APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

# ViSi Input Objects

DOCUMENT DATE: **15th APRIL 2019**
DOCUMENT REVISION: **1.1**

## Description

This application note requires:

- Any of the following 4D Picaso and gen4 Picaso display modules:

| | | |
|---|---|---|
| gen4-uLCD-24PT | gen4-uLCD-28PT | gen4-uLCD-32PT |
| uLCD-24PTU | uLCD-32PTU | 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and other superseded modules which support the ViSi Genie environment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-35D 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 / uUSB-PA5/uUSB-PA5-II
  for non-gen4 displays(uLCD-xxx)
- 4D Programming Cable & gen4-PA, / gen4-IB / 4D-UPA
  for gen4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- Any Arduino board with a UART serial port
- 4D Arduino Adaptor Shield (optional) or connecting wires
- Arduino IDE
- 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 is intended to demonstrating to the user the basic of using the input objects of the Workshop IDE ViSi environment. Input objects in ViSi are those that respond to touch such as a slider, track bar, rotary switch, and others. They are useful as a means by which a GUI application gets an input from the user.

# 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**

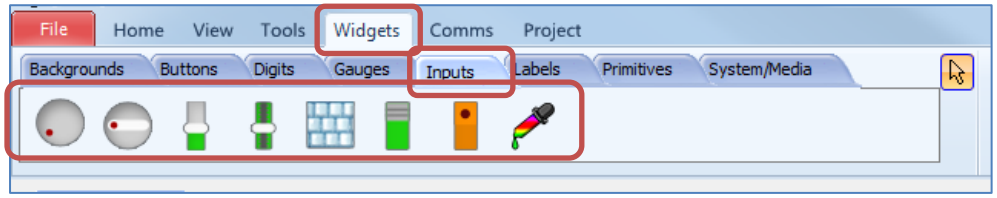
# Create a New Project

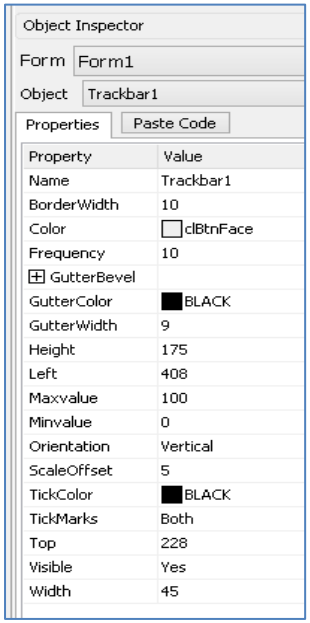
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

**ViSi Getting Started - First Project for Picaso and Diablo16**

## Design the Project

Adding the Input objects on the ViSi environment is fairly simple. Start by accessing the Widgets menu of an open project. WIDGETS >> INPUTS

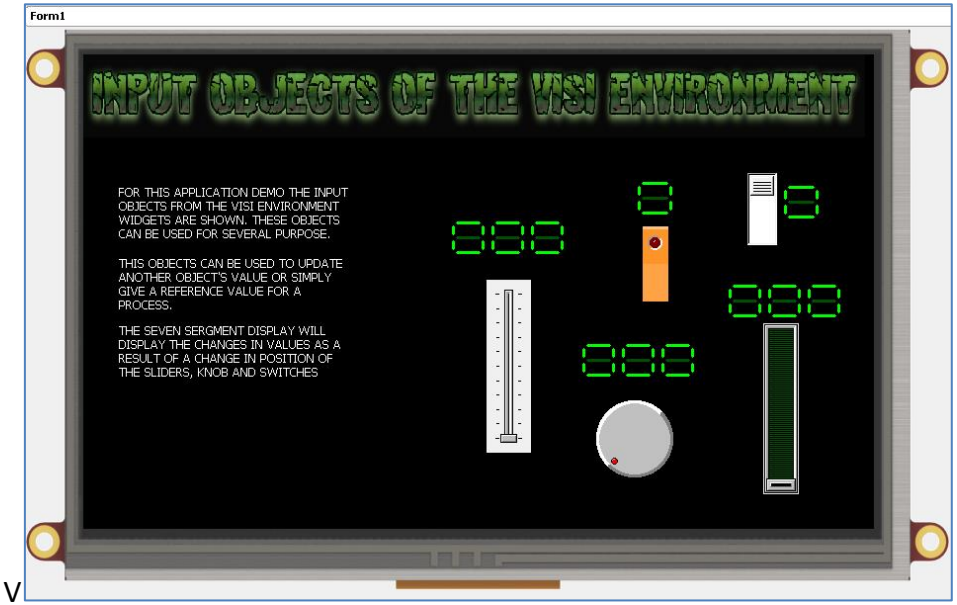



Setting up the objects in the likeness of the project presented here is also easy. The objects can be placed and positioned according to the need of the user. There are particular options on the object inspector that greatly helps the user in achieving their desired graphic user interface.

Let us take the image on the side, we could see here the track bar's properties. Colours and position alongside with other features are enlisted in this object inspector which can be easily changed according to preferences.

### The ViSi-based Inputs Project




V

After all the objects have been laid-out, let's continue with the other half which involves the coding of the project. This will be presented in a sectional manner so as not to create confusion with the project. For an in-depth detail of the functions used in this application note please refer to the DIABLO16 Internal Functions Reference Manual.

### The Include Section

This project starts with the identification of the platform being used as declared by the #platform function. For the program to be able to function properly files are included herein using the #inherit function. Three other files are included automatically during creation of the new project, namely:

4DGL_16bitColours.fnc, the VisualConst.inc, and user_imgConst.inc. Notice that one of the include file is almost the same as the project filename. This file is automatically generated as soon as project saving is done.

INPUTS

```
1  #platform "uLCD-70DT"
2  #inherit "4DGL_16bitColours.fnc"
3  #inherit "VisualConst.inc"
4  #inherit "INPUTSConst.inc"
5  #inherit "XYPOSTODEGREE.inc"
6  #inherit "LEDDIGITSDISPLAY.inc"
```

Other additional include files were added to this project. The 'XYPOSTODEGREE.inc' is an include files dedicated for the proper operation of the KNOB object. Likewise, the 'LEDDIGITSDISPLAY.inc' is needed to enable the use of LED DIGITS properly.

## The Main Program

The main program contains a simple program that first detects and initializes the micro SD card, the initial displaying of the objects used in the project and an endless loop that detects the buttons pressed by the user and subsequently executes other included statements or call out a sub-routine.

## The micro-SD Initialization

Let's start with the initialization of the uSD card. The uSD card contains all the image information about the objects used in the project. The object information and data are saved under a *.DAT and a *.GCI filename extension which is copied to the uSD during project compilation. Mounting of the disk in this application note was done using the following set of program statements.

```
11 func main()
12
13     putstr("Mounting...\n");
14     if (!(disk:=file_Mount()))
15         while(!(disk :=file_Mount()))
16             putstr("Drive not mounted...");
17             pause(200);
18             gfx_Cls();
19             pause(200);
20         wend
21     endif
22
23     hndl := file_LoadImageControl("INPUTS.dat", "INPUTS.gci", 1);
24     gfx_Set(SCREEN_MODE,LANDSCAPE) ;
25     touch_Set(TOUCH_ENABLE);                    // enable the touch screen
```

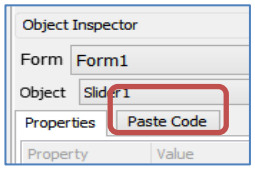
When starting a new project in the ViSi environment these set of statements are already included in the coding area. Another part of these set of statements uses a function file_LoadImageControl() to call on the object data/information files on the uSD drive. This initializes the data to be called in using the variable 'hndl'. In addition, the filenames for the dat and gci files are automatically changed to the project filename as soon as the project is saved.

Added to the statements above are the screen orientation setup function and the touch enable function. The screen orientation is set to portrait thereby providing an 800x480 pixel dimension. This setup is attained using the gfx_Set() function.

Having been able to load and initialize the uSD drive, the processor is now able to access the information stored therein. As mentioned from the previous section, the filenames with an extension of DAT and GCI has the

image data and information. Therefore, the next part of the main program is to display all the objects that were placed on the Workshop IDE form viewer.

A special button from the Object Inspector can help reduce the time of coding of this part. The 'Paste Code" simply pastes object code.




## Displaying the Objects

In this part of the program, the img_Show() function simply calls out the object image and information found in the microSD drive. This set of statements call out the images from their handlers and are displayed on screen. The img_ClearAttributes() function is used to enable the touch feature for the input objects.

```
27      img_Show(hndl,iSlider1) ;
28      img_Show(hndl,iKnob1) ;
29      img_Show(hndl,iTrackbar1) ;
30      img_Show(hndl,iDipswitch1) ;
31      img_Show(hndl,iRockerswitch1) ;
32
33      img_Show(hndl, iLeddigits1);
34      img_Show(hndl, iLeddigits2);
35      img_Show(hndl, iLeddigits3);
36      img_Show(hndl, iLeddigits4);
37      img_Show(hndl, iLeddigits5);
38
39      img_ClearAttributes(hndl, iTrackbar1, I_TOUCH_DISABLE); // set to enable touch, only need
40      img_ClearAttributes(hndl, iKnob1, I_TOUCH_DISABLE); // set to enable touch, only need to d
41      img_ClearAttributes(hndl, iRockerswitch1, I_TOUCH_DISABLE); // set to enable touch, only n
42      img_ClearAttributes(hndl, iDipswitch1, I_TOUCH_DISABLE); // set to enable touch, only need
43      img_ClearAttributes(hndl, iSlider1, I_TOUCH_DISABLE); // set to enable touch, only need to
44
45      img_Show(hndl,iStatictext2) ;
46      img_Show(hndl,iStatictext1) ;
47      img_Show(hndl,iStatictext3) ;
48
49      img_Show(hndl,iImage1) ;
```

### The Repeat-forever Loop

The endless loop repeat-forever contains several statements. The first of the statements contained in this loop enables a constant monitoring of the touch status. The resulting status of the touch is saved on a variable named state. Following this is the variable 'n' that saves the index of the object being touched.

```
51      repeat
52          state := touch_Get(TOUCH_STATUS);              // get touchscreen status
53          n := img_Touched(hndl,-1) ;
54          //-------------------------------------------------------------------------
55          if(state == TOUCH_PRESSED)                      // if there's a press
56              x := touch_Get(TOUCH_GETX);
57              y := touch_Get(TOUCH_GETY);
58                      |
59              if(n == iRockerswitch1)
60                  rockerPos := !rockerPos & 1 ;        // just toggle for clickrect = v
61                  img_SetWord(hndl, iRockerswitch1, IMAGE_INDEX, rockerPos) ;      // v
62                  img_Show(hndl,iRockerswitch1) ;
63
64                  img_Show(hndl, iLeddigits5);  // show all digits at 0, only do this
65                  ledDigitsDisplay(rockerPos, iLeddigits5+1, 560, 1, 1, 43, 0) ;
66
67              endif
68
69          //-------------------------------------------------------------------------
70          else if(state == TOUCH_RELEASED)                    // if there's a release
71
72
73          //-------------------------------------------------------------------------
74          else if(state == TOUCH_MOVING)                      // if it's moving
75              x := touch_Get(TOUCH_GETX);
76              y := touch_Get(TOUCH_GETY);
77
78              if(n == iTrackbar1)     trackbar();
79              if(n == iKnob1)         knob();
80              if(n == iSlider1)       slider();
81              if(n == iDipswitch1)    dipswitch();
82
83          endif
84      forever
```

Inside the repeat-forever loop are the touch related conditions that involve – a 'touch_pressed', a 'touch_released' and a 'touch_moving'. Each of this conditions contains several other statements that are related to the input objects that were placed into the project.

For the touch_pressed condition, if a press is detected on the touch panel the next to be considered is the object that was touched. This is returned to the variable n.

```
55          if(state == TOUCH_PRESSED)                      // if there's a press
56              x := touch_Get(TOUCH_GETX);
57              y := touch_Get(TOUCH_GETY);
58                      |
59              if(n == iRockerswitch1)
60                  rockerPos := !rockerPos & 1 ;        // just toggle for clickrect = v
61                  img_SetWord(hndl, iRockerswitch1, IMAGE_INDEX, rockerPos) ;      // v
62                  img_Show(hndl,iRockerswitch1) ;
63
64                  img_Show(hndl, iLeddigits5);  // show all digits at 0, only do this
65                  ledDigitsDisplay(rockerPos, iLeddigits5+1, 560, 1, 1, 43, 0) ;
66
67              endif
```

Within the touch_pressed condition, if the rockerswitch is being set to the compliment of its initial frame position- given by the rockerPos value. The value is a transitioned between 1 and 0 each time the object is touched therefore giving a toggle effect.

