APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Getting Started – First Project for Picaso and Diablo16

DOCUMENT DATE: **8th MAY 2020**
DOCUMENT REVISION: **1.2**

Description

This application note shows how to program a 4D display module in the ViSi environment to make it print text and display an object on the screen. Before getting started, the following are required:

- 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  and other superseded modules which support the ViSi Genie environment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

It is often difficult to design a graphical display without being able to see the immediate results of the application code. ViSi is the perfect environment that allows users to see the instant results of their desired graphical layout. There is a selection of inbuilt dials, gauges, and meters (called widgets or objects) that can simply be dragged and dropped onto the simulated module display. From here, each object can have its properties edited and at the click of a button, all relevant code is produced in the user program.

This application note shows how to create a new project, how to select the target display module, how to connect a display module to the PC, and how to compile and download a simple “Hello-world” program to the target device. This application note also introduces the 4DGL (4D Graphics Language) and the basic use of the WYSIWYG (What-You-See-Is-What-You-Get) screen.

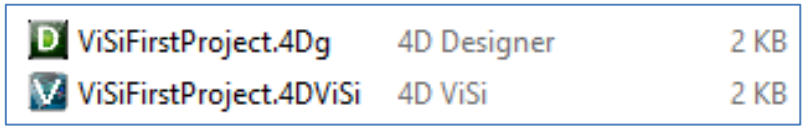
This application note uses a uLCD-32DT display module, which is, as of writing, a discontinued product, as the example target device. The procedures described, however, are applicable to Picaso and other Diablo16 displays.

Setup Procedure

Open a Project from the File Explorer

This document comes with a demo ViSi project in a zip file.

In the file explorer window, extract the contents of the zip file to a desired location. The contents are a ViSi file and a Designer file.



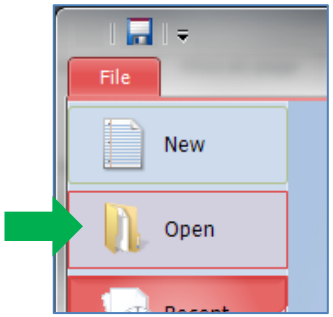
The user can open the project by double-clicking on any of the two files or by selecting them in Workshop 4. Users who want to learn how to create a new ViSi project may proceed to the next section.

Open the Example in Workshop 4

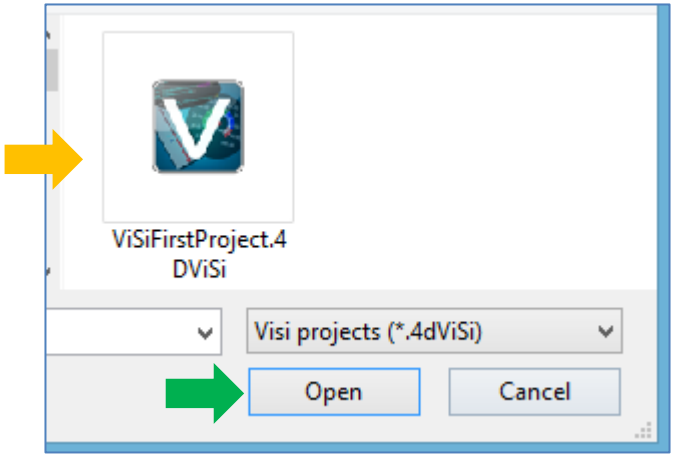
There is a shortcut for Workshop 4 on the desktop.

Launch Workshop 4 by double-clicking on the icon. Workshop 4 opens and displays the Recent page.

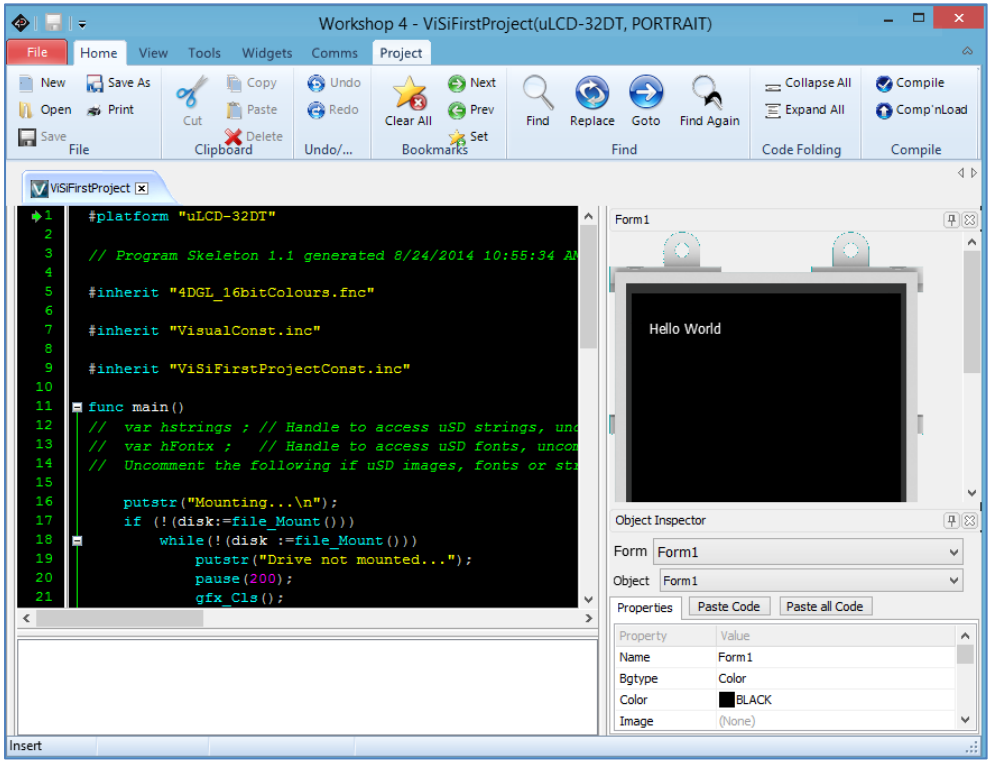
To load the existing project, click on Open.



A standard open window asks for a ViSi project. Select the demo file.

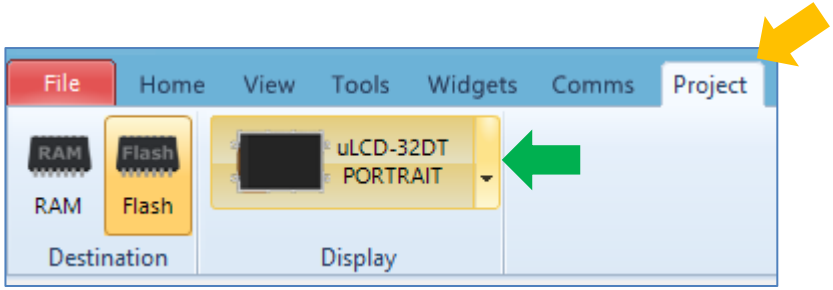


The project opens.

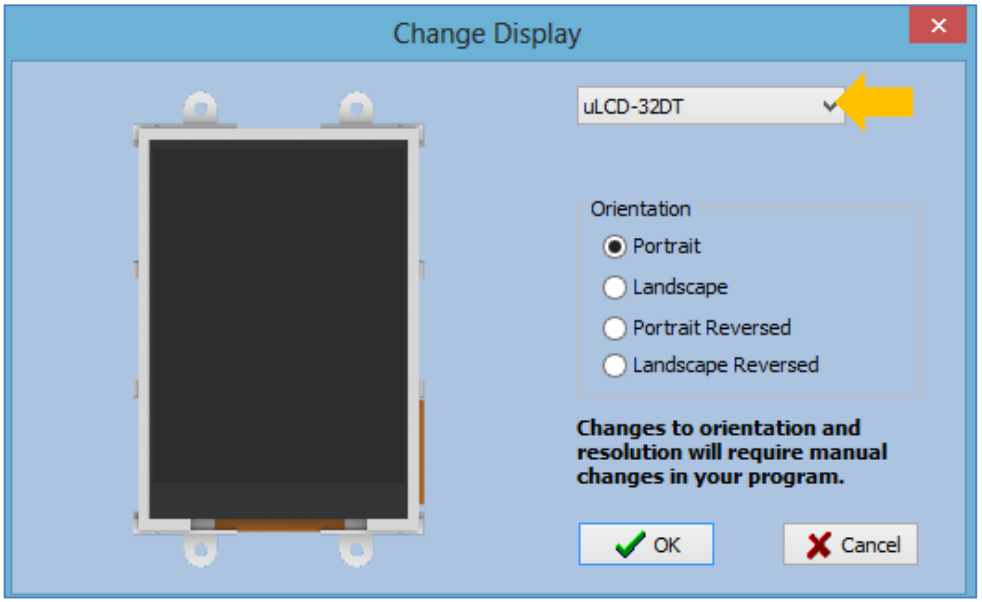


How to Change the Target Display

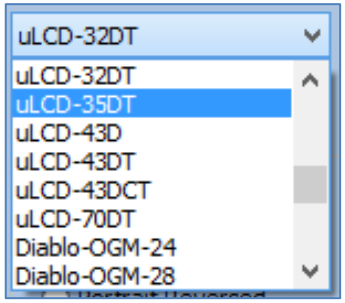
The attached project has its target display configured to be a uLCD-32DT. To change the target device, go to the **Project** menu and click on the display button.



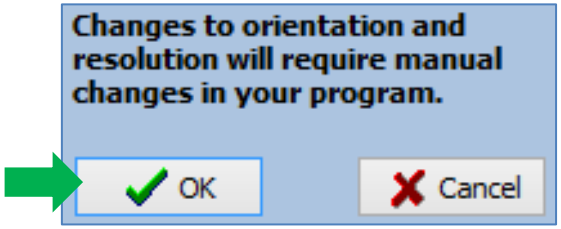
The Change Display window appears. Change the target display by clicking on the display name.



A drop-down menu will appear. Select your new target device.



Click OK to confirm when done.



Changing the orientation is done by manually editing the code. This will be discussed later.

Create a New Project

Launch Workshop 4

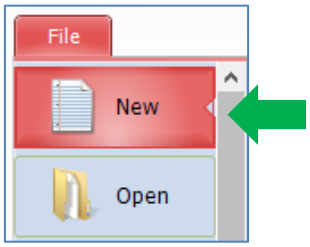
There is a shortcut for Workshop 4 on the desktop. Launch Workshop 4 by double-clicking on the icon.

Create a New Project

Workshop 4 opens and displays the **Recent** page.

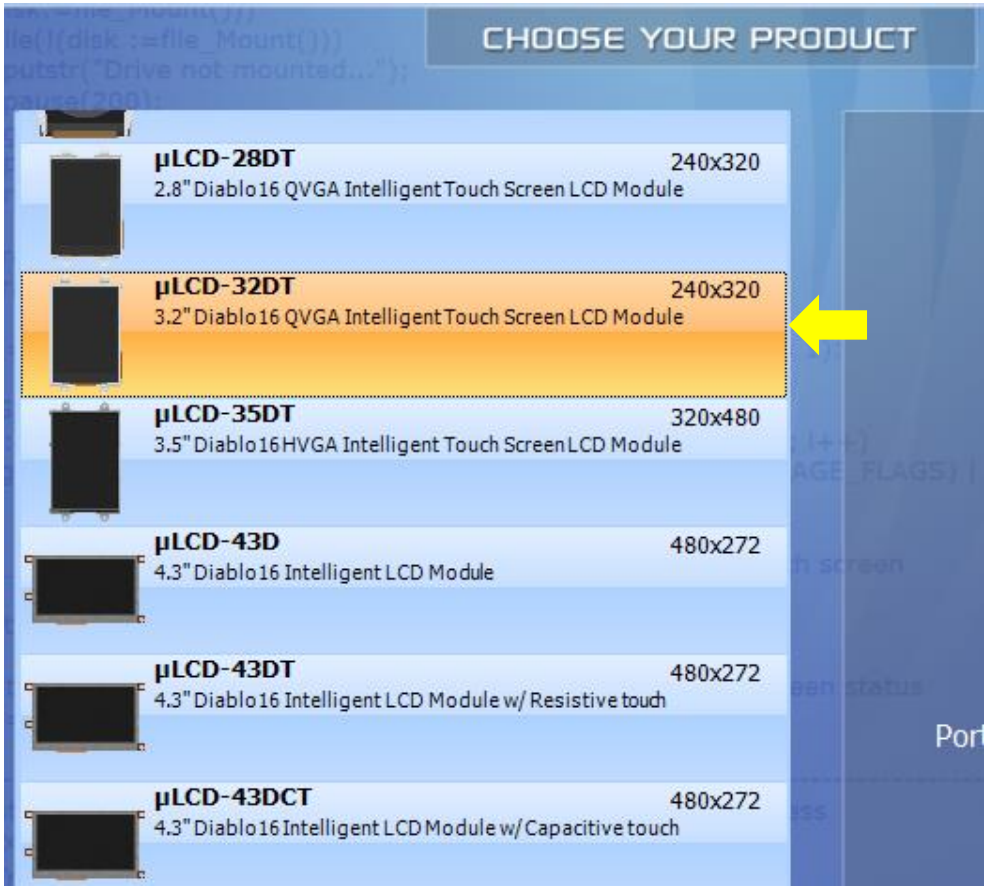
To create a new project, there are two options.

Click on the top left-most icon, New.



Or Click on the icon beside Create a new Project.

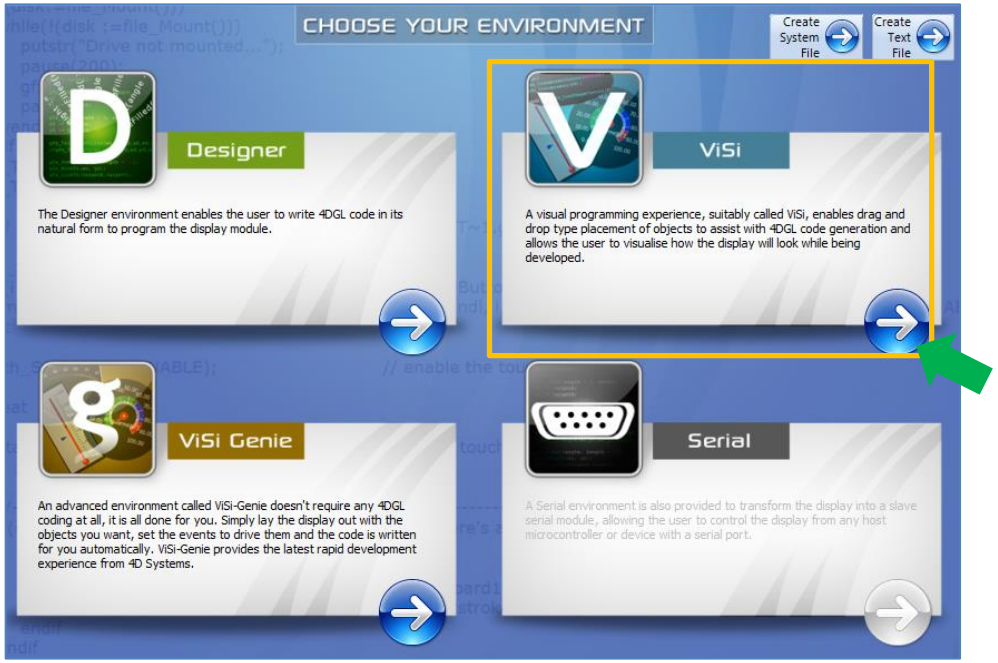
The Choose-Your-Product window appears. Select the appropriate screen and preferred orientation. The screen used in this example is a **uLCD-32DT (portrait orientation)**.



Select the desired orientation by clicking on the display on the right part of the Choose-Your-Product window.

