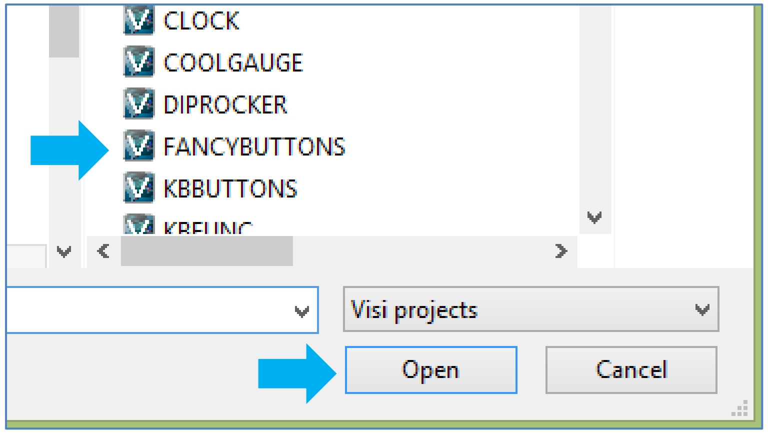
Near the bottom of the drop down menu, you can find the Samples button, click on it.



The samples window now appears. Select Picaso ViSi.


4D Labs - Click to browse all files
Goldelox Designer - Click for filtered browse
Goldelox ViSi - Click for filtered browse
Goldelox Serial - Click to browse
Picaso Designer - Click for filtered browse
Picaso ViSi - Click for filtered browse
Picaso ViSi Genie - Click for filtered browse
Picaso Serial - Click to browse

Select the file FANCYBUTTONS then click on Open.



The program now opens.

Run the Program

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination (this option is not available for Goldelox displays), and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

The uLCD-32PTU and uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability