APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi Custom Gauges

DOCUMENT DATE: **20th May 2019**
DOCUMENT REVISION: **1.1**

Description

This Application note is intended to demonstrating to the user the basic on how to create a custom gauge object. The 4DGL code of the Designer project can be copied and pasted to an empty ViSi project and it will compile normally. The code can also be integrated to that of an existing ViSi project.

Before getting started, the following are required:

- Any of the following Picaso display modules:

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

and other superseded display modules which support the ViSi environment

- The target module can also be a Diablo16 display

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

Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

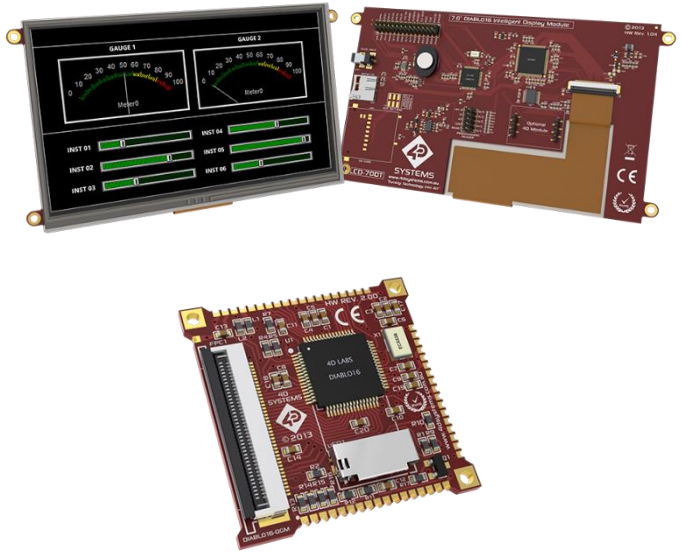
- 4D Programming Cable / µUSB-PA5/µUSB-PA5-II
  for non-gen4 displays (uLCD-xxx)
- 4D Programming Cable & gen4-IB / gen4-PA / 4D-UPA,
  for gen-4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- When downloading an application 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 document is mainly focused on demonstrating to the user the basic on how to create custom gauge objects. Customizing gauge objects requires the user to use other tools such as photo editors such as the Adobe Photoshop to enable creation of image frames. Presented in this document is the use of the User Images object of the ViSi environment of the Workshop IDE to create user custom gauges. Also, an alternative way of creating custom gauges with the use of the Graphics Composer is presented in this document.

The DIABLO16 Embedded Graphics Processor



Driving the display and peripherals is the DIABLO16 embedded graphics processor, a very capable and powerful chip which enables stand-alone functionality, programmed using the 4D Systems Workshop 4 IDE Software. The Workshop IDE enables graphic solutions to be constructed rapidly and with ease due to its design being solely for 4D's graphics processors.

The DIABLO16 Processor offers considerable FLASH and RAM upgrades over the PICASO processor, and also provides map-able functions such as I2C, SPI, Serial, PWM, Pulse Out, and Quadrature Input, to various GPIO, and also provide up to 4 Analogue Input channels.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **Designer** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note
Designer Getting Started - First Project

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

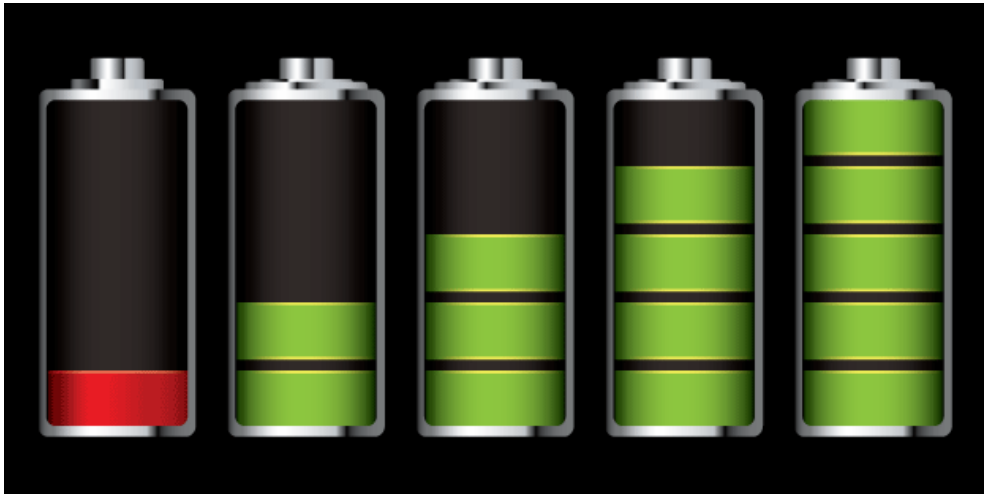
For instructions on how to create a new **Designer** project, please refer to the section "**Create a New Project**" of the application note
Designer Getting Started - First Project

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

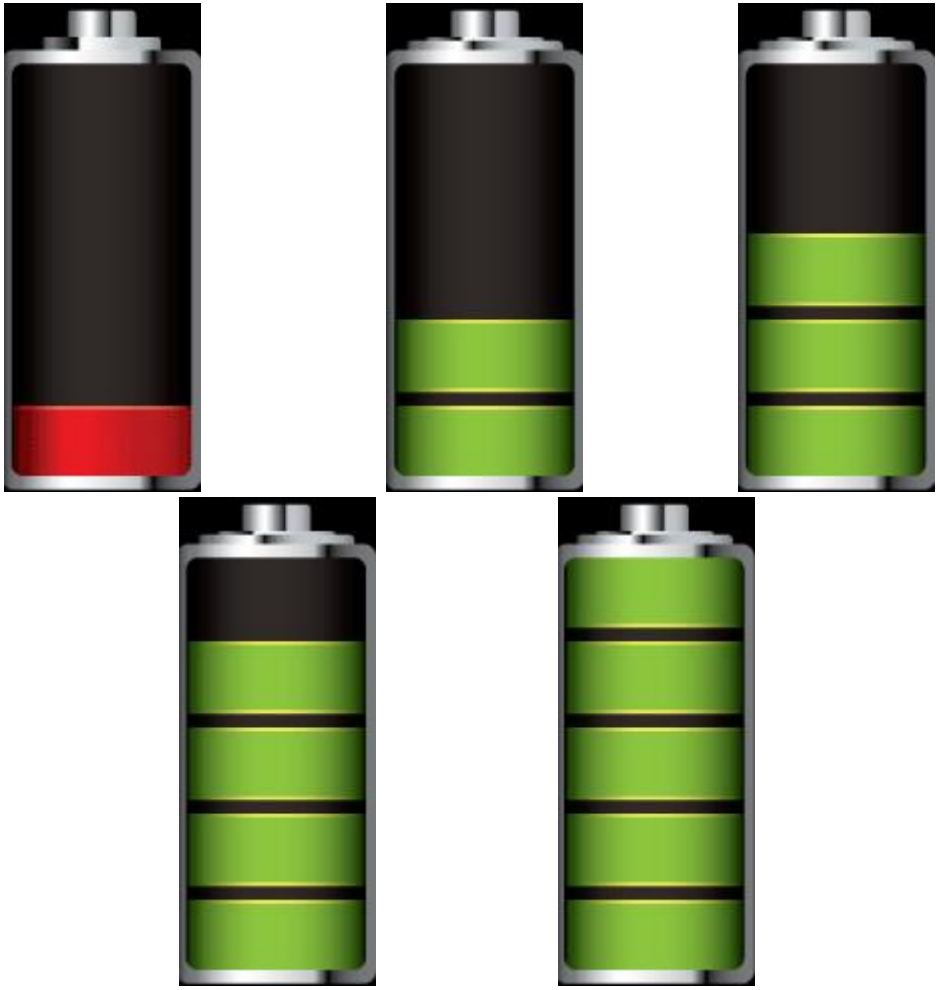
Design the Project

How to create an image file for the custom gauge object.

In this section the user will need to make use of an image editing software to create the custom gauge. Referring to the image below, it is shown that a custom battery gauge will display a grade of 20 percent battery level. The levelling starts from a critical level of 0~20 %, 20~40% and so on. To start making the images need we need to splice the images below exactly the same as the other. This will give the effect of animating the red/green bars inside the battery figure.



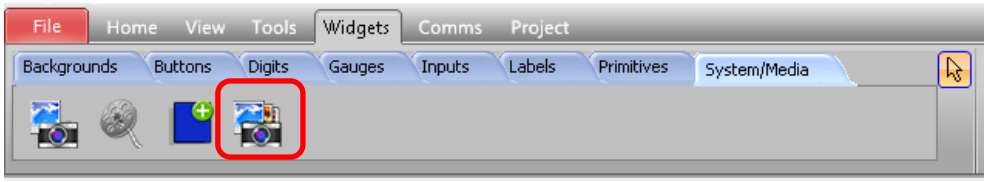
Referring to the images on the next column, we could see that the images are spliced in the same dimension at the other. For best result in creating images, build the images in proportion to the display module or according to user's preference.



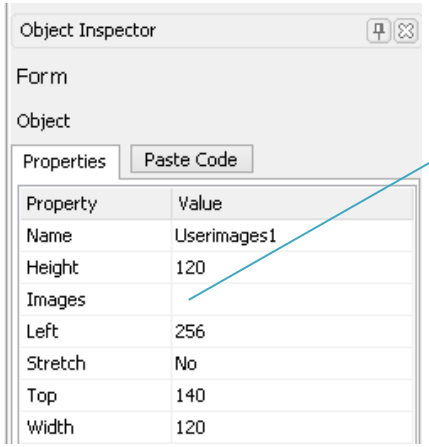
** Observe that the images to be used in the next section is the same in dimension as the other.

Custom Gauges using the User Images object

Another way of creating a custom gauge object is through the use of the User Image Object. The User Images allows addition of several images all at the same time and addition afterwards.

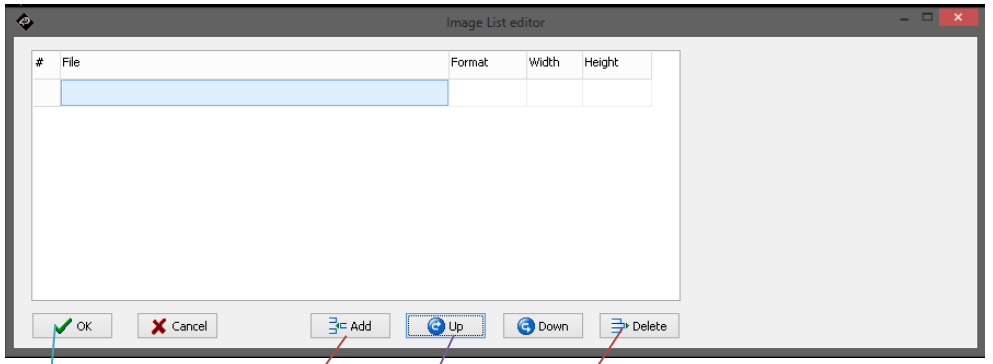


When a User Image object is first added in the project form, a broken line boarder is seen. Adding the images to the object is done in the Object Inspector properties.



Click on the “Images ” option on the properties.

Clicking on the ‘Images’ will open a new window. This new window enables the user to locate the images that are to be added into the project.