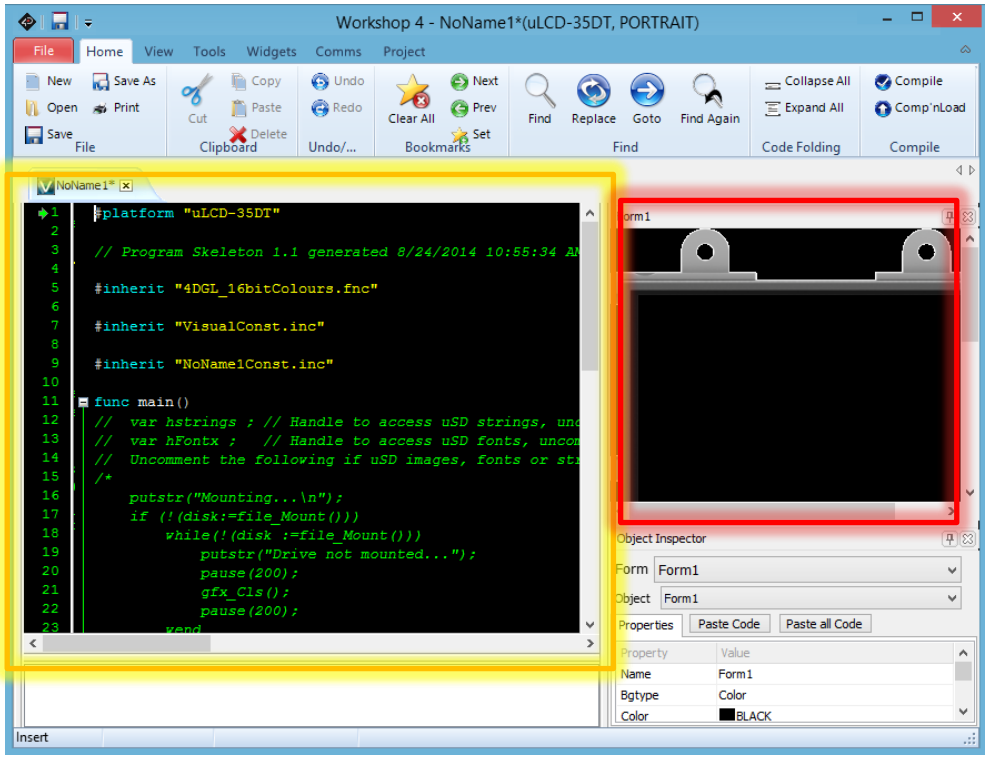
The image of the display rotates as you click it. When done, click on the next button on the lower right part of the Choose-Your-Product window.

Select ViSi



Design the Project

Everything is now ready to start designing the project. Workshop 4 displays the code area (left side – orange box) and the WYSIWYG screen (right side – red box).



Default Code

Workshop 4 automatically provides a default code – a program skeleton to which developers can simply add their codes. The program skeleton contains the basic parts needed to start designing an application.

Define the Target Device

The first line of the default code defines the target device.

```
1   #platform "uLCD-32DT"
```

Include Files

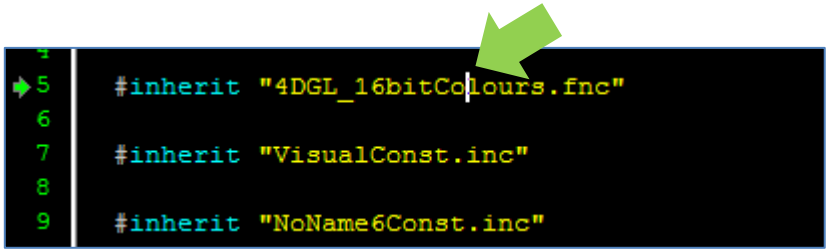
To include a source file in 4DGL, use the pre-processor directive "**#inherit**". The program skeleton has three include files.

```
5   #inherit "4DGL_16bitColours.fnc"
6
7   #inherit "VisualConst.inc"
8
9   #inherit "NoName1Const.inc"
```

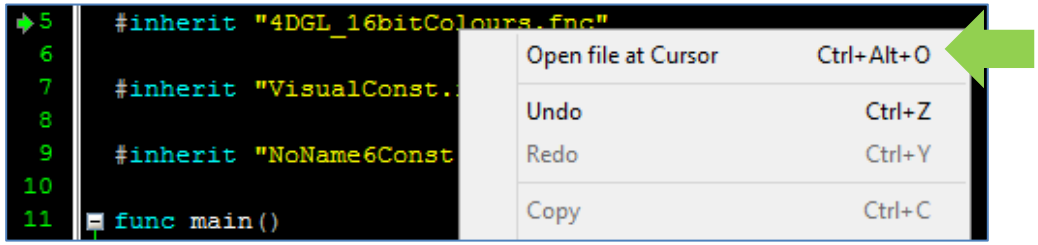
The user can view the contents of an include file by following the instructions below.

How to Open an Include File

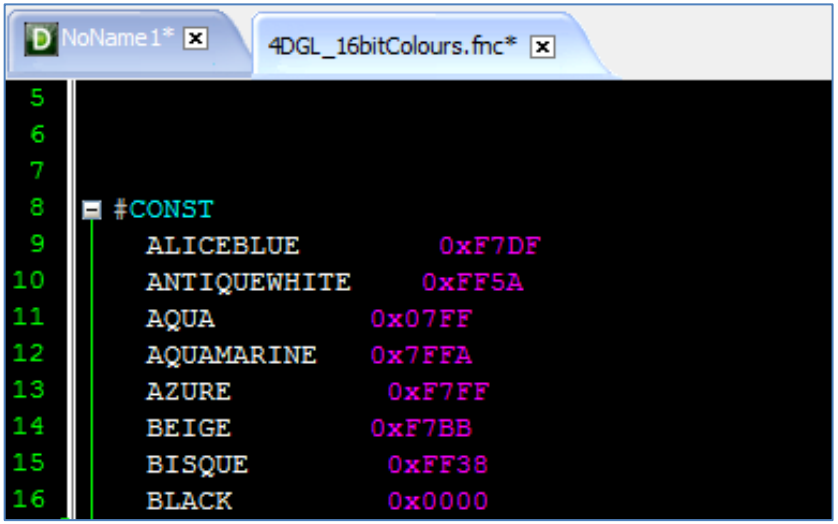
1. Put the cursor on the filename.



2. Click the right mouse button and a menu will appear. Select the first option (**Open file at Cursor**).



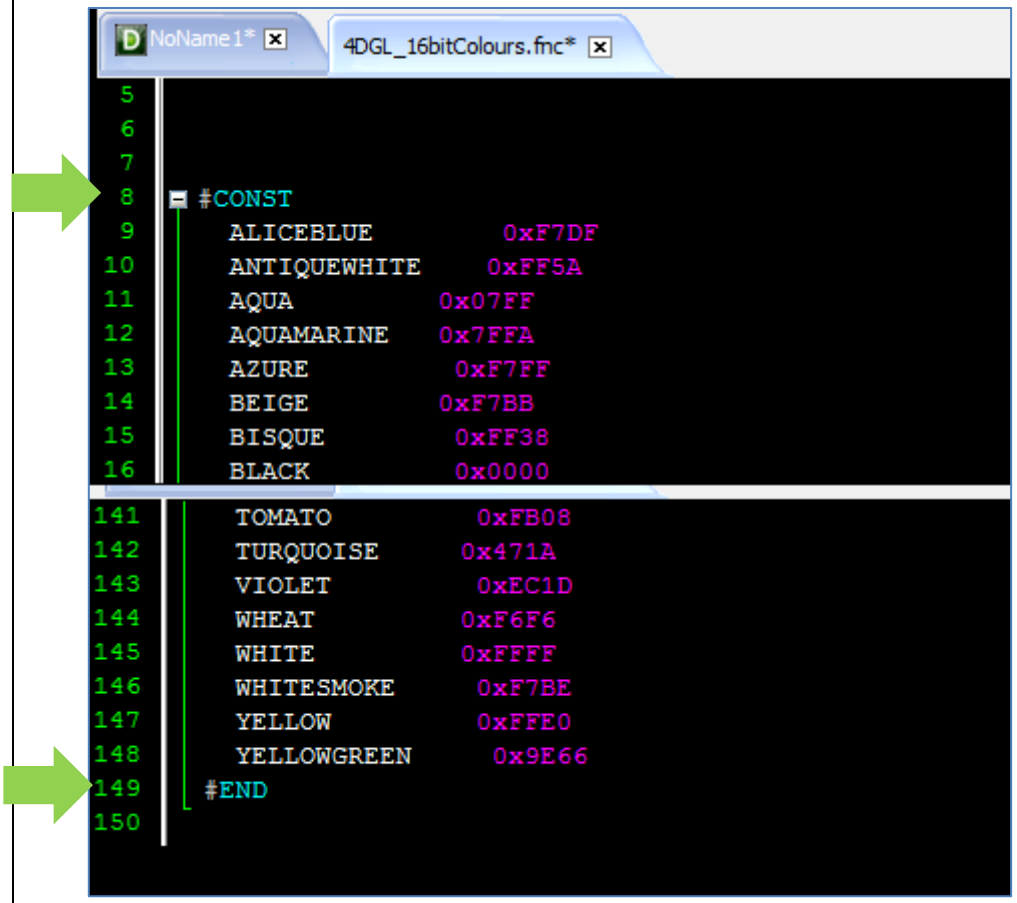
3. Workshop 4 now opens the file 4DGL_16bitColours.fnc.



