



# 4D SYSTEMS

*TURNING TECHNOLOGY INTO ART*

## Serial Arduino Displaying Images from the uSD Card FAT16

Document Date: 11<sup>th</sup> March 2015

Document Revision: 1.01

## Description

This application note explores the possibilities provided by the Serial environment in Workshop for a 4D display module to work with an Arduino host. In this example, the host is an Arduino Uno board. The host can also be an Arduino Mega 2560 or Due. Ideally, the application described in this document should work with any Arduino board that supports software serial or with at least one UART serial port. [See specifications of Aduino boards here.](#)

Before getting started, the following are required:

- Any Picaso or Diablo16 display module. Visit [www.4dsystems.com.au](http://www.4dsystems.com.au) to see the latest products using any of these graphics processors.
- [4D Programming Cable](#) or [µUSB-PA5](#)
- [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

<b>Description .....</b>	<b>2</b>
<b>Content .....</b>	<b>2</b>
<b>Application Overview.....</b>	<b>3</b>
<b>Setup Procedure .....</b>	<b>3</b>
<b>Generate the Graphics File .....</b>	<b>4</b>
<i>Launch Workshop 4.....</i>	<i>4</i>
<i>Add an Image Object.....</i>	<i>5</i>
<i>Add a User Images Object .....</i>	<i>8</i>
<i>Save and Compile.....</i>	<i>12</i>
<b>Load the SPE Application.....</b>	<b>13</b>
<b>Program the Arduino Host.....</b>	<b>13</b>
<i>Load Image Control.....</i>	<i>13</i>
<i>Image Show.....</i>	<i>14</i>
<i>Setting Image Parameters.....</i>	<i>14</i>
Set the IMAGE_INDEX Parameter	15
<b>Main Loop .....</b>	<b>16</b>
Display Frame 0	17
Display Frame 1	17
Display Frame 2	18
Display Frame 3	18
Display Frame 4	19

Connect the 4D Display Module to the Arduino Host.....	19
Proprietary Information .....	20
Disclaimer of Warranties & Limitation of Liability.....	20

## Application Overview

This application note shows how an Arduino host is programmed to control a 4D display module (slave device) and make it display images. The images are contained in a graphics file saved on a  $\mu$ SD card mounted on the  $\mu$ SD card slot of the display. The Arduino host sends serial commands to the 4D display slave to access the graphics file and display the images. The graphics file is of a format that is readable by 4D-LAB's processors. This graphics file is generated using the Workshop 4 IDE.

## Setup Procedure

The following are the steps involved in doing this project:

1. **Generate the graphics file** using Workshop 4 ViSi. Copy the generated graphics file to the uSD card and mount the uSD card to the display module.
2. Configure the display module as a serial slave device by **loading it with the SPE application**.
3. **Program the Arduino host**. The Arduino board will be the master. By communicating with the slave display module, it will be able to access and display the images stored inside the graphics file on the uSD card of the display module.

## Generate the Graphics File

This application note, although written for Serial, requires the use of the ViSi environment to generate the necessary files which will be copied to the uSD card. The procedure for creating a ViSi project will be shown. Users who want to learn more about the ViSi environment may consult the application note

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

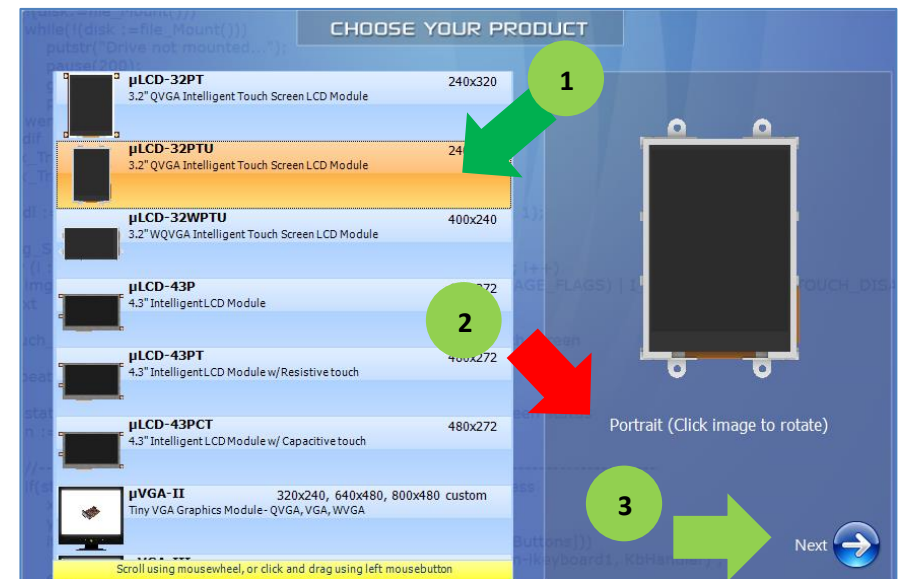
Topics discussed in that application note include instructions on how to launch Workshop 4, how to open a ViSi project, how to change the target display, how to create a new ViSi project, how to save a ViSi project, how to connect the target display to the PC, how to copy the graphics file to the uSD card, and how to compile and upload a program.

### Launch Workshop 4

Open the Workshop 4 (WS4) IDE and click "Create a new project".



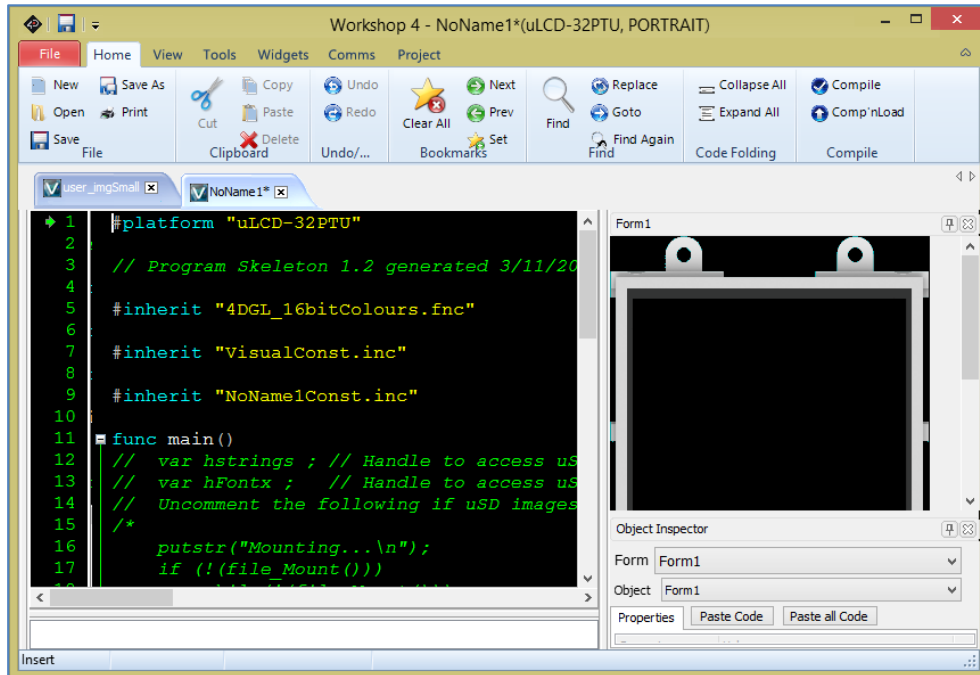
Choose your target device. For this example we select the **uLCD-32PTU**. Orientation is portrait. Click **Next**.



Select the **ViSi** environment.

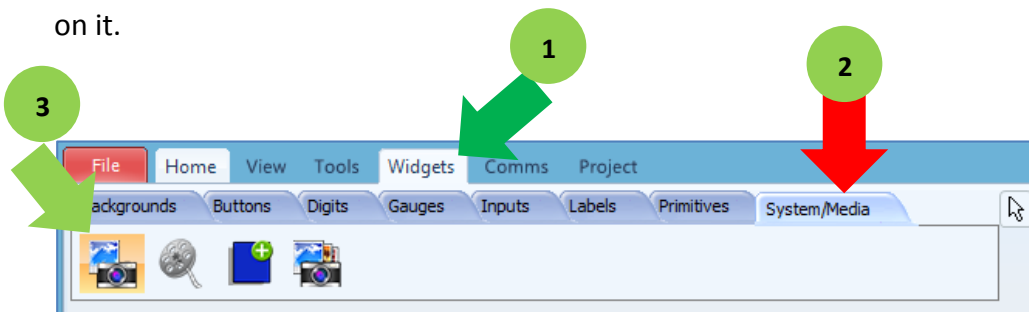


This will open the ViSi development environment window within the WS4 IDE as shown below.