The next of the touch-related conditions is the touch_moving condition. This condition has several statements that are related to the other four input objects. If a touch_moving condition is detected over an object this results to the call and execution of the sub-routine.

## The trackbar() and slider() Sub-routines

When a 'touch_moving' condition is detected over the track bar input object, this results to the call and execution of a sub-routine named trackbar(). The sub-routine contains a code automatically generated by the Object Inspector – 'Paste Code'.

```
88  func trackbar()
89      var posn;
90
91      img_Show(hndl,iTrackbar1) ; // show initialy, if required
92      img_ClearAttributes(hndl, iTrackbar1, I_TOUCH_DISABLE); // set to enable touch, only
93      posn := y - 242 ;                       // y - top - borderwith - 4
94      if (posn < 0)
95          posn := 100 ;                       // maxvalue-minvalue
96      else if (posn > 147)                    // height - 2*borderwidth - 8
97          posn := 0 ;
98      else
99          posn := 100 - 100 * posn / 147 ;    // max-min - (max-min) * posn / (height-2*bo
100     endif
101     img_SetWord(hndl, iTrackbar1, IMAGE_INDEX, posn);
102     img_Show(hndl, iTrackbar1);
103
104     img_Show(hndl, iLeddigits2);
105     ledDigitsDisplay(posn, iLeddigits2+1, 372, 3, 1, 40, 0) ;
106 endfunc
```

For the generated code on the trackbar() sub-routine, the variable 'posn' should be declared. The content of the trackbar() sub-routine is a conditional calculation of the current value for a trackbar object. The resulting value of the calculation is made the effective frame number of the trackbar. The same value is then made to be the effective value for the LedDigits object.

The same routine is effective for the slider object. The trackbar and slider input objects have the same manner of calculation. The only difference is the reference starting value for the x and y position. This is the result of the different position by which it is placed.

## The dipswitch() Sub-routine

As compared to the previous sub-routine, the dipswitch() sub-routine is much simpler. Due to the limited number of positions that is takes it is relatively easier to understand the automatically generated object codes of the dip switch object. The simple calculation is still included to set the object into the desired switch position.

```
155 func dipswitch()
156     var state;
157     state := (y - 148) / 16 ;          // (y - top_left) / (height_width / positions)
158     if (state > 2) state := 2 ;        // if positions > 2 and height not divisible by pos
159     img_SetWord(hndl, iDipswitch1, IMAGE_INDEX, state) ; // where state is 0 to 2
160     img_Show(hndl,iDipswitch1) ;
161
162     img_Show(hndl, iLeddigits4);  // show all digits at 0, only do this once
163     ledDigitsDisplay(state, iLeddigits4+1, 708, 1, 1, 43, 0) ;
164
165 endfunc
166
```

The resulting value of the switch position is saved on the variable 'state'. This is then used to the Led digit object for a better visual of the current position.

## The knob() Sub-routine

This subroutine is quite different from the other input objects. It involves conversion of x and y coordinate values into angular position. Using the detected result for x and y under the touch_moving condition, this is processed with a conditional mathematical set of equations to convert this x/y value to position to an angular position in degrees saved in the variable 'posit'.

```
130 func knob()
131     var posit;
132
133     img_Show(hndl,iKnob1) ; // show initialy, if required
134     img_ClearAttributes(hndl, iKnob1, I_TOUCH_DISABLE); // set to enable touch, only need to
135     degrees := XYposToDegree(x-558,      // x - CentreX
136                              y-390) ;    // y - centreY
137     if (degrees < 45)                    // anything in the first 'dead zone' is minvalue
138         degrees := 0 ;
139     else
140         if (degrees > 315)               // anything in the last 'dead zone' is maxvalue
141             degrees := 270 ;
142         else
143             degrees -= 45 ;              // offset by -baseangle
144         endif
145     endif
146     posit := degrees * 100 / 270 ;       // convert degrees to position
147     img_SetWord(hndl, iKnob1, IMAGE_INDEX, posit);
148     img_Show(hndl, iKnob1);
149
150     img_Show(hndl, iLeddigits1);  // show all digits at 0, only do this once
151     ledDigitsDisplay(posit, iLeddigits1+1, 504, 3, 1, 40, 0) ;
152
153 endfunc
```

Similar to the other objects, this is transferred to the led digits to display the value in degree units.

## 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