The file contains **constants** and their values. **Values of constants do not change throughout the duration of the program.**

Constants

The file **4DGL_16bitColours.fnc** contains hexadecimal values of 140 commonly used colour constants. Scroll down to see all the defined colour constants. Take note of lines 8 and 149 below. This is how a block of constants is declared in 4DGL.



The include file "**VisualConst.inc**" contains constants for line patterns. Here each of the constants is defined as a single-line entry.

```
1  // Line Patterns
2  #constant   LPCOARSE     0xF0F0
3  #constant   LPMEDIUM     0x3333
4  #constant   LPFINE       0xAAAA
5  #constant   LPDASHDOT    0x03CF
6  #constant   LPDASHDOTDOT 0x0333
7  #constant   LPSOLID      0x0000
```

The include file **"NoName6Const.inc"** does not exist prior to the compilation of the project. The final name of this include file will be that of the project when it is saved. This include file will contain, among others, constant values of memory locations.

The Main Function

Lines 11 to 36 make up the main function of the program.

Main function

```
11 func main()
12 //  var hstrings ; // Handle to access uSD
13 //  var hFontx ;   // Handle to access uSD
14 //  Uncomment the following if uSD images,
15 /*
16     putstr("Mounting...\n");
17     if (!(disk:=file_Mount()))
18         while(!(disk :=file_Mount()))
19             putstr("Drive not mounted...");
20             pause(200);
21             gfx_Cls();
22             pause(200);
23         wend
24     endif
25 //    gfx_TransparentColour(0x0020);    //
26 //    gfx_Transparency(ON);             //
27
28 //  hFontn := file_LoadImageControl("NoName
29 //  hstrings := file_Open("NoName1.txf", 'r
30     hndl := file_LoadImageControl("NoName1.
31 */
32
33
34     repeat
35     forever
36 endfunc
```

Note that the main function block starts with

```
11  func main()
```

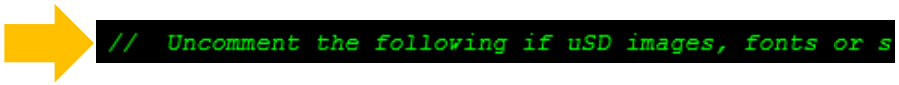
and ends with

```
36  endfunc
```

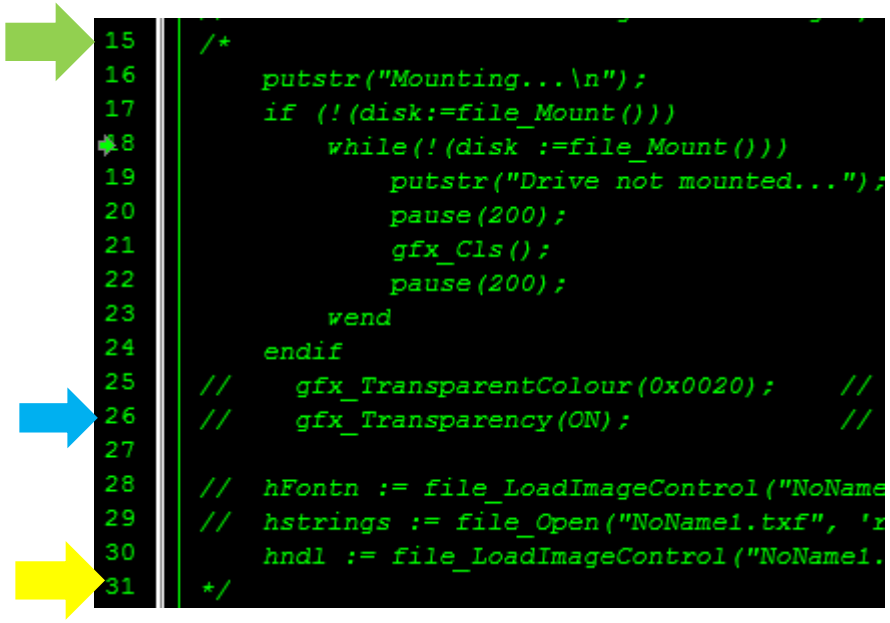
This is how a function is defined in 4DGL.

Comments in 4DGL

Single-line comments in 4DGL look like as shown below.

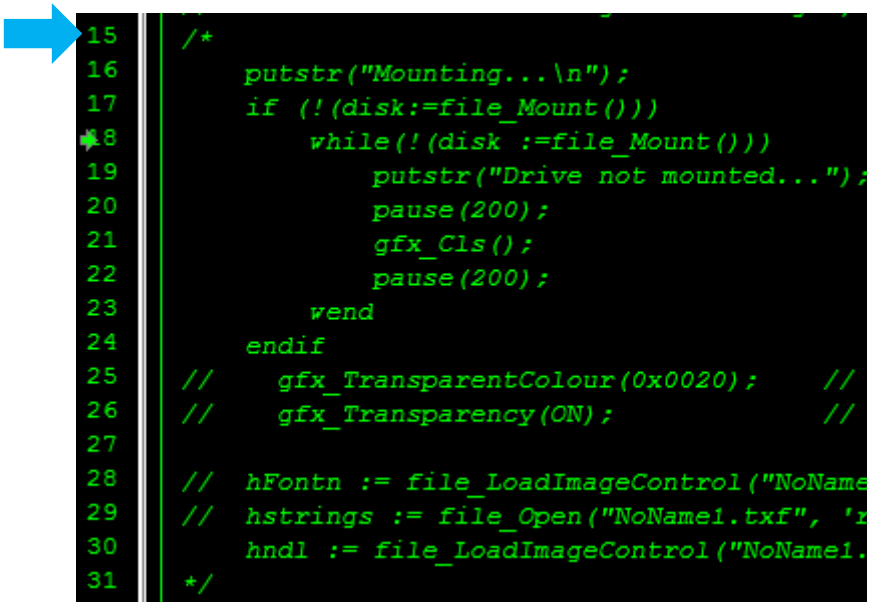


A block comment starts with **"/*"** and ends with **"*/"**, as shown below.



Uncomment the uSD Card Initialization Routine

This application will need a µSD card. Before the uSD card can be accessed, it has to be initialized first. Observe that in the main function, an entire block is commented out. This block (lines 15 to 31) is a uSD-card-initialization routine and it defines the behaviour of the program while waiting for a µSD card to be inserted. Basically the routine flashes the string "Mounting…" while no uSD card is detected and successfully initialized. Remove the block comment symbols indicated below.



The code screen will be updated accordingly, showing the block as an actual part of the code.

```
15
16      putstr("Mounting...\n");
17      if (!(disk:=file_Mount()))
18          while(!(disk :=file_Mount()))
19              putstr("Drive not mounted...");
20              pause(200);
21              gfx_Cls();
22              pause(200);
23          wend
24      endif
25  //    gfx_TransparentColour(0x0020);     // u
26  //    gfx_Transparency(ON);              // u
27
28  //  hFontn := file_LoadImageControl("NoName1
29  //  hstrings := file_Open("NoName1.txf", 'r'
30      hndl := file_LoadImageControl("NoName1.d
31
32
```

Print a String

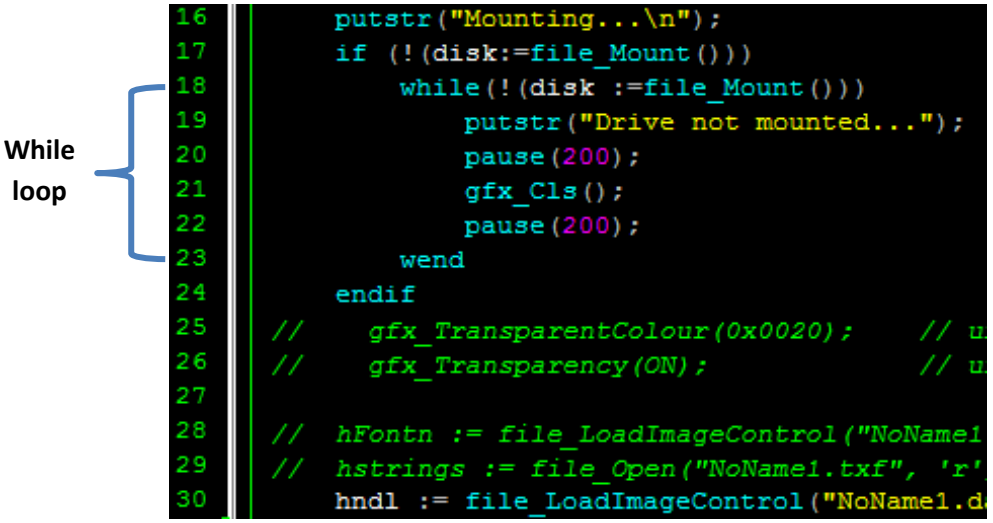
The line

```
16      putstr("Mounting...\n");
```

will print the string “Mounting…” on the screen followed by a new line.

