



ViSi Getting Started – First Project for Pixxi Display Modules

DOCUMENT DATE: 19th MAY 2020
DOCUMENT REVISION: 1.0

Description

This application note provides a first hands-on example with ViSi and describes all the steps related to a project.

Before getting started, the following are required:

Hardware

- Any 4D Systems display module powered by any of the following processors:
 - o Pixxi28/44
- Programming Adaptor for target display module
- uSD Card
- USB Card Reader

Software

- Workshop4

This application note comes with one (1) ViSi project:

- ViSiFirstProject.4DVisi

Note: Using a non-4D programming interface could damage the processor and void the warranty.

Content

Description2

Content.....2

Application Overview3

Setup Procedure4

Open a Project from the File Explorer4

Open the Example in Workshop4.....4

How to Change the Target Display.....5

Create a New Project.....6

Launch Workshop4.....6

Create a New Project6

Select ViSi8

Design the Project9

Default Code.....9

Define the Target Device9

Include Files9

 How to Open an Include File 9

Constants10

The Main Function.....11

 Comments in 4DGL 12

 The File System 12

 The Repeat-forever Loop 14

Supporting Graphics Files15

micro SD card	15
On-board Internal Flash Memory	16
<i>Hello World</i>	16
<i>Add a Static Text Object</i>	17
Paste the Code for a Static Text Object	18
Run the Program	19
<i>Save the Project</i>	19
<i>Insert the uSD Card to the PC</i>	19
<i>Connect the Display Module to the PC</i>	20
<i>Choose the Program Destination</i>	20
<i>Compile and Download</i>	21
micro SD Card	21
On-board Internal Flash Memory	23
<i>Running the Program</i>	23
<i>Updating the Contents of the File System</i>	24
Detected Changes Made with the Objects	25
Detected Changes Made with the Code	25
Changing the Properties of Objects Dynamically	26
Proprietary Information	27
Disclaimer of Warranties & Limitation of Liability	27

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 pixxiLCD-24P4CT module, which is, as of writing, the latest intelligent display module released by 4D Systems. The procedures described, however, are applicable to all Pixxi display modules.

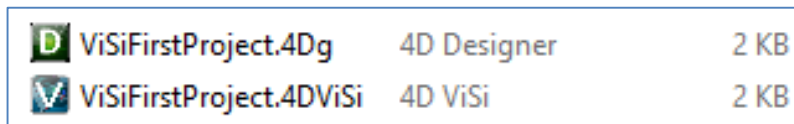
Setup Procedure

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 Workshop4. Users who want to learn how to create a new ViSi project may proceed to the next section.

Open the Example in Workshop4

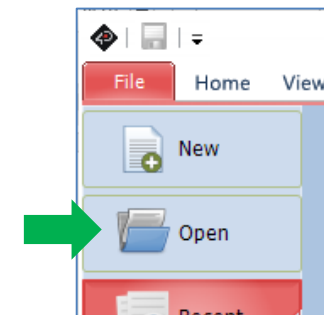
There is a shortcut for Workshop4 on the desktop.



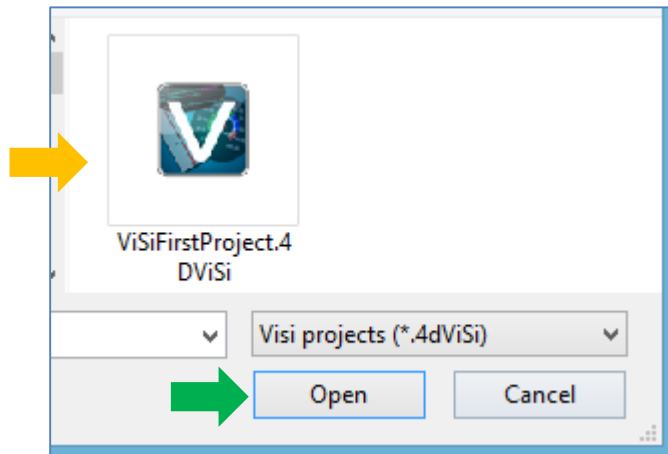
Launch Workshop4 by double-clicking on the icon. Workshop4 opens and displays the Recent page.



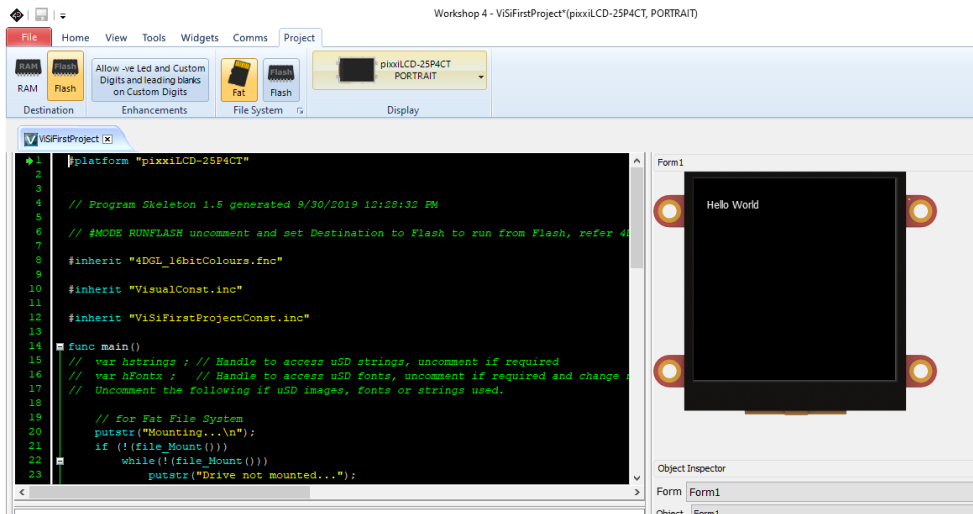
To load the existing project, click on Open.



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

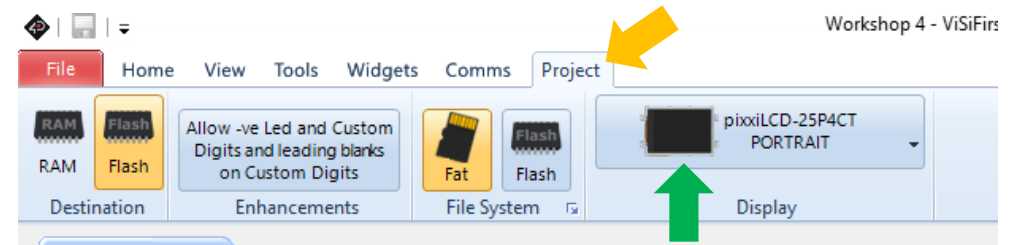


The project opens.

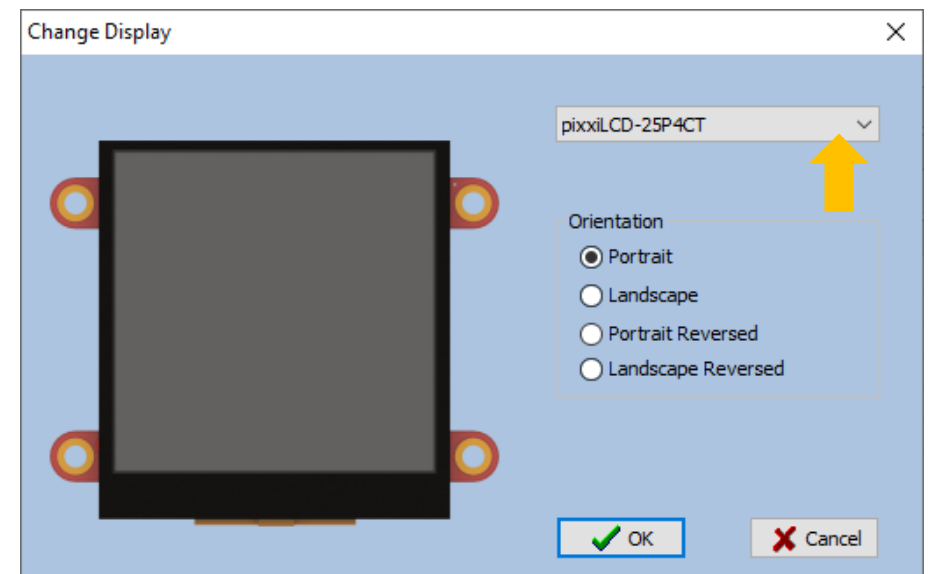


How to Change the Target Display

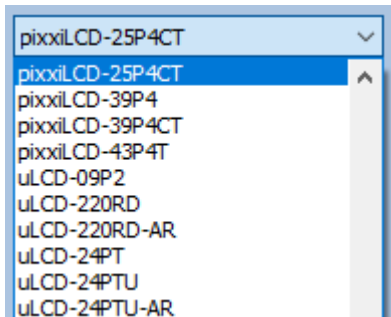
The attached project has its target display configured to be a pixxiLCD-24P4CT. To change the target device, go to the **Project** menu and click on the display button.



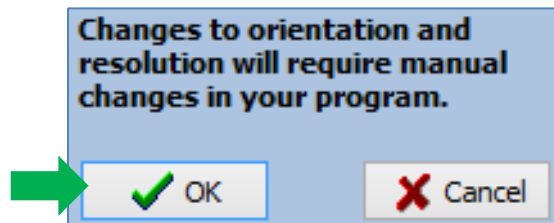
The Change Display window appears. Change the target display by clicking on the display name.



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 Workshop4

There is a shortcut for Workshop4 on the desktop. Launch Workshop4 by double-clicking on the icon.



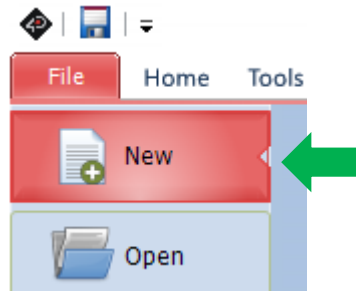
Create a New Project

Workshop4 opens and displays the **Recent** page.



To create a new project, there are two options.

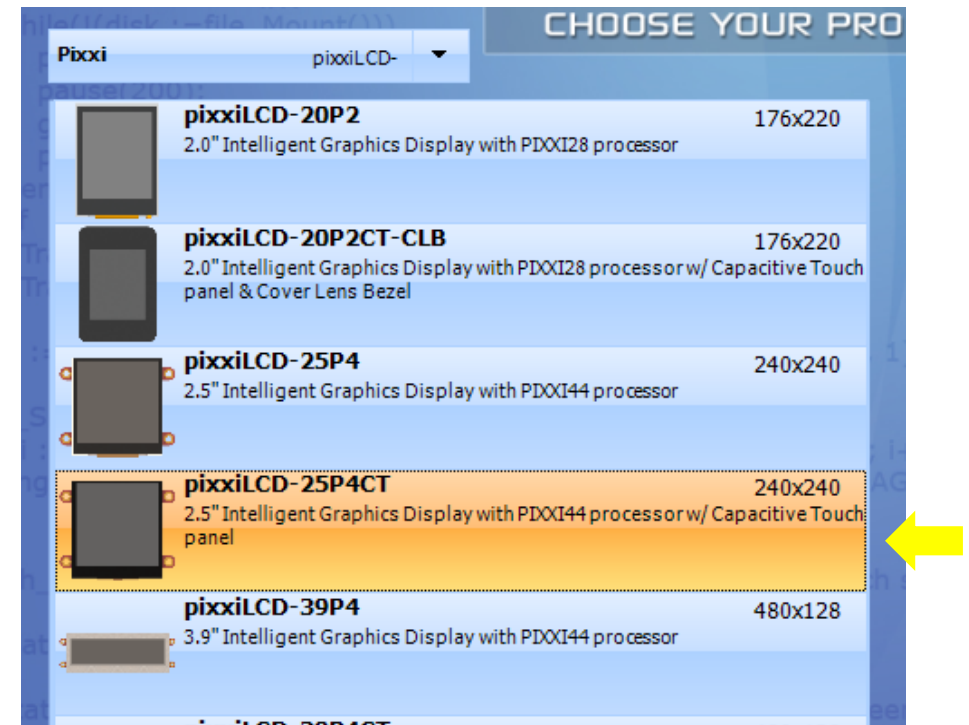
Click on the top left-most icon, New.



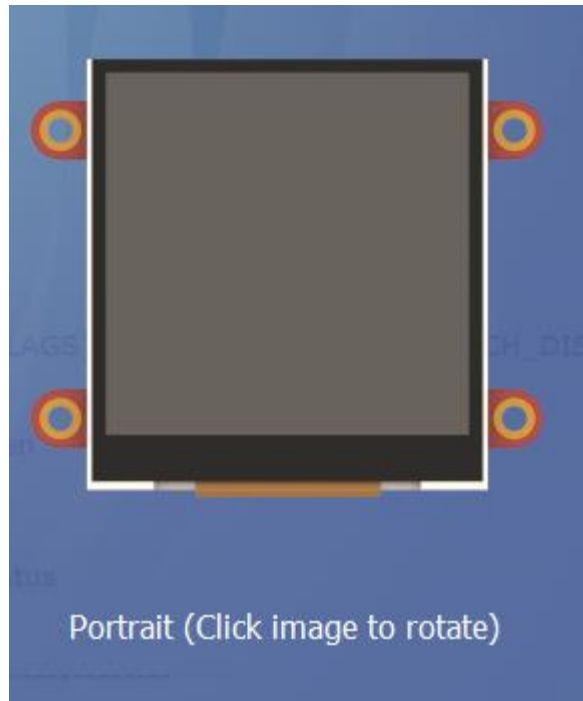
Or Click on the icon beside Create a new Project.



The Choose-Your-Product window appears. Select the appropriate screen and preferred orientation. The screen used in this example is **pixxiLCD-24P4CT (portrait orientation)**.



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

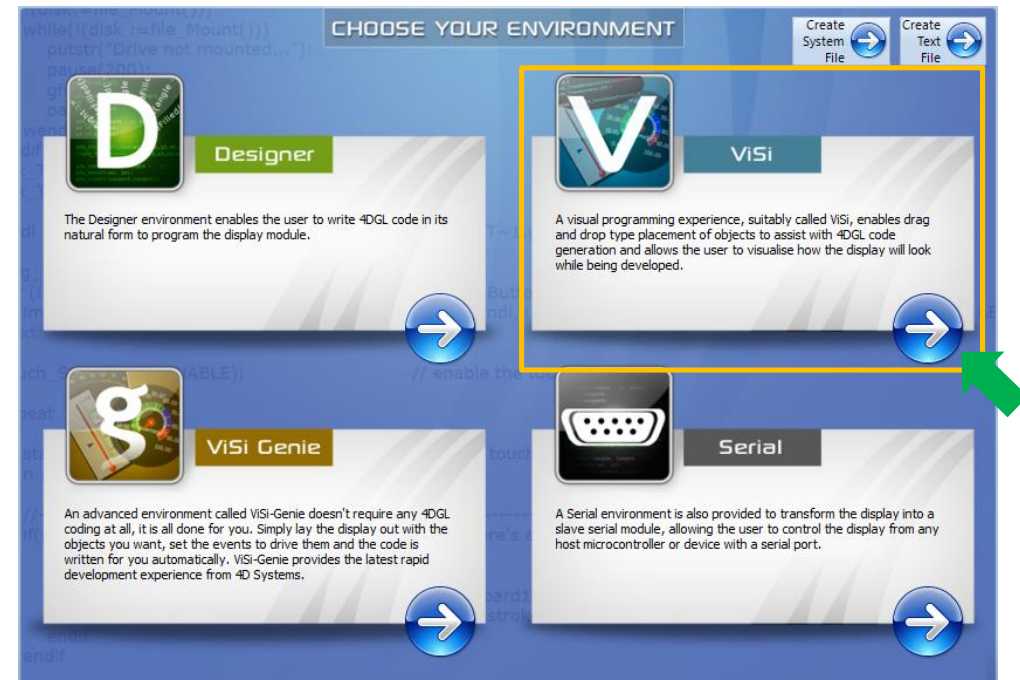


The image of the display rotates as you click it. When done, click on the next button on the lower right part of the Choose-Your-Product window.



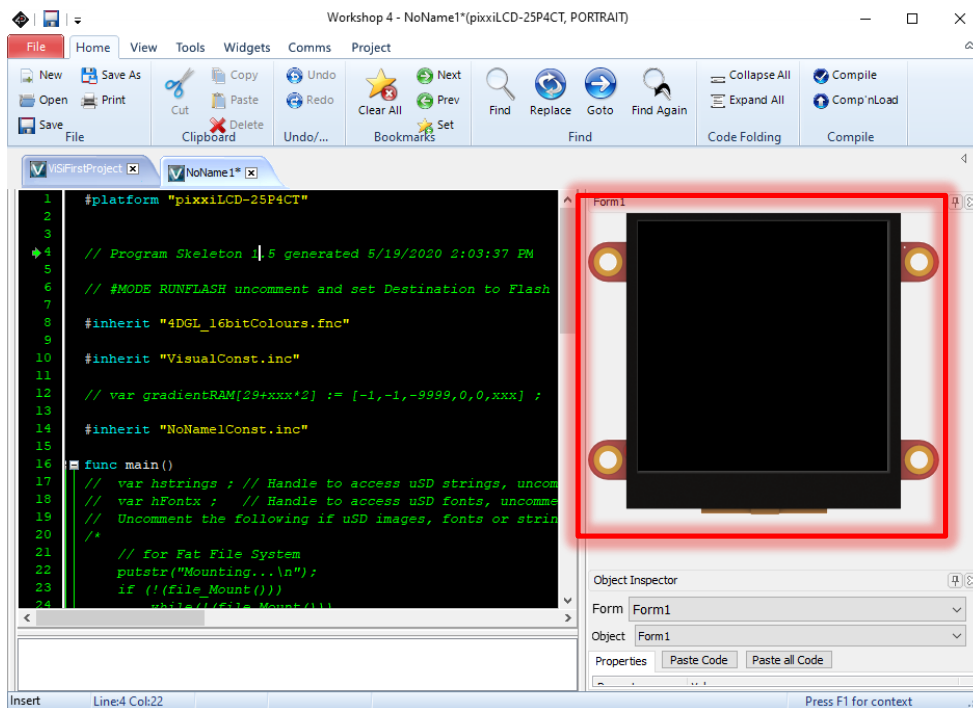
Select ViSi

To start a project under the ViSi environment click the proceed button as shown below.



Design the Project

Everything is now ready to start designing the project. Workshop4 displays the code area (left side – orange box) and the WYSIWYG screen (right side – red box).



Default Code

Workshop4 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 "pixxiLCD-25P4CT"
```

Include Files

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

```
8 #inherit "4DGL_16bitColours.fnc"
9
10 #inherit "VisualConst.inc"
11
12 #inherit "NoName1Const.inc"
```

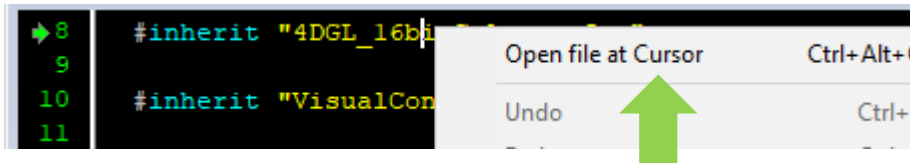
The user can view the contents of an include file by following the instructions as follows.

How to Open an Include File

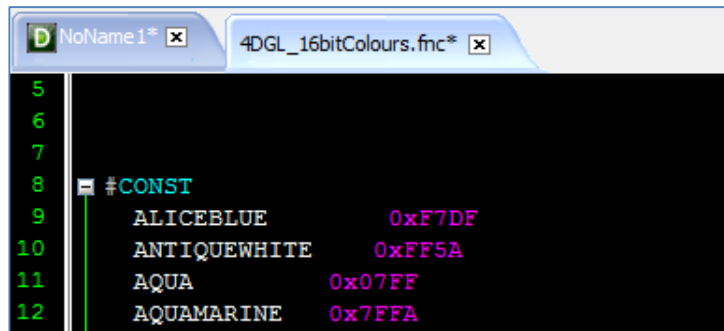
1. Put the cursor on the filename.

```
8 #inherit "4DGL_16bitColours.fnc"
9
10 #inherit "VisualConst.inc"
11
12 // var gradientRAM[29+xxx*2] := [-1,-1,-1]
13
14 #inherit "NoName1Const.inc"
```

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



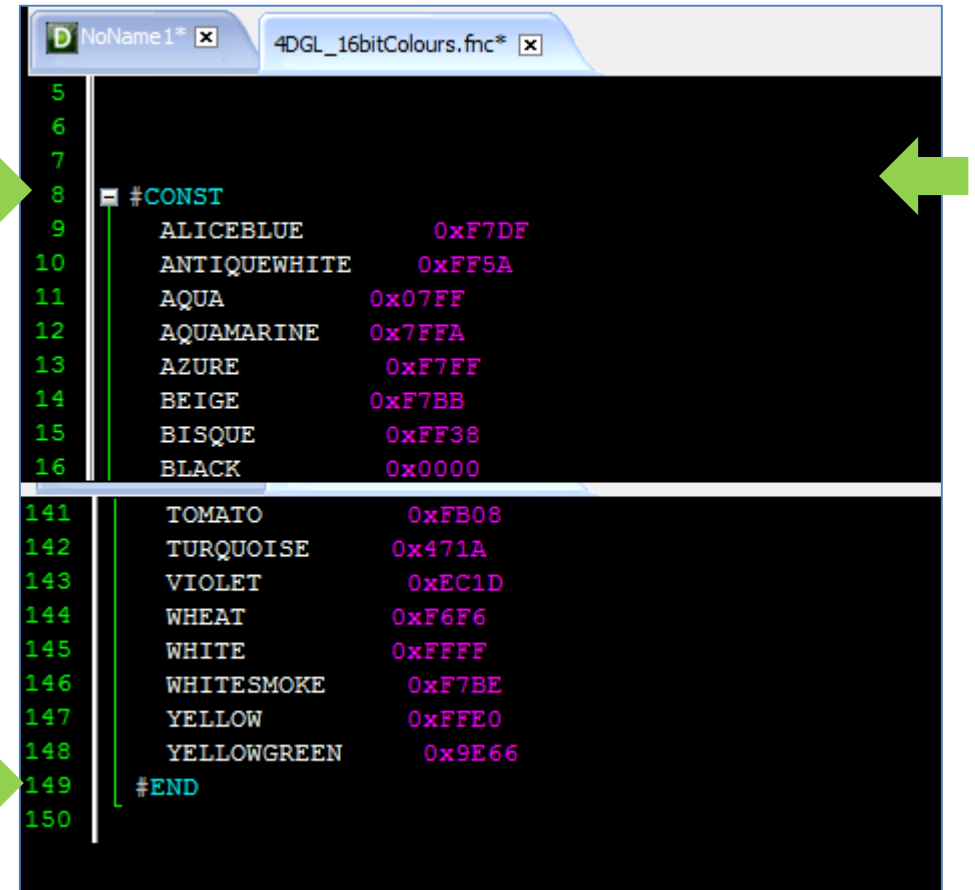
- Workshop4 now opens the file 4DGL_16bitColours.fnc.



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 14 to 45 make up the main function of the program.

```

16 func main()
17 // var hstrings ; // Handle to access uSD string
18 // var hFontx ; // Handle to access uSD fonts,
19 // Uncomment the following if uSD images, fonts
20 /*
21 // for Fat File System
22 putstr("Mounting...\n");
23 if (!(file_Mount()))
24 while (!(file_Mount()))
25 putstr("Drive not mounted...");
26 pause(200);
27 gfx_Cls();

```

```

28 pause(200);
29 wend
30 endif
31 // for (GCI) Flash File System
32 media_Init();
33 // gfx_TransparentColour(0x0020); // uncomm
34 // gfx_Transparency(ON); // uncomm
35
36 // for Fat File System
37 // hFontn := file_LoadImageControl("NoName1.dan"
38 // hstrings := file_Open("NoName1.txf", 'r') ; /
39 hndl := file_LoadImageControl("NoName1.dat",
40 // for (GCI) Flash File System
41 hndl := file_LoadImageControl(0, 0, 3); // fo
42 */
43
44 repeat
45 forever
46
47 endfunc
48

```

Note that the main function block starts with

```

16 func main()

```

and ends with

```

47 endfunc

```

This is how a function is defined in 4DGL.

Comments in 4DGL

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

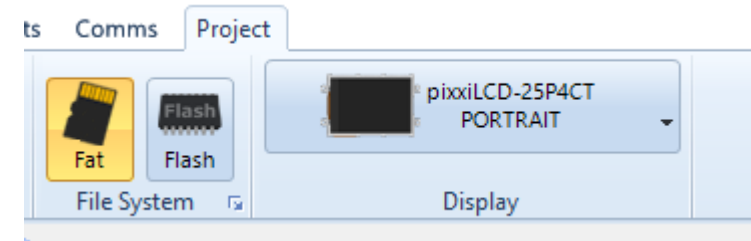
```
// Uncomment the following if uSD images, fonts or
```

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

```
20  /*
21  // for Fat File System
22  putstr("Mounting...\n");
23  if (!(file_Mount()))
24      while(!(file_Mount()))
25          putstr("Drive not mounted...");
26          pause(200);
27          gfx_Cls();
28          pause(200);
29      wend
30  endif
31  // for (GCI) Flash File System
32  media_Init();
33  //  gfx_TransparentColour(0x0020);    // uncommen
34  //  gfx_Transparency(ON);            // uncommen
35
36  // for Fat File System
37  // hFontn := file_LoadImageControl("NoName1.dan",
38  // hstrings := file_Open("NoName1.txf", 'r') ; //
39  hndl := file_LoadImageControl("NoName1.dat", "N
40  // for (GCI) Flash File System
41  hndl := file_LoadImageControl(0, 0, 3); // font
42  */
```

The File System

The graphical files on the Pixxi display modules can be stored on the On-board Serial Flash Memory (Flash) or on the external micro SD card (FAT) depending on the hardware and PmmC filesystem support.



The pixxi-LCD displays features an automatic selection of memory device. To use the micro SD card (FAT), insert a uSD card in the card slot of the module. To use the Internal Flash Memory (Flash), remove the uSD card in the slot.

Note: There is a dedicated PmmC version for both FAT and Flash File System. If you would be using the micro SD card (FAT), use the PmmC file with **-u** on its name, e.g. **pixxiLCD-25P4CT-u-A-R13**. Otherwise, if you would be using the flash memory (Flash), use the PmmC file with **-f** on its name, e.g. **pixxiLCD-25P4CT-f-A-R13**.

Please check the specific Display Module datasheet to know more about the configuration that you need to do in order to use either of them.

micro SD card

For the external microSD card, you need to remove the block command and comment the code used for the Serial Flash Memory (line 32 and 41), as shown below.

```

21 // for Fat File System
22 putstr("Mounting...\n");
23 if (!(file_Mount()))
24     while(!(file_Mount()))
25         putstr("Drive not mounted...");
26         pause(200);
27         gfx_Cls();
28         pause(200);
29     wend
30 endif
31 // for (GCI) Flash File System
32 // media_Init();
33 // gfx_TransparentColour(0x0020); // uncomment if transparency
34 // gfx_Transparency(ON); // uncomment if transparency
35
36 // for Fat File System
37 // hFontn := file_LoadImageControl("NoName1.dan", "NoName1.gcn", 1);
38 // hstrings := file_Open("NoName1.txf", 'r') ; // Open handle to acc
39 hndl := file_LoadImageControl("NoName1.dat", "NoName1.gci", 1);
40 // for (GCI) Flash File System
41 // hndl := file_LoadImageControl(0, 0, 3); // fonts, strings and
42

```

Print a String

The line

```

22 putstr("Mounting...\n");

```

will print the string "Mounting..." on the screen followed by a new line.

The While Loop

Lines 24 to 29 make up a while loop.

While loop

```

24 while(!(file_Mount()))
25     putstr("Drive not mounted...");
26     pause(200);
27     gfx_Cls();
28     pause(200);
29 wend
30 endif

```

Note that the while loop starts with

```

24 while(!(file_Mount()))

```

and ends with

```

29 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 while loop are executed as long as the condition is true.

Mount the uSD card

The function **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.

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

Delay

The function **pause(milliseconds)** delays the execution of the program for 200 milliseconds.

```
pause(200);
```

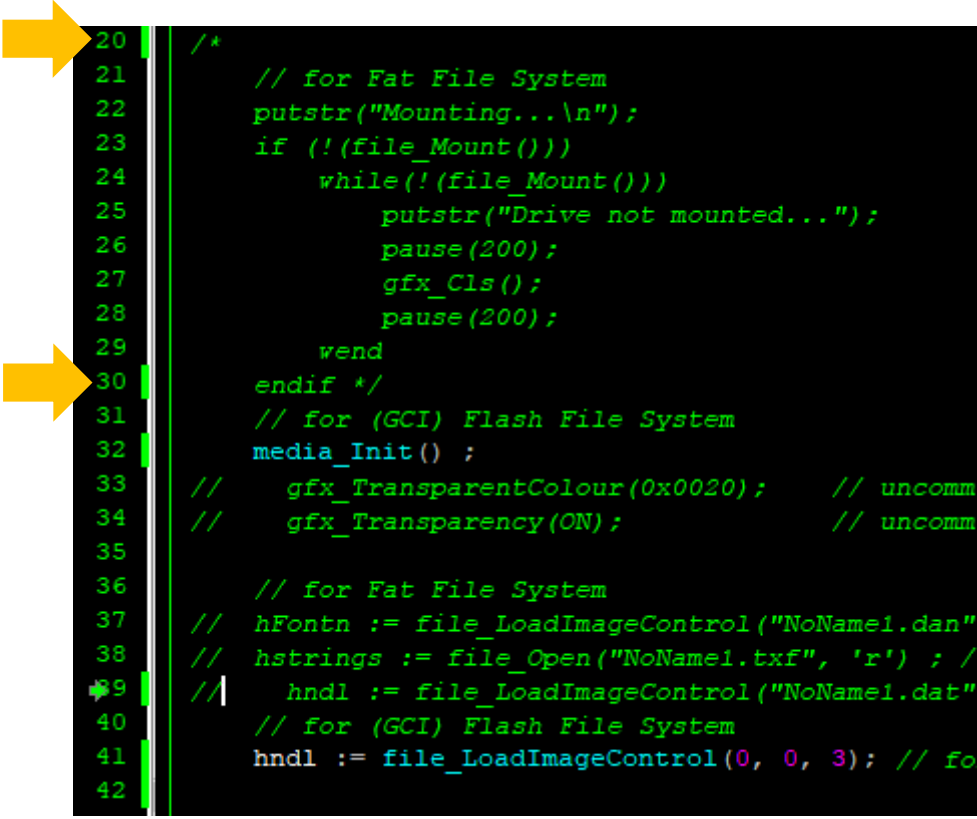
Clear the Screen

To clear the screen, use the function **gfx_Cls()**.

```
gfx_Cls();
```

Serial Flash Memory

To use the Serial Flash Memory, you need to comment out the uSD Card Initialization routine and comment the code for the micro SD card (line 39), as shown below.



```
20  /*
21      // for Fat File System
22      putstr("Mounting...\n");
23      if (!(file_Mount()))
24          while (!(file_Mount()))
25              putstr("Drive not mounted...");
26              pause(200);
27              gfx_Cls();
28              pause(200);
29      wend
30  endif */
31  // for (GCI) Flash File System
32  media_Init();
33  //  gfx_TransparentColour(0x0020);    // uncomm
34  //  gfx_Transparency(ON);             // uncomm
35
36  // for Fat File System
37  // hFontn := file_LoadImageControl("NoName1.dan"
38  // hstrings := file_Open("NoName1.txf", 'r') ; /
39  // hndl := file_LoadImageControl("NoName1.dat"
40  // for (GCI) Flash File System
41  hndl := file_LoadImageControl(0, 0, 3); // fo
42
```

The Repeat-forever Loop

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

```
45  repeat
```

and ends with

```
46 | forever
```

The program will execute the statements between lines 45 and 46 indefinitely. Here the program does nothing indefinitely after printing the text “Hello World” since no instructions exist between lines 45 and 46. 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, Workshop4 combines all of these objects into a single graphics file, which is of a format readable by 4D graphics processors.

micro SD card

There are 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.

Workshop4 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

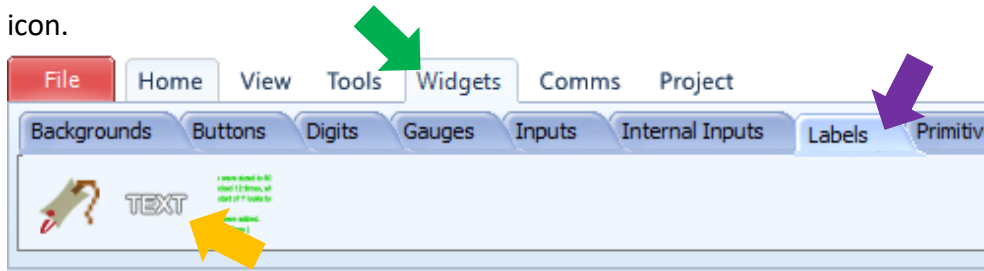
```
39 | hndl := file_LoadImageControl  
    | control("NoName3.dat", "NoName3.gci", 1);
```

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 Pixxi Internal Functions Reference Manual.

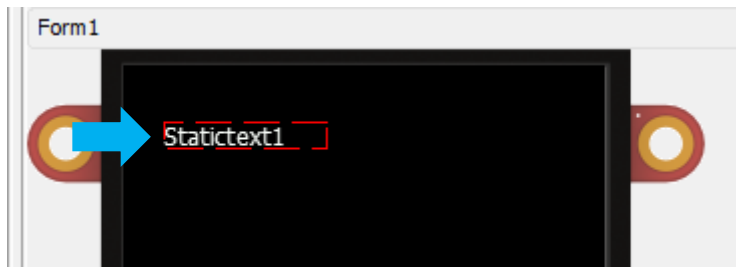
Note: Workshop4 will automatically derive the filenames of the GCI and DAT files from the project name. The filenames will be reflected in the code and in the names of the actual GCI and DAT files. When the project is saved with the name “**ViSiFirstProject**”, line 39 of the code will be updated as shown below.

Add a Static Text Object

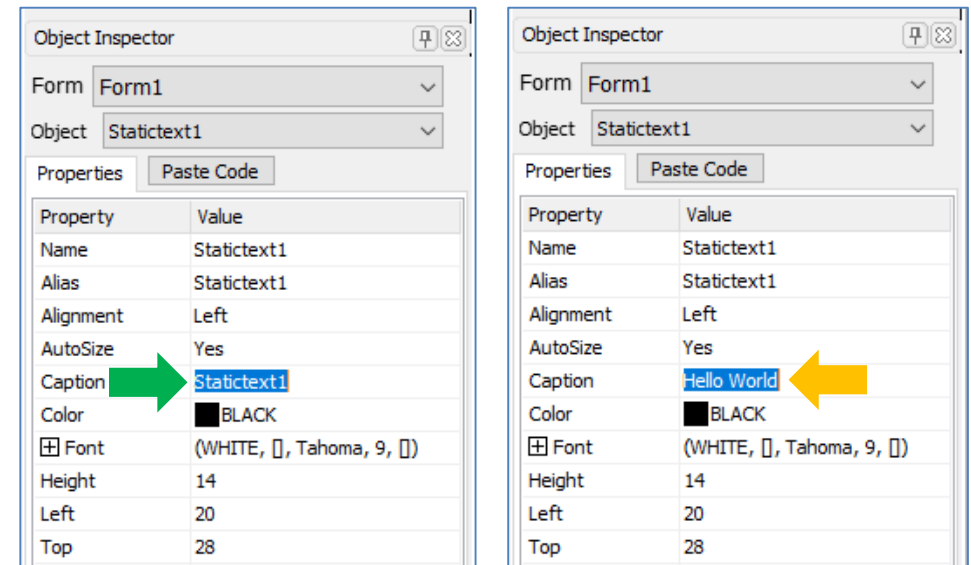
Go to the Widgets menu, select the Labels pane, and click on the static text icon.



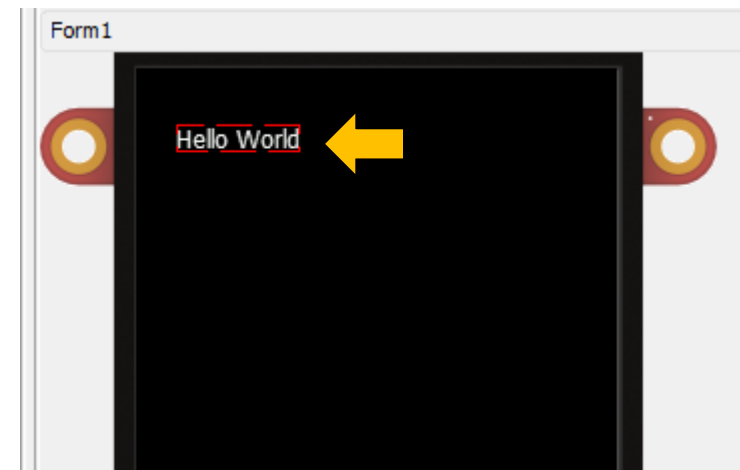
Once the static text icon is selected, click on the WYSIWYG screen to place it.



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.

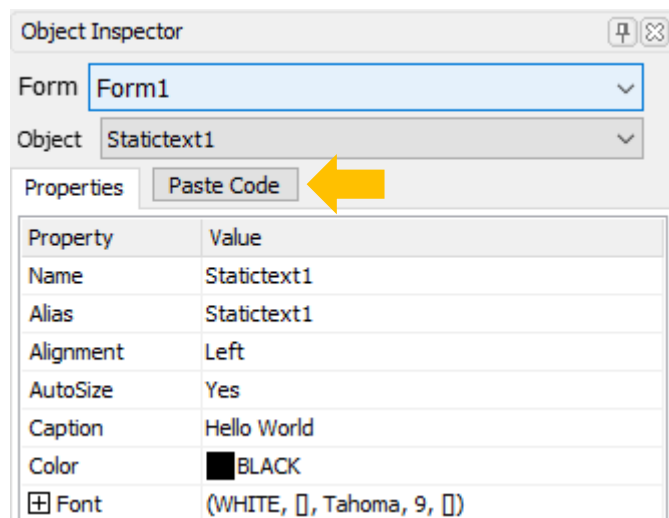


Paste the Code for a Static Text Object

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

```
41      hndl := file_LoadImageControl(0, 0, 3);
42
43      print("Hello World");
44      |
45      repeat
46      forever
47  endfunc
48
```

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.

```
41      hndl := file_LoadImageControl(0, 0, 3); // fonts, s
42
43      print("Hello World");
44
45      // Statictext1 1.0 generated 5/19/2020 3:22:35 PM
46      img_Show(hndl,iStatictext1) ;
47      |
48      repeat
49      forever
50  endfunc
51
```

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.



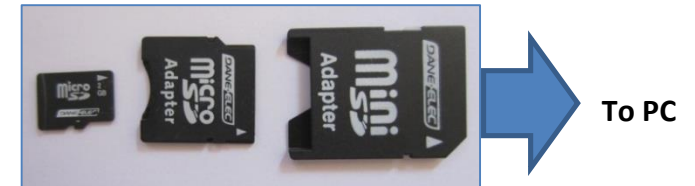
Insert the uSD Card to the PC

Note: If you are using the Serial Flash Memory, you can skip this step.

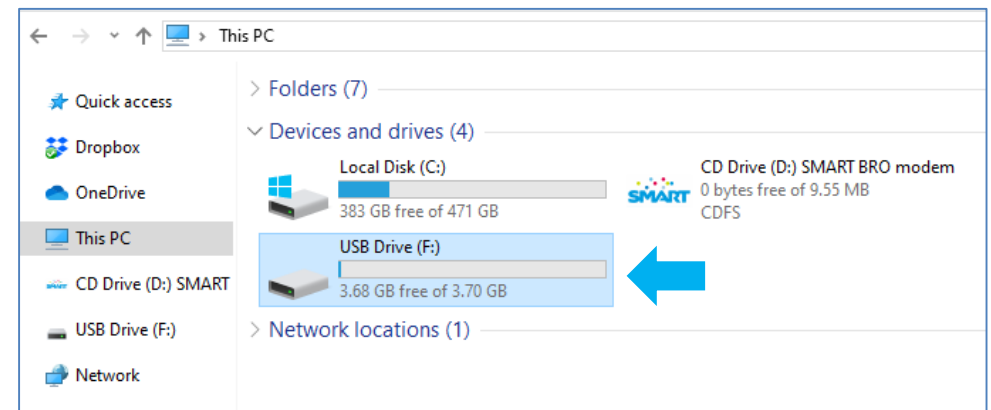
Insert the μ SD card into the USB adaptor and plug the USB adaptor into a USB port of the PC.



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



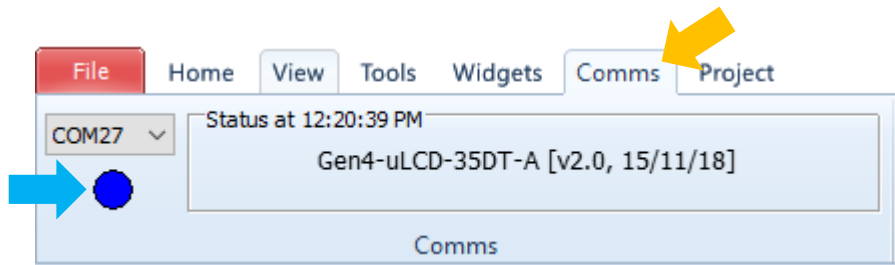
Check if the μ SD card is mounted. In the image below, it is mounted as drive F:



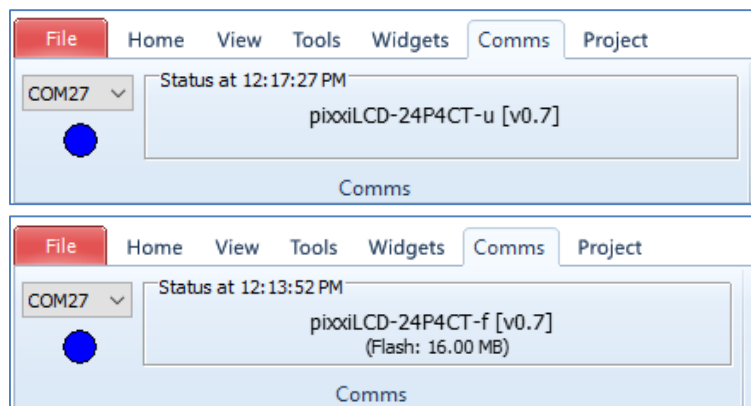
Connect the Display Module to the PC

Connect the display module to the PC using a **4D-UPA, 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 gen4-uLCD-35DT is connected to the PC.



Below is an example of how the Comms tab will look like if a pixxiLCD-24P4CT is connected to the PC. For the FAT File system, the suffix '-u' is used whilst the suffix '-f' is used for Flash as its PMMC.

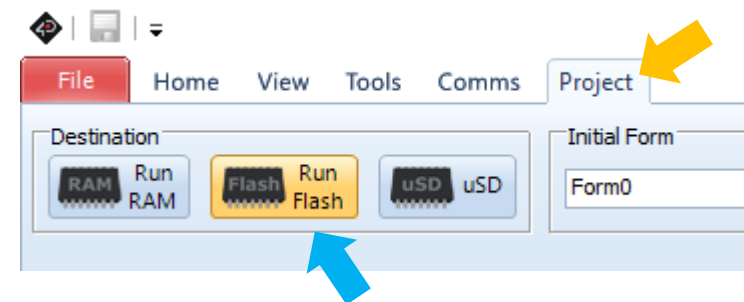


If not familiar with how to connect a 4D display to the PC using a 4D-UPA, 4D USB programming cable, or a uUSB-PA5 and with how to update the firmware, the user should consult the application note below.

General How to Update the PmmC for Pixxi

Choose the Program Destination

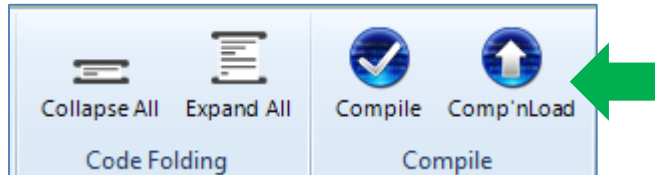
Under the **Project** menu, choose **Flash** as the destination.



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

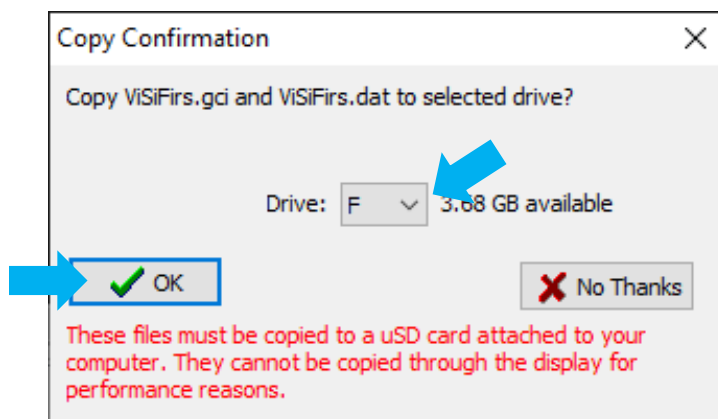
After making sure that the device is detected, go to the Home menu and click on the **Comp n'Load** (Compile and Download) button.



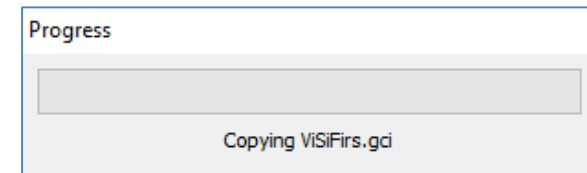
Workshop4 now builds the graphics files and copies them to the uSD card / On-board Flash Memory.

micro SD Card

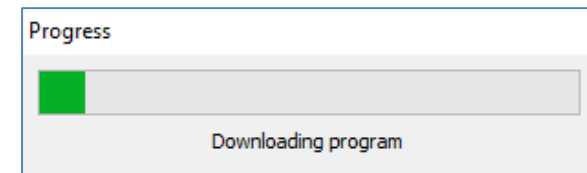
The Copy Confirmation window appears. The user will be prompted to choose the correct drive for the memory card. Now choose the correct drive by clicking on the drop-down arrow. Then click OK.



Workshop4 copies the supporting graphics files to the μ SD card.



After copying the graphics files on the uSD card, the Workshop4 then downloads the program to the display module's processor.



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

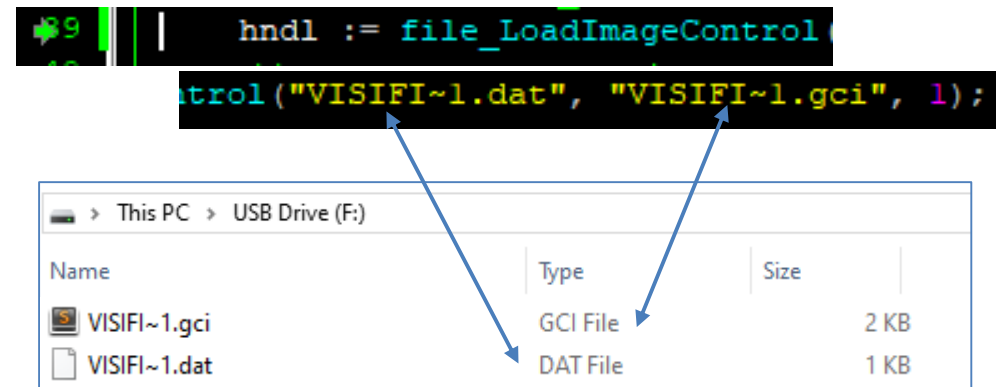
```
0 errors
0 warnings
0 notices
No Errors, code size = 169 bytes out of 32750 total
Initial RAM size = 202 bytes out of 30290 total
Program will run from ram so total initial RAM size = 371 bytes out of 30290 total
Download to Flash successful.
```

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

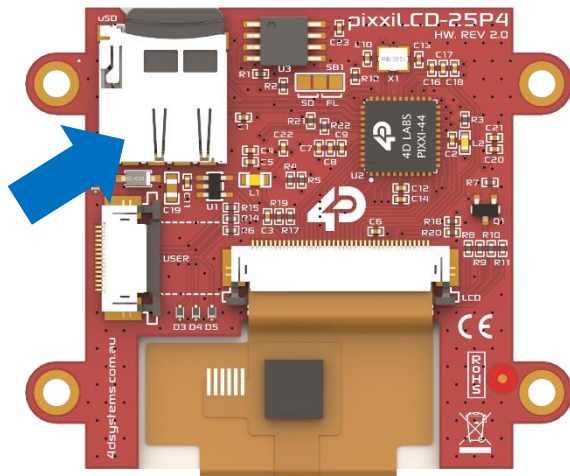
In the section “**Accessing the uSD Card**”, it was emphasized that Workshop4 automatically derives the filenames of the GCI and DAT files from the project name. The filenames follow the 8.3 format and are used in the code and as the names of the actual GCI and DAT files. Before unmounting the uSD card 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 Workshop4 automatically does this in the background for you, other factors such as the settings of your PC may sometimes prevent Workshop4 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**.

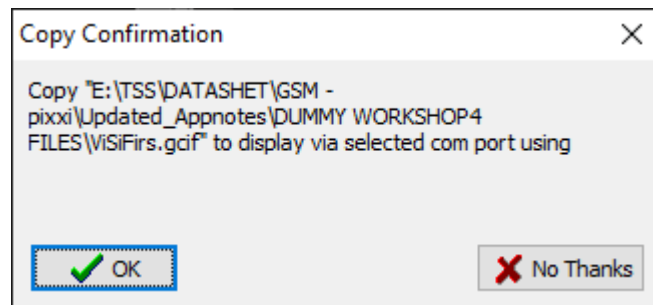
Mount the uSD Card to the Display Module

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

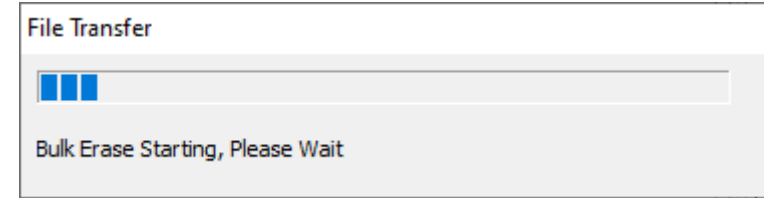


On-board Internal Flash Memory

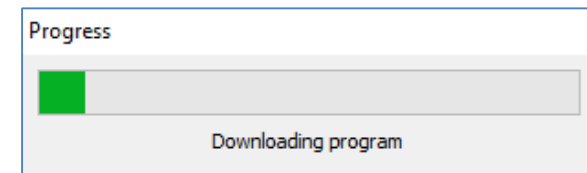
The Copy Confirmation window appears. Click OK.



A File Transfer window will pop-up. Wait for this process to end and the graphics will now show on the display module.



After copying the graphics files on the uSD card, the Workshop4 then downloads the program to the display module's processor.

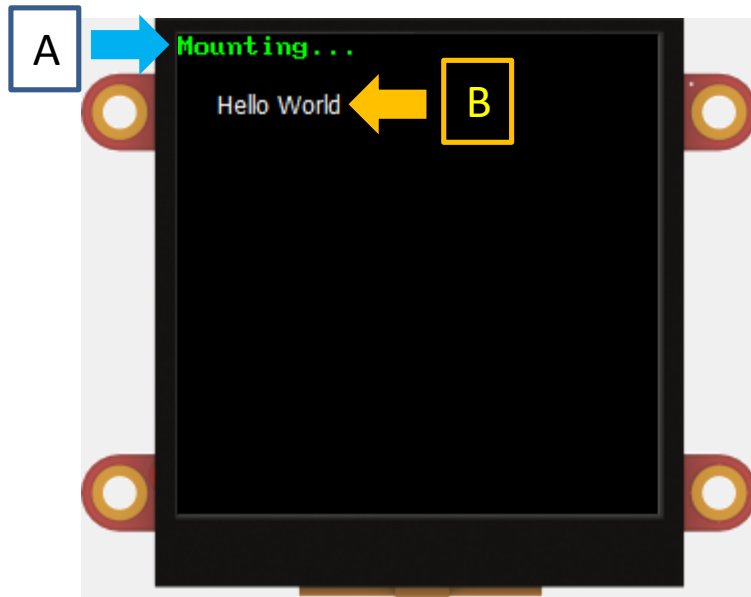


The message box will look like as shown below after a successful download.

```
GCI Flash File System requires a Flash chip of at least 4.52 KB
0 errors
0 warnings
2 notices
No Errors, code size = 130 bytes out of 32750 total
Initial RAM size = 204 bytes out of 30290 total
Program will run from ram so total initial RAM size = 334 bytes out of 30290 total
Download to Flash successful.
```

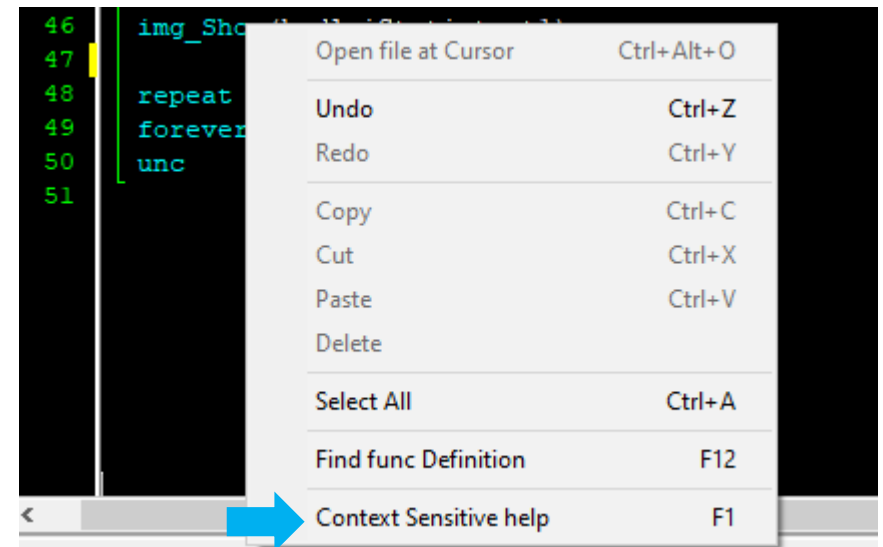
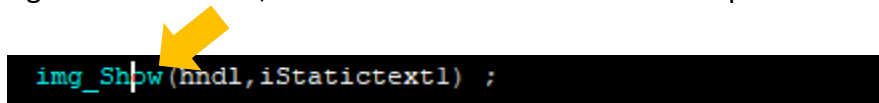
Running the Program

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



Item A is a string printed as a result of the instruction and **Item B** is an image retrieved from File System (uSD card / Internal Flash Memory) and is displayed as a result of the instructions

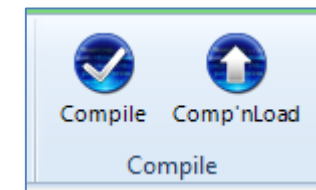
You may consult the internal functions reference manual of your display module's processor for more information. The manual can be opened from Workshop4. To do so, put the cursor on the desired function name, click on the right mouse button, then choose "Context Sensitive help".



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

Updating the Contents of the File System

One important thing to remember is that Workshop4 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 / Internal Flash Memory. 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 / Internal Flash Memory. For 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 Workshop4 can copy the newly generated files. For the Internal Flash Memory, it will directly copy the newly generated files.

Detected Changes Made with the Objects

Workshop4 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, Workshop4 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 / Internal Flash Memory.

As previously mentioned, for the uSD card, you will have to unmount the uSD card from the display module to update its contents while on the Flash Memory, because the file system is on-board, it will just copy the contents directly and update its contents.

However, if you haven't touched the WYSIWYG area or modified any field in the Object Inspector, Workshop4 will not generate a new set of support files since it is not necessary. This behaviour is important because you wouldn't want Workshop4 to be generating support files all the time, most especially when the project is large (Workshop4 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 Workshop4, after you click on the Compile button, to compile the program. You will then have to click on the Download button to make Workshop4 download the program to the processor. Alternatively, you can click on the Comp'nLoad button to make Workshop4 compile the code and download the program to the processor of the display module. Also, you will notice that Workshop4 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, Workshop4 will display the Compile button and the Compn'Load button. Clicking on the Compile button will cause Workshop4 to compile the code. To make Workshop4 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, Workshop4 will display the Compile button and the Download button. Clicking on the Compile button will cause Workshop4 to compile the code. Clicking on the Download button will cause Workshop4 to download the program to the display module processor.

If Workshop 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.

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.