



ViSi Gauges Objects Part II

DOCUMENT DATE: **16th 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and other superseded modules which support the ViSi Genie environment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](#)

- [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

Description	2
Content	2
Application Overview	3
Setup Procedure	3
Create a New Project	3
Design the Project	3
<i>The ViSi-based Gauge Application Project</i>	4
The include section	4
The main program	4
The micro-SD initialization	4
Displaying the objects	5
The repeat-forever loop	5
Run the Program	6
Proprietary Information	7
Disclaimer of Warranties & Limitation of Liability	7

Application Overview

This document is mainly focused on showing the simple use of the Gauges object of the ViSi Genie environment of the Workshop IDE. This objects are very useful in providing a real-life like visual for displaying temperature, volume, content percentage and many more. Depending on the user's preferences, colours, meter values, and meter names/units can be set using the objects' properties tab in the Object Inspector of the ViSi environment.

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

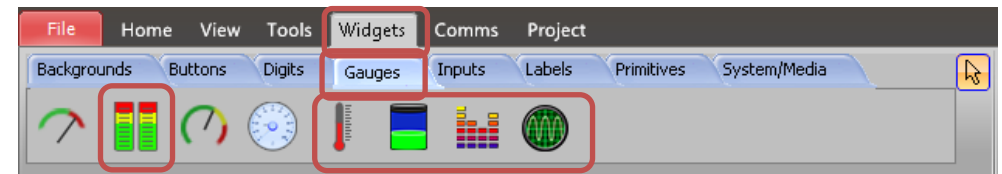
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

Design the Project

Adding the Gauges objects for a project created in the ViSi environment is relatively easy. The Gauges tab can be located under the Widgets menu.

Widgets >> Gauges >> Thermometer / Tank / Spectrum/Gauge



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

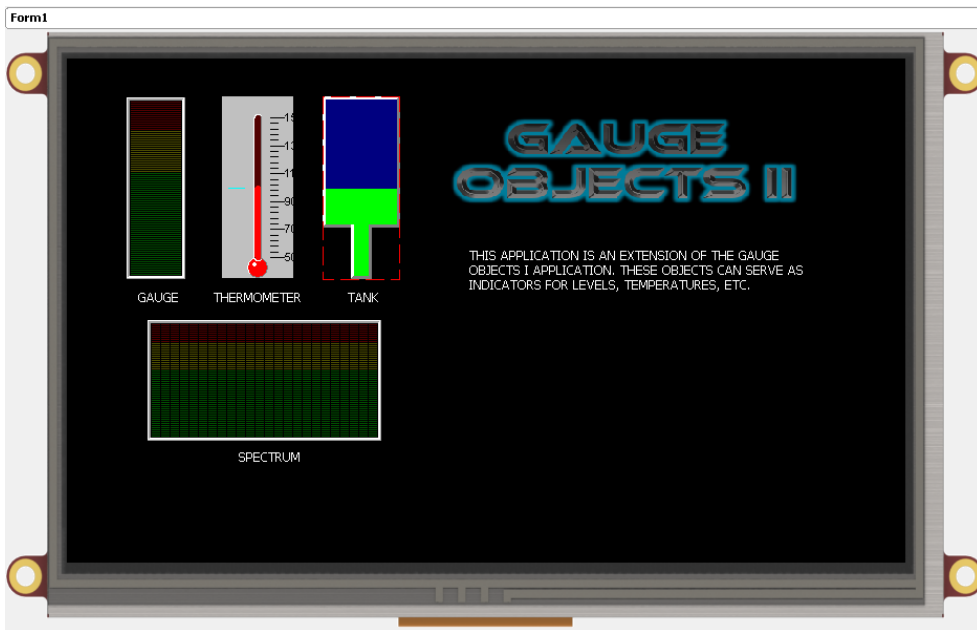
user. There are particular options on the object inspector that greatly helps the user in achieving their desired graphic user interface.

Object Inspector	
Form	Form1
Object	Tank1
Properties Paste Code	
Property	Value
Name	Tank1
Color	BLACK
Container	
EmptyColor	NAVY
FillColor	LIME
Height	175
Left	244
Max	100
Min	0
Position	50
Shape	Rectangle
Top	36
Width	74

Each of the objects can be set to a certain colour, size, etc., according to the enlisted properties available to a particular object. Note that each object are made uniquely, consequently, each object will have a property that is not present in the other. Also, since the objects are made programmatically, it is therefore conclusive that there are certain limitation to the objects.

The properties shown on the side shows that of the tank object.

The ViSi-based Gauge Application Project



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.

```
spectrum x
1 #platform "uLCD-70DT"
2 #inherit "4DGL_16bitColours.fnc"
3 #inherit "VisualConst.inc"
4 #inherit "spectrumConst.inc"
```

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 includes a random number from 0 to 100 in value. The generated value is saved on a global variable named 'frame' and is applied to the gauges objects as their respective sub-routine is called and executed.

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.

```

6  var bar, frame;
7
8  func main()
9
10     putstr("Mounting...\n");
11     if (!disk:=file_Mount())
12         while(!disk:=file_Mount())
13             putstr("Drive not mounted...");
14             pause(200);
15             gfx_Cls();
16             pause(200);
17         wend
18     endif
19     gfx_Cls();
20     hndl := file_LoadImageControl("spectrum.dat", "spectrum.gci", 1);

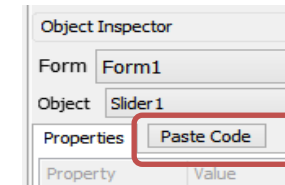
```

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.

The screen orientation is set to portrait thereby providing an 800x480 pixel dimension. This setup is attained using the `gfx_Set()` function. A `gfx_Cls()` function is inserted after the statements for mounting the micro-SD drive. This is done to clear the displayed 'Drive not mounted' message following the completion of the micro-SD initialization.

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 micro-SD drive. This set of statements call out the images from their handler and are displayed on screen. These set of statements plainly display the images as how they are placed into the form viewer of the Workshop IDE- ViSi environment.

```

22  img_Show(hndl,iStatistic1);
23  img_Show(hndl,iStatistic2);
24  img_Show(hndl,iStatistic3);
25  img_Show(hndl,iStatistic4);
26  img_Show(hndl,iImage1);
27  img_Show(hndl,iStatistic5);
28  img_Show(hndl,iGauge1);
29  img_Show(hndl,iThermometer1);
30  img_Show(hndl,iTank1);
31
32  for(bar:=0; bar <= 23; bar++)
33      img_SetWord(hndl, iiSpectrum1, IMAGE_XPOS, 81 + bar * 9); // where bar is 0 to 23
34      img_SetWord(hndl, iiSpectrum1, IMAGE_INDEX, frame); // where frame is 0 to 100 (for a display)
35      img_Show(hndl,iiSpectrum1);
36  next

```

Notice that a for-next loop is included in the statements shown above. The for-next loop was used to display the individual segments of the Spectrum object. The number of segments of the Spectrum objects is marked by the variable 'bar'.

The repeat-forever loop

The endless loop repeat-forever contains several statements. The first of these statements contains a random number generating statement. The function `RAND()` simple generates a random number that is both signed and unsigned. If merged

with a modulo-100 this limits the signed numbers produced to be between -100 and 100. To make this value acceptable for the gauge objects another function is used to convert all the generated random number into positive – the ABS() function.

```

40 repeat
41   frame := ABS(RAND() % 100);
42
43   img_SetWord(hndl, iGaugel, IMAGE_INDEX, frame) ; // where frame is 0 to 100 (for a displayed 0 to 100)
44   img_Show(hndl,iGaugel) ;
45
46   img_SetWord(hndl, iThermometer1, IMAGE_INDEX, frame) ; // where frame is 0 to 100 (for a displayed -1 to -1)
47   img_Show(hndl,iThermometer1) ;
48
49   img_SetWord(hndl, iTank1, IMAGE_INDEX, frame) ; // where frame is 0 to 100 (for a displayed 0 to 100)
50   img_Show(hndl,iTank1) ;
51

```

The random number generated from the combination of the ABS() and RAND() function is saved to the variable 'frame'. This value is then applied to the Gauge, Thermometer and Tank objects. This change in frame value will result to an equal change in the gauge, thermometer and tank object display.

Another part within the repeat-forever loop is the setting of random values to the spectrum object. Refer to the image below.

```

52 for(bar :=0; bar <= 23; bar++)
53   frame := ABS(RAND() % 100);
54   img_SetWord(hndl, iISpectrum1, IMAGE_XPOS, 81 + bar * 9) ; // where bar is 0 to 23
55   img_SetWord(hndl, iISpectrum1, IMAGE_INDEX, frame) ; // where frame is 0 to 100 (for a displayed 0 to 100)
56   img_Show(hndl,iISpectrum1) ;
57
58 next
59
60 forever

```

Since the spectrum object is comprised of several individual segments, setting values to the spectrum will require two data. The first data needed is the 'bar' value. The 'bar' value represents the column number within the spectrum. And another value is the 'frame' value. Using a for-loop, these pair of data is applied to the spectrum object. Each time the for-loop increments in value a new frame value

is generated by the ABS() and RAND(). The result of the loop and the random data will be a spectrum object with different values for each segment.

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

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.