The While Loop

Lines 18 to 23 make up a while loop.



Note that the while loop starts with

```
18          while(!(disk :=file_Mount()))
```

and ends with

```
23          wend
```

A **loop** in a program performs instructions repetitively. The while loop has the basic syntax

```
while(condition)
    [statements]
wend
```

The statements or instructions inside the block are executed as long as the condition is true.

Mount the uSD Card

The function

```
if (!(file_Mount()))
```

starts up the FAT16 disk file services. This function must be called before any other FAT16 file related functions can be used. The function returns a non-zero value if a memory card is present and successfully initialized, and a value of '0' otherwise.

Delay

The function

```
pause(200);
```

delays the execution of the program for 200 milliseconds.

Clear the Screen

To clear the screen, use the function

```
gfx_Cls();
```

The Repeat-forever Loop

The repeat-forever loop is another kind of loop. It starts with

```
34    repeat
```

and ends with

```
35    forever
```

The program will execute the statements between lines 34 and 35 indefinitely. Here the program actually does nothing indefinitely after printing the text "Hello World" since no instructions exist between lines 34 and 35. If this empty loop is omitted, the program exits the main function.

Supporting Graphics Files

In this application note the user will learn how to add a static text widget or object to the WYSIWYG screen. Besides static texts, images, videos, buttons, and many others can also be added as objects to the WYSIWYG screen. When the user clicks on the Compile or Comp'nLoad button, Workshop combines all of these objects into a single graphics file, which is of a format readable by 4D graphics processors. There are actually two files generated – the GCI file and the DAT file. The GCI file contains the actual object images and the DAT file contains a list of the objects in the GCI file. Workshop copies these two files to a FAT16-formatted uSD card mounted on the PC. When the uSD card is unmounted from the PC and mounted to the display module and if the program is downloaded to the display module's processor, the project will now run.

Note that the program is different from the graphics files. The program contains the instructions to be executed by the processor. It resides on and runs from the processor of the display module. The graphics files, on the other hand, are supporting files on the uSD card. These are accessed by the program during runtime. For more information on this important differentiation, refer to the application note ViSi-Genie Program Destination.

Accessing the Graphics Files

With the uSD card properly mounted by the uSD card initialization routine, the program can now access the GCI and DAT files using the function

```
30      hndl := file_LoadImageControl
                ("NoName1.dat", "NoName1.gci", 1);
```

This function returns a handle (a pointer to the memory allocation) to the image control list that has been created. In other words, we can use the variable "hndl" if we want to access the contents of the graphics files, e.g. if we want to display a certain object. To learn more about the **file_LoadImageControl(…)** function, consult the Picaso Internal Functions Reference Manual or the Diablo16 internal Functions Reference Manual.

Note: Workshop will automatically derive the filenames of the GCI and DAT files from the project name. The filenames will be reflected in the code and in the names of the actual GCI and DAT files. When the project is saved with the name "**ViSiFirstProject**", line 30 of the code will be updated as shown below.

```
30      hndl := file_LoadImageControl
                ("VISIFIRS.dat", "VISIFIRS.gci", 1);
```

These will also be the names of the actual files generated inside the project folder and those copied to the uSD card. Note that the filenames follow the 8.3 format.

It is always a good practice to check if the filenames hardcoded into the code correspond to the actual names of the files on the uSD card. Although Workshop automatically does this in the background for you, other factors such as the settings of your PC may sometimes prevent Workshop from "synchronizing" the filenames.

Hello World

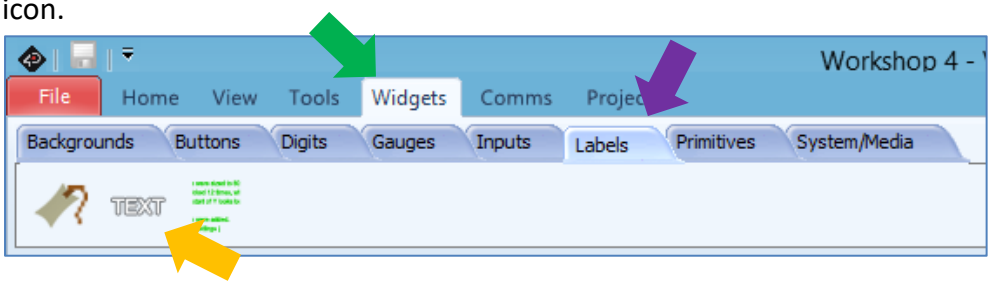
To print “Hello World” on the screen, write

```
25  //    gfx_TransparentColour(0x0020);
26  //    gfx_Transparency(ON);
27
28  //  hFontn := file_LoadImageControl('
29  //  hstrings := file_Open("NoName4.tx
30      hndl := file_LoadImageControl("Nc
31
32      print("Hello World!");
33
34      repeat
35      forever
```

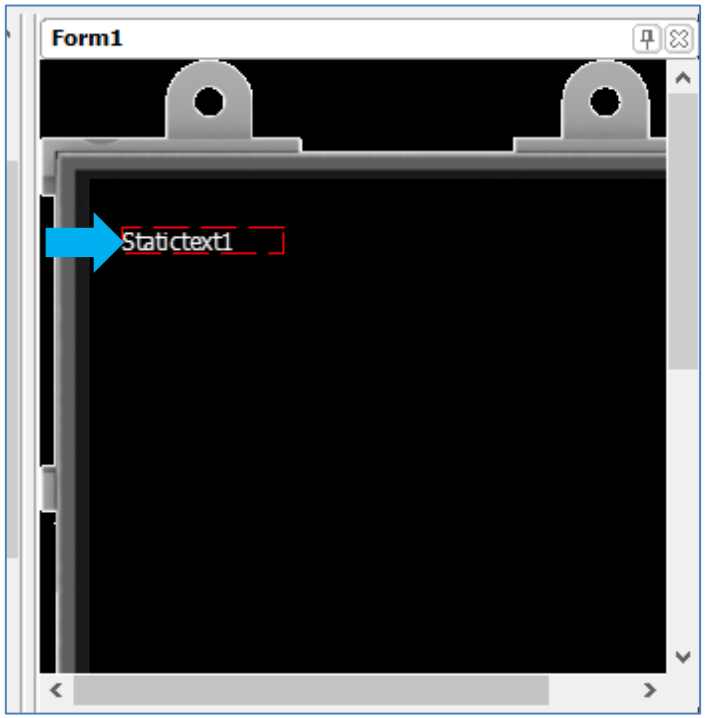
The function **print()** is another function for printing a string.

Add a Static Text Object

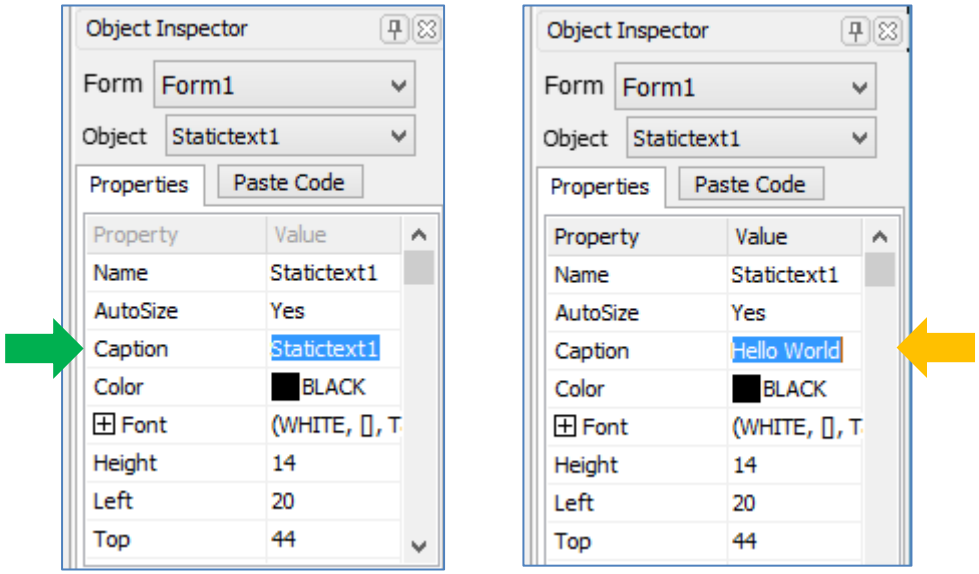
Go to the Widgets menu, select the Labels pane, and click on the static text icon.



Once the static text icon is selected, click on the WYSIWYG screen to place it.



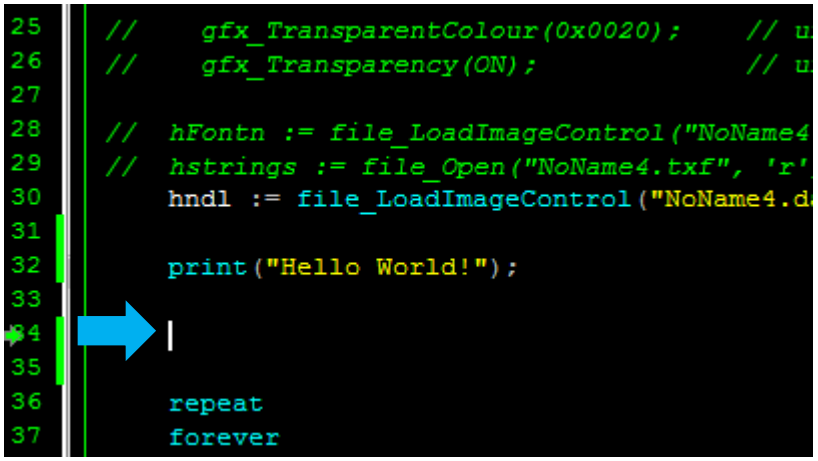
The Object Inspector shows the different properties of the object. The object has the name “Statictext1”, which means that it is the first static text object added to the project. By default, an object has the same name and caption. Change the value of the property “Caption” from “Statictext1” to “Hello World”.



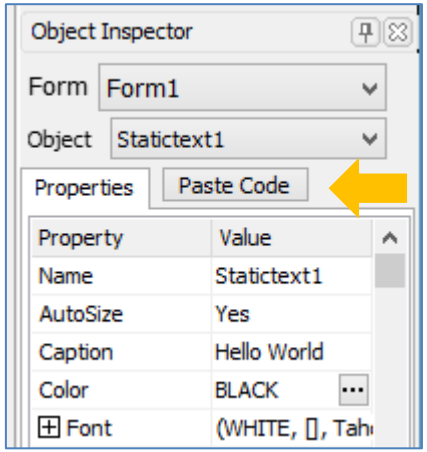
The WYSIWYG screen is updated accordingly.

Paste the Code for a Static Text Object

In the code area, place the cursor somewhere between the "print("Hello World!");" statement and the repeat-forever loop.



With Statictext1 still selected, click on the "Paste Code" button of the Object Inspector.



A new line is added to the code. This is the command for showing the object Statictext1 on the screen.

```
32      print("Hello World!");
33
34
35      // Statictext1 1.0 generated 8/24/2014 5
36      img_Show(hndl,iStatictext1) ;
37
38
39      repeat
40      forever
41  endfunc
42
```

After executing the uSD-card-initialization routine, the program will now print the string “Hello World” and will display the object Statictext1.

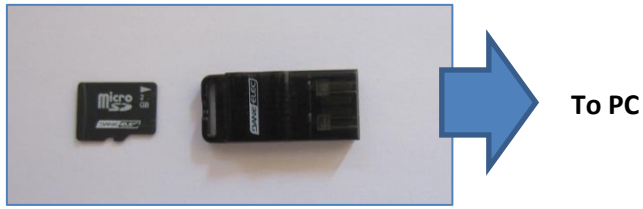
Run the Program

Save the Project

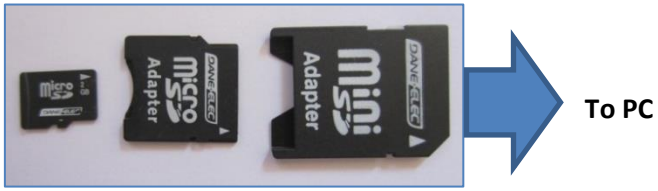
Save the program with the desired file name first.

Insert the uSD Card to the PC

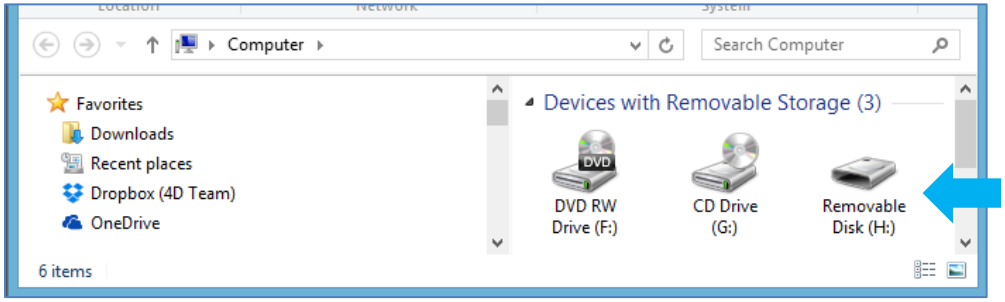
Insert the µSD card into the USB adaptor and plug the USB adaptor into a USB port of the PC.



OR insert the µSD card into a µSD-to-SD card converter and plug the SD card converter into the SD card slot of the PC.

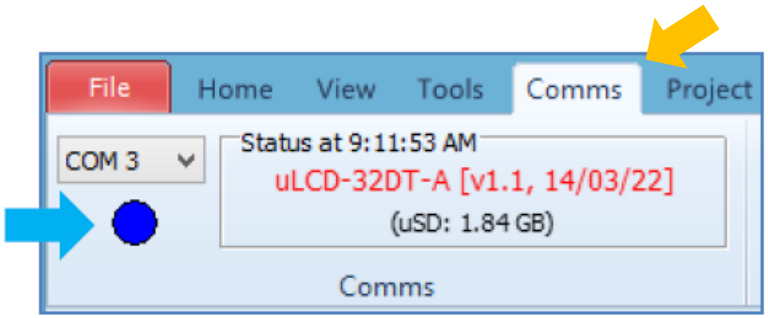


Check if the µSD card is mounted. In the image below, it is mounted as drive H:

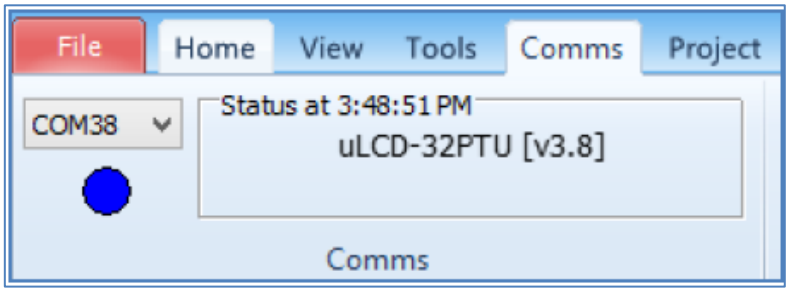


Connect the Display Module to the PC

Connect the display module to the PC using a 4D Programming Cable or a uUSB-PA5 adapter. Go to the Comms menu to check if the module is detected. The button should be blue in colour. Below is an example of how the Comms tab will look like if a uLCD-32DT is connected to the PC. Note that Workshop detects that the PmmC and driver of the connected display module are outdated.



Below is an example of how the Comms tab will look like if a uLCD-32PTU is connected to the PC.

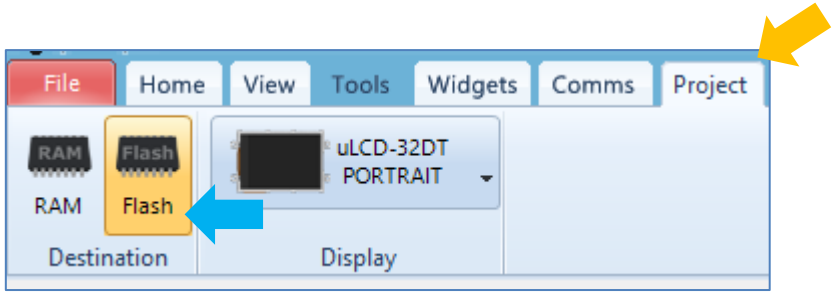


If not familiar with how to actually connect a 4D display to the PC using a 4D USB programming cable or a uUSB-PA5 and with how to update the firmware, the user should consult the application notes below.