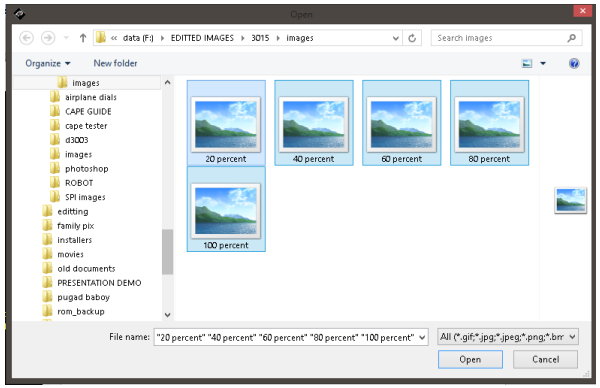


Removes added image(s) from the project

Up and down enables ‘frame’ number arrangements

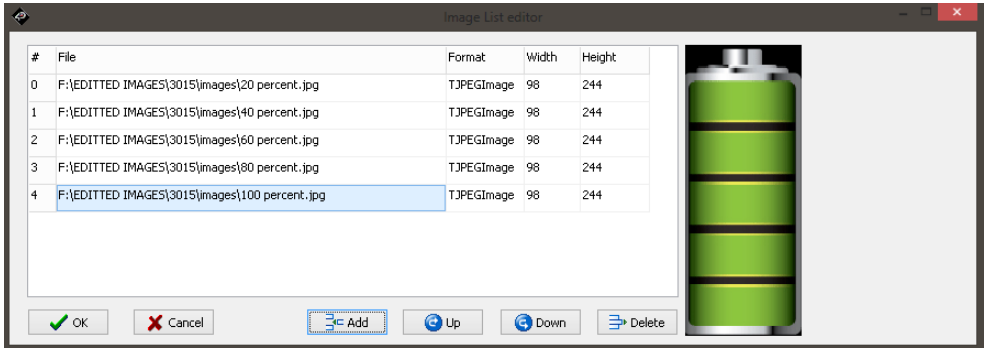
Locate and add image files to the project

Saves the added images to the project

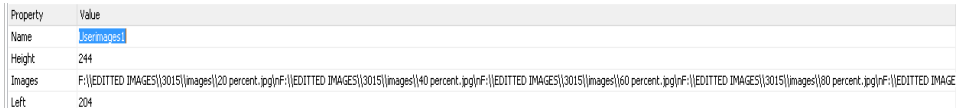


After clicking the ‘Add’ button, another window will appear enabling the user to locate the images. Several files can be selected at the same time and be added to the project. Clicking open will then result to the enlisting of the image files in the ‘Image List Editor’ window.

After all the files are added, the images can be arranged using the up/down button. The number on the first column identifies the frame number of the image file. Hitting the 'OK' button finalizes the addition and arranging of images.



Going back to the Object inspector, the once empty 'Images' options will now be populated with the image filename and location similar to the image show below.

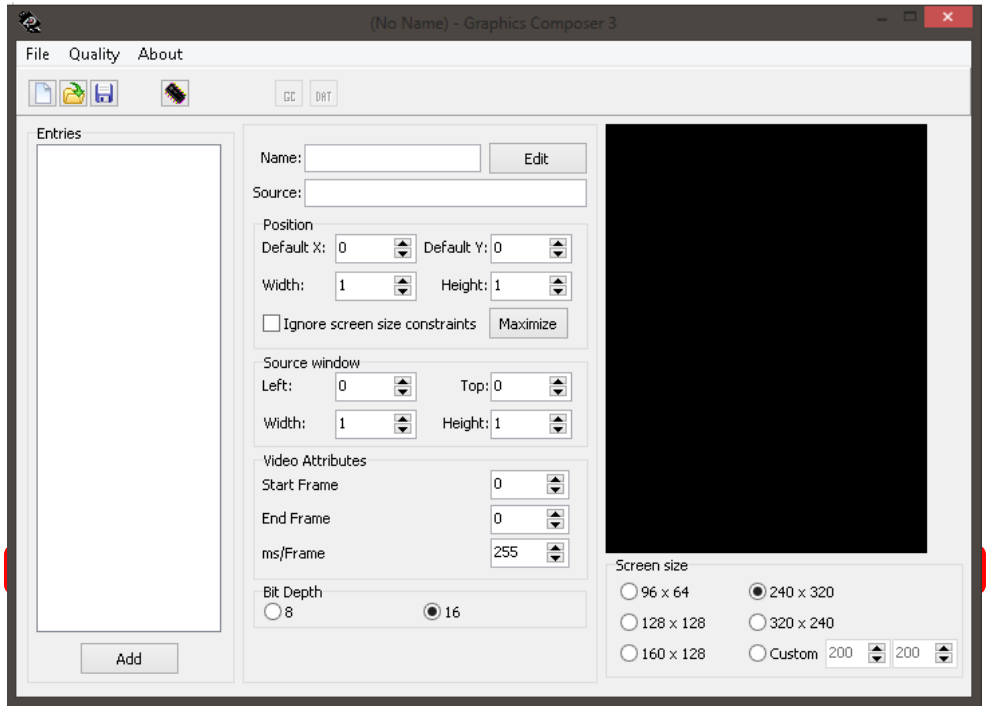


Following all the steps discussed herein will have the 'User Image' objects ready for use in the coding area.

** NOTE: ALTHOUGH APPLICABLE, THE FOLLOWING METHOD IS **NOT ENCOURAGED** FOR CUSTOM OBJECT CREATION FOR DIABLO16. IT IS BEST FOR PICASO RAW OR SGC TYPE APPLICATIONS.

Creating the Custom Gauge Object

Custom gauges can be easily created using Graphics Composer software. The Graphics composer, which is shown below, is the software tool needed to create GCI and DAT file of the custom gauge.

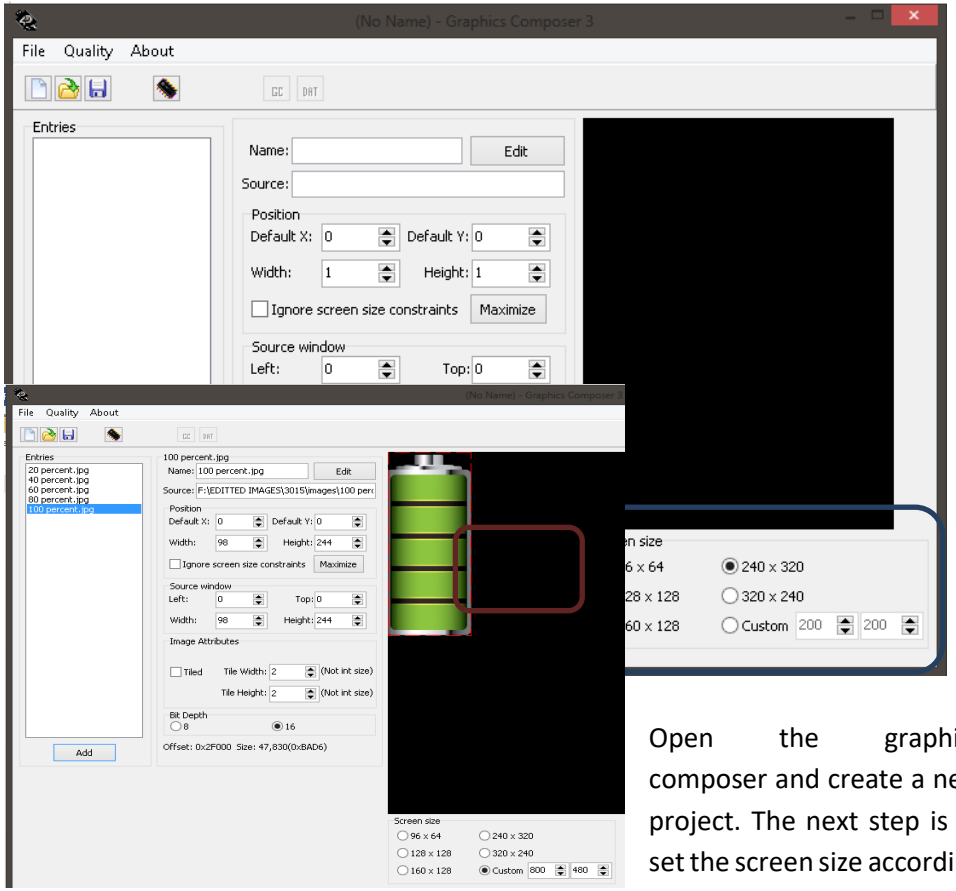


The steps on how to use this software tool will be shown later on this document. The first things needed is to have the images that is needed to create the custom gauge. Let us take a battery level gauge indicator as an example.

The custom battery gauge level indicator that is to be made in this application document is in a way the same as the native gauge objects of the Workshop IDE. The only difference between these two objects is that, the native gauge objects are

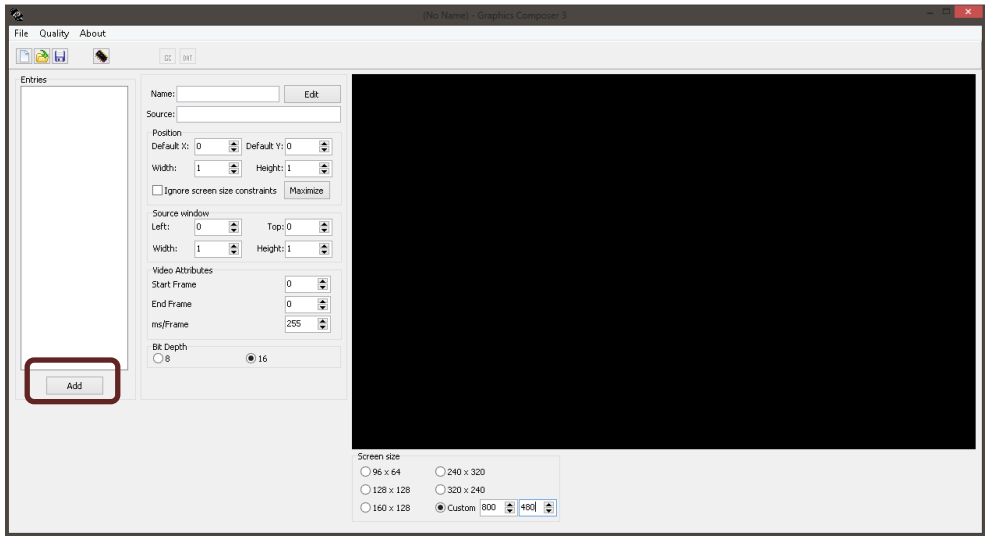
programmatically created and generated by the Workshop IDE, while a custom gauge object would require images and image files manually created by the user.

Creating the GCI and DAT file using GRAPHICS COMPOSER

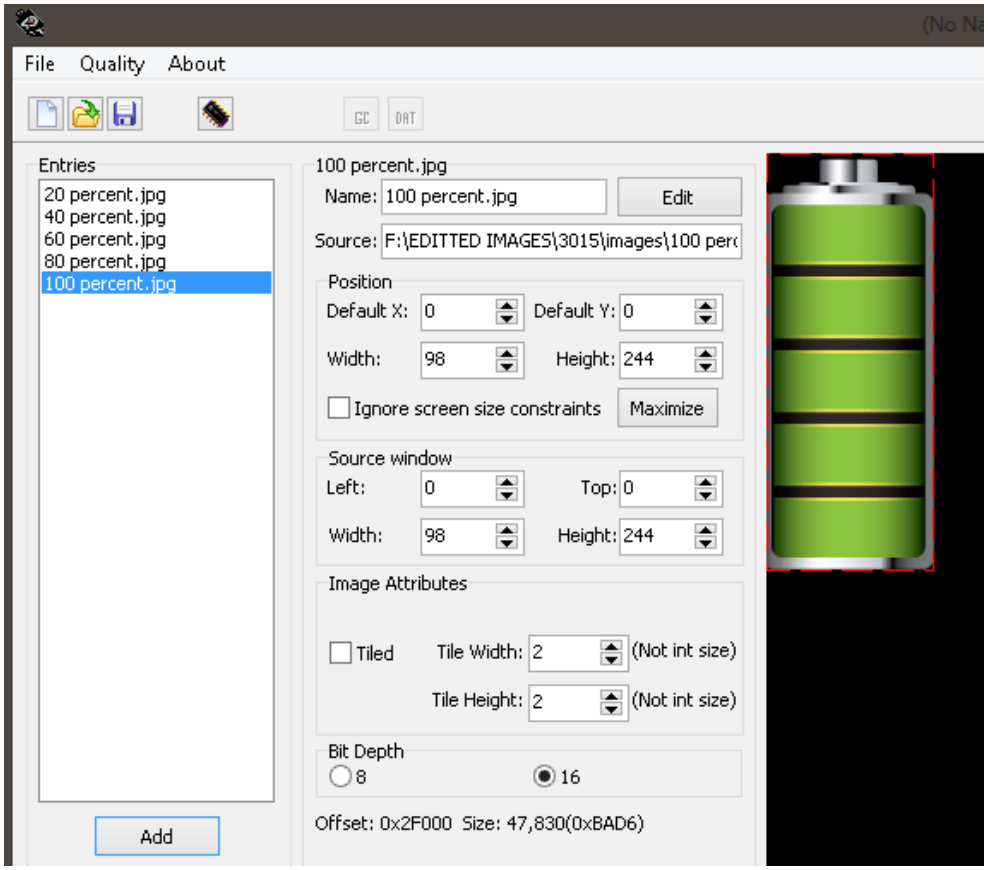


Open the graphics composer and create a new project. The next step is to set the screen size according to the dimension of the display module that user prefers to use. Make sure that the orientation is also followed so that the image is created according to the preferred display setting.

**Note: The displays may be different but the process of creating the files needed for custom gauges and other animated object follows the same procedure.

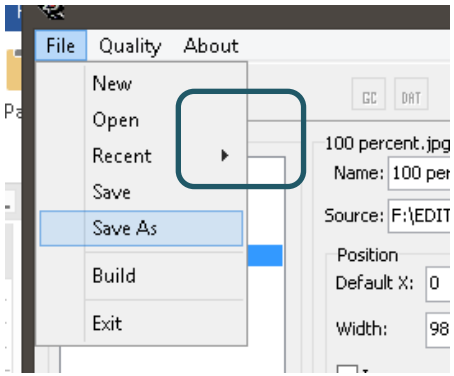


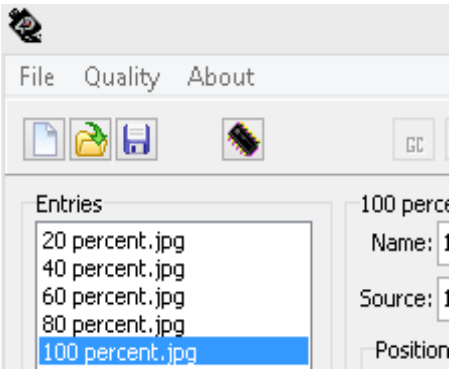
After the adjustment of the screen size, click on the “ADD’ button to include the image files of the custom battery gauge. A pop-up window will appear. Using this window locate the images that are needed for the battery project.

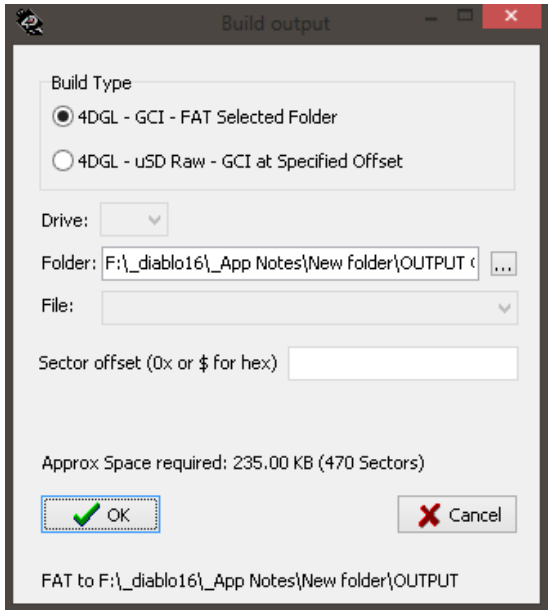


The files added in the Graphics Composer are now listed in the Entries section. Changing the position of the images within the screen can be an option but this could also be done inside the ViSi-based program.

Having been able to add the images, we first need to save the project into a filename which will be used for the output GCI and DAT files. To avoid any problems it is a good practice to simply name this with a limit of 8 characters. For this project we will save it as – BATTERY.







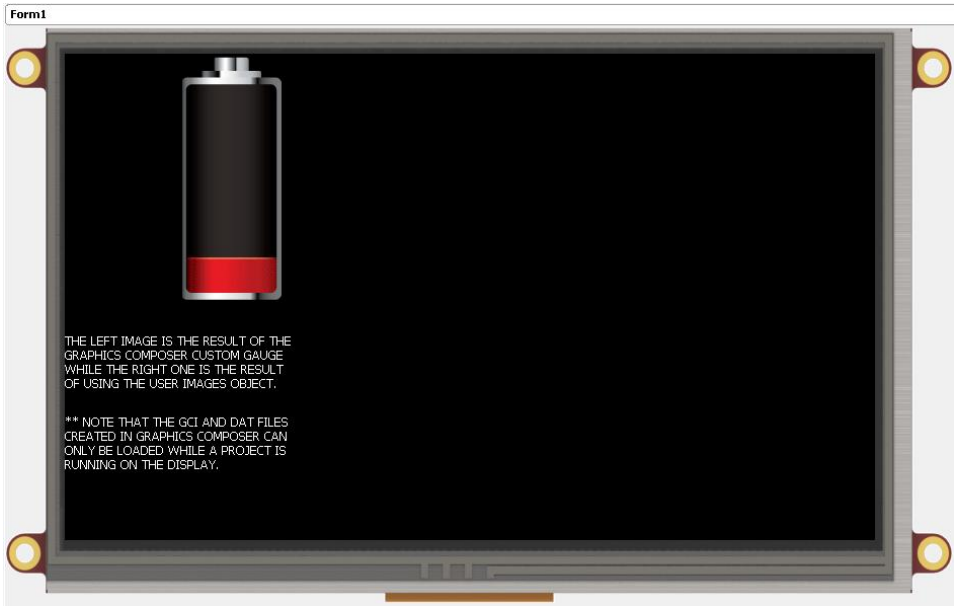
After saving the file as BATTERY. We now proceed to creating the GCI and DAT files. This is done by first clicking on the "BUILD" button. Clicking this button will result to a pop-up window which will ask the user the type of build preferred.

**This build type is relative to the FAT/RAW file system of the micro-SD drive used.

Unless the user wishes to make changes clicking 'OK' will result in the creation of the files in the output folder specified. This output folder is automatically located to be the same as where the project was initially saved.

** Now that we have the GCI and DAT files, copy these files to a micro-SD that will be used for the project.

The ViSi-based Custom Gauge Application Project



After the user images object 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.

** NOTICE THAT THE IMAGES CREATED USING THE 'GRAPHICS COMPOSER' IS NOT DISPLAYED ON THE SCREEN. A GRAPHICS COMPOSER CREATED FILE CAN ONLY BE DISPLAYED DURING A PROGRAM RUN.

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.

C_GAUGES*

```
1  #platform "uLCD-70DT"
2  #inherit "4DGL_16bitColours.fnc"
3  #inherit "VisualConst.inc"
4  #inherit "C_GAUGESConst.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 for-loop counts of 0 to 4. The generated value is used as the IMAGE_INDEX reference number.

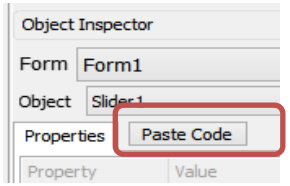
The micro-SD initialization

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   func main()
7       var custom;
8       putstr("Mounting...\n");
9       if (!(disk:=file_Mount()))
10          while(!(disk :=file_Mount()))
11              putstr("Drive not mounted...");
12              pause(200);
13              gfx_Cls();
14              pause(200);
15          wend
16      endif
17
18
19      hndl := file_LoadImageControl("C_GAUGES.dat", "C_GAUGES.gci", 1);
20      custom := file_LoadImageControl("BATTERY.dat", "BATTERY.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 additional declared variable 'custom' will be used as the handler variable for the BATTERY.gci and BATTERY.dat files.

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(custom, 0);      // custom gauge using graphics composer
23      img_Show(hndl, 0);        // custom gauge using User Images object
```

The repeat-forever loop

```
26      repeat
27      var x;
28      for(x := 0; x <= 4; x ++)
29          img_Show(custom, x);  // custom gauge using graphics composer
30          pause(1000);
31      next
32
33      for(x := 0; x <= 4; x ++)
34          img_SetWord(hndl, iUserimages1, IMAGE_INDEX,x);
35          img_Show(hndl, iUserimages1);       // custom gauge using User Images object
36          pause(1000);
37      next
38      forever
```

The statements contained inside the repeat-forever loop simply runs both the custom made gauges into a cyclic pattern of changing the IMAGE_INDEX number from the 0 to 4. Building and downloading the project will result to a screen image

similar to what is shown below with an animated transition from image index 0 to 4 repeatedly and alternately.

** NOTE: The left image is the GCI and DAT output image which was made using the graphics composer and programmatically loaded into the image variable handler- custom. GCI and DAT files created using the Graphics Composer can only be called into a project programmatically but these files can never be displayed in the form viewer during the course of project development. The image on the right is the custom gauge created using the User Image object which is native to the ViSi environment of the Workshop-IDE.

Run the Program

For instructions on how to save a **Designer** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

Designer Getting Started - First Project

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination, 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