General How to Update the PmmC for Picaso
General How to Update the PmmC for Diablo16

Choose the Program Destination

Under the **Project** menu, choose **Flash** as the destination.

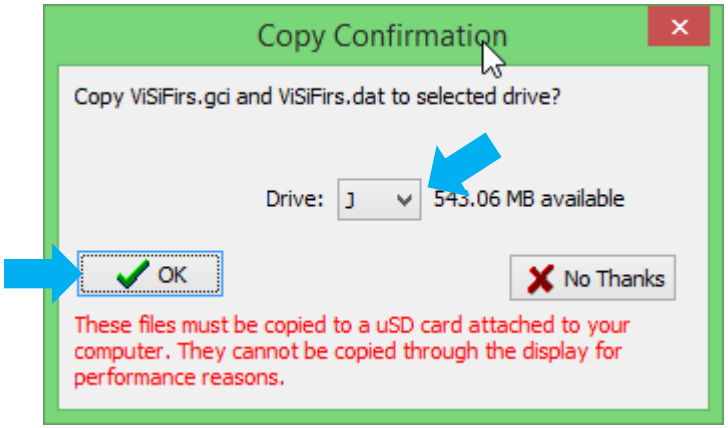


To know more about the destination options “RAM” and “Flash”, read the application note General Downloading an Application Program to RAM or Flash Memory.

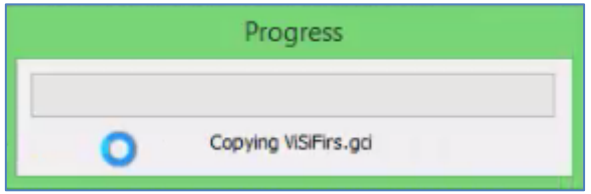
Compile and Download

After making sure that the device is detected, go to the Home menu and click on the **Comp n’Load** (Compile and Download) button.

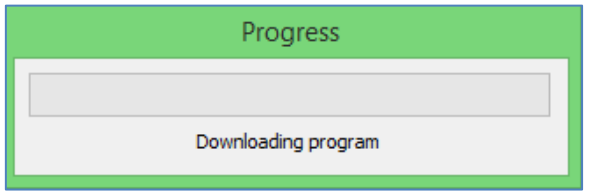
Workshop now builds the graphics files and copies them to the uSD card. The Copy Confirmation window appears. The user will be prompted to choose the correct drive for the memory card. Now choose the correct drive by clicking on the drop down arrow. Then click OK.



Workshop copies the supporting graphics files to the µSD card.



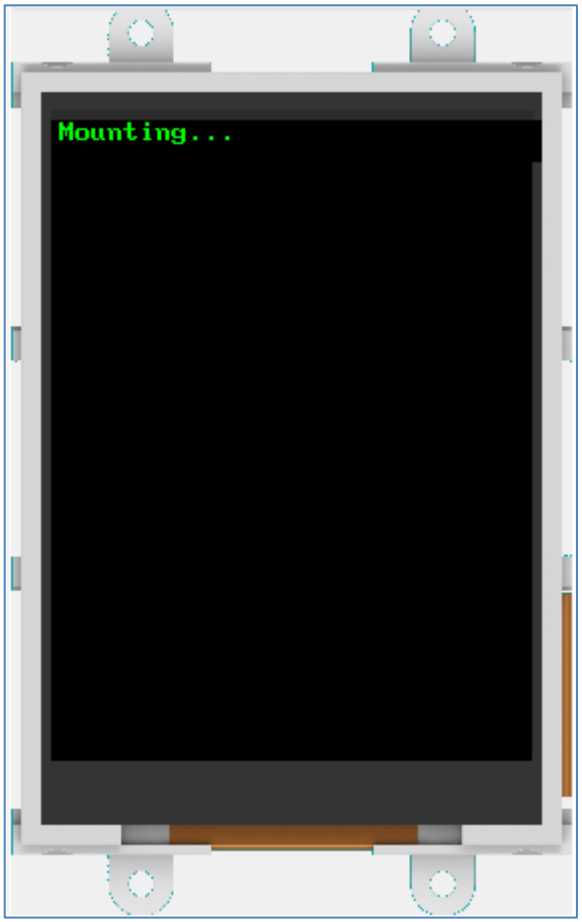
Workshop then downloads the program to the display module’s processor.



The message box will look like as shown below after a successful download.

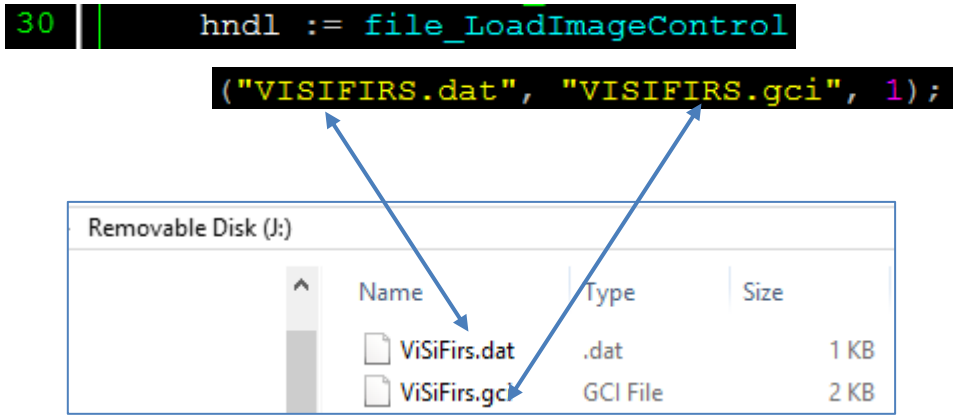
```
0 errors
0 warnings
0 notices
No Errors, code size = 189 bytes out of 14400
total
Initial RAM size = 202 bytes out of 14400 total
Program will run from ram so total initial RAM
size = 391 bytes out of 14400 total
Download to Flash successful.
```

The program should now run on the processor of the display module and it should be waiting for a uSD card.



Check the Contents of the uSD Card

In the section "**Accessing the uSD Card**", it was emphasized that Workshop automatically derives the filenames of the GCI and DAT files from the project name. The filenames follow the 8.3 format and are used in the code and as the names of the actual GCI and DAT files. Before unmounting the uSD card from the PC, ensure that the files are present and that the filenames are correct.

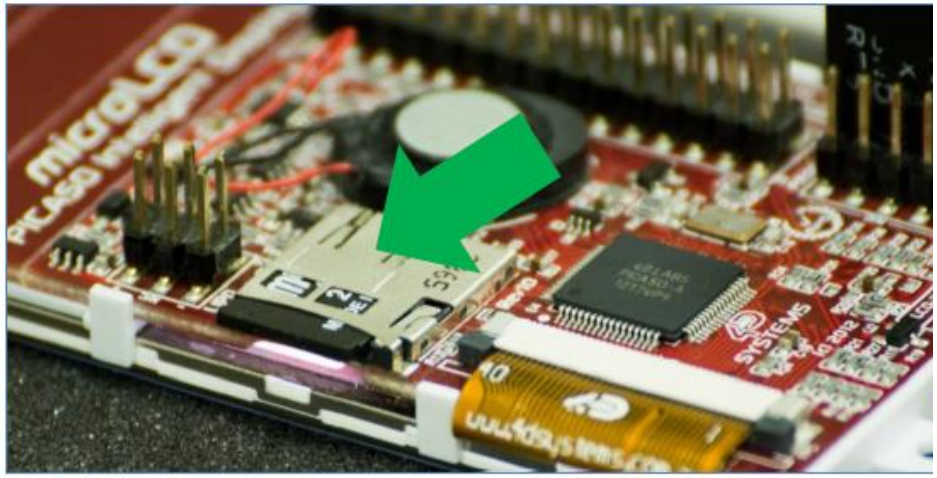


Again, it is always a good practice to check if the filenames hardcoded into the code correspond to the actual names of the files on the uSD card. Although Workshop automatically does this in the background for you, other factors such as the settings of your PC may sometimes prevent Workshop from "synchronizing" the filenames. There are other supporting files besides the graphics files. Checking the names of these files both in the code and with the actual files may sometimes be necessary for troubleshooting purposes. For more information on the different types of

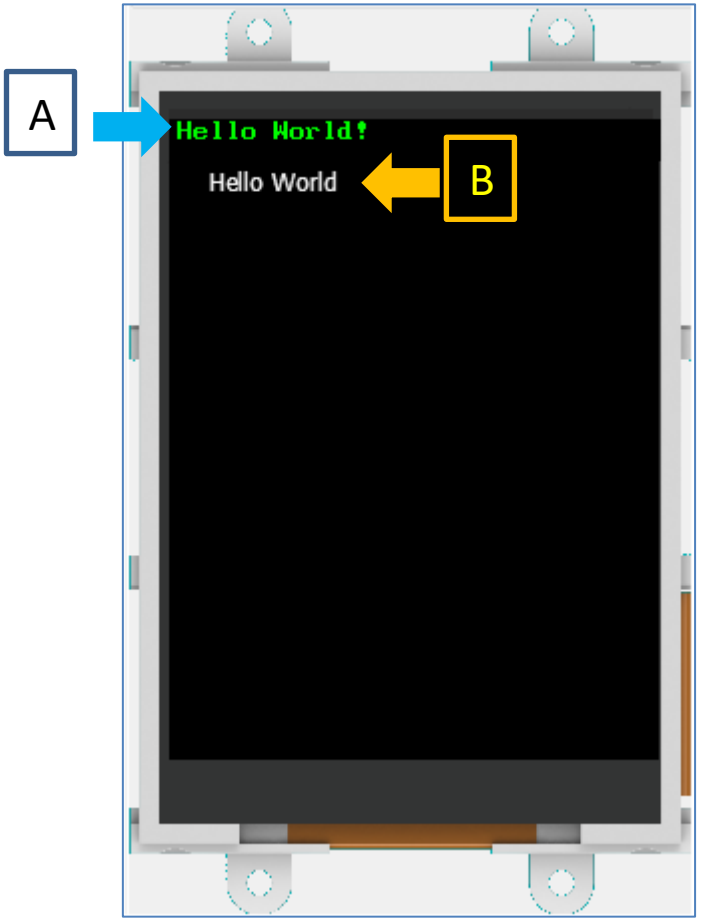
supporting files, refer to the application note ViSi-Genie Program Destination.

Mount the uSD Card to the Display Module

Properly remove the µSD card from the PC and insert it into the µSD card slot of the display module.



The program prints the string “Hello World!” and displays the object Statictext1.



Item A is a string printed as a result of the instruction,

```
32        print("Hello World!");
```

The string “Hello World” in this case is a literal constant inside the program.

Item B is an image retrieved from the uSD card and is displayed as a result of the instructions

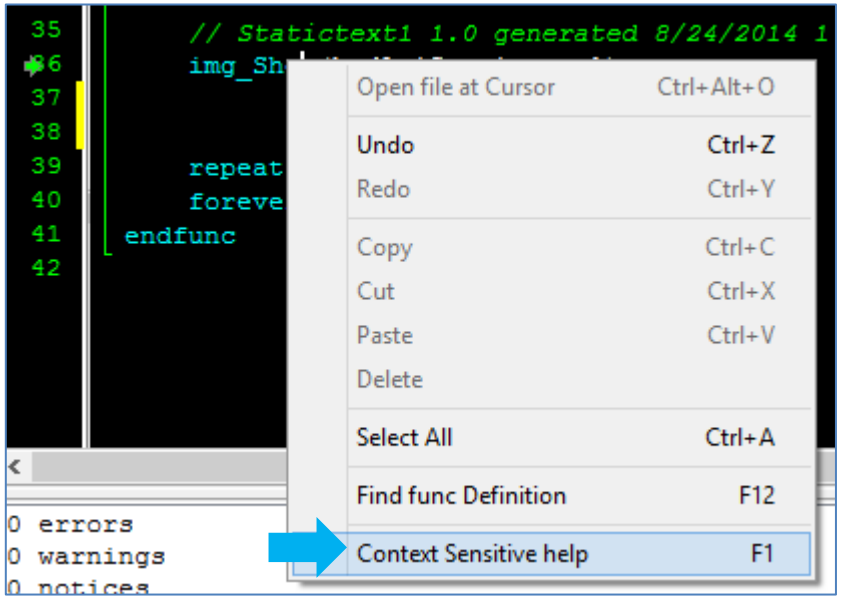
```
30        hndl := file_LoadImageControl
              ("VISIFIRS.dat", "VISIFIRS.gci", 1);
```

and

```
// Statictext1 1.0 generated 8/24/2014 1:59:11 PM
img_Show(hndl,iStatictext1) ;
```

You may consult the internal functions reference manual of your display module's processor for more information. The manual can be opened from Workshop. To do so, put the cursor on the desired function name, click on the right mouse button, then choose "Context Sensitive help".

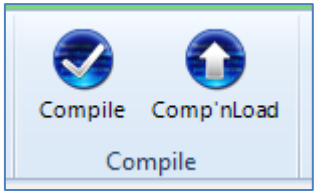
```
// Statictext1 1.0 generated 8/24/2014 1:59:11 PM
img_Show(hndl,iStatictext1) ;
```



Alternatively, you can also open the manual by pressing F1 while the cursor is on the function name.

Updating the Contents of the uSD Card

One important thing to remember is that Workshop generates or builds the graphics for all the objects in the project during design time, when you click the Compile or Comp'nLoad button.



As previously discussed, all graphics are integrated into the supporting files, which are eventually copied to the uSD card. Thus, if you want to remove, change, or add an object from/in/to the project, you will have to update the contents of the uSD card. This means that you will have to unmount the uSD card from the display module and mount it back to the PC, so that Workshop can copy the newly generated files.

Detected Changes Made with the Objects

Workshop detects changes made with the objects of the project by monitoring the WYSIWYG area and the Object Inspector. For example, if you changed the label of a static text object or dragged a button object to a new location, Workshop will, after you click on the Compile, Compn'Load, or Download button, automatically generate a new set of supporting graphics files to be copied to the uSD card. As previously mentioned, again you will have to unmount the uSD card from the display module to update its contents.

However, if you haven't touched the WYSIWYG area or modified any field in the Object Inspector, Workshop will not generate a new set of support files since it is not necessary. This behaviour is important because you wouldn't want Workshop to be generating support files all the time, most especially when the project is large (Workshop may take a while to generate the graphics files of a very large project).

Detected Changes Made with the Code

Changes made with the 4DGL code only will cause Workshop, after you click on the Compile button, to compile the program. You will then have to click on the Download button to make Workshop download the program to the processor. Alternatively, you can click on the Comp'nLoad button to make Workshop compile the code and download the program to the processor of the display module. Also, you will notice that Workshop displays the Compn'Load button or the Download button, depending on whether it has detected changes made with the code and/or objects, or not.

In summary, remember that, if you have made changes with the 4DGL code only, Worskhop will display the Compile button and the Compn'Load button. Clicking on the Compile button will cause Workshop to compile the code. To make Workshop compile the code and download the program to the processor of the display module, click on the Compn'Load button.

If no changes have been made with the code and/or objects recently, Workshop will display the Compile button and the Download button. Clicking on the Compile button will cause Workshop to compile the code. Clicking on the Download button will cause Workshop to download the program to the display module processor.

If Worskhop has detected changes made with the objects, it will always generate a new set of supporting files and will prompt you for the correct uSD card drive to which the supporting files will be copied.

Changing the Properties of Objects Dynamically

It is not possible to change the properties of an object during runtime. For instance, changing the caption of a static text object or a button is not

possible when the program is running. Again, the graphics for the objects are generated or pre-rendered during design time.

Transfer of Supporting Files

Since supporting files can become significantly very large in size, Workshop always copies them directly to the uSD card, i.e. the uSD card has to be mounted to the PC. Programs, on the other hand, are relatively small in size. Hence, they can be downloaded to the display module processor through the programming cable. Note that the transfer rate when using the programming cable is very limited (in the range of several KB per second) compared to that when the uSD card is mounted to the PC (several MB per second).

Although impractical for large supporting files, there is a way to transfer small supporting files to the uSD card mounted on the display module through the programming cable. For more information, refer to the application note General Serial File Transfer from PC to Display Module uSD Card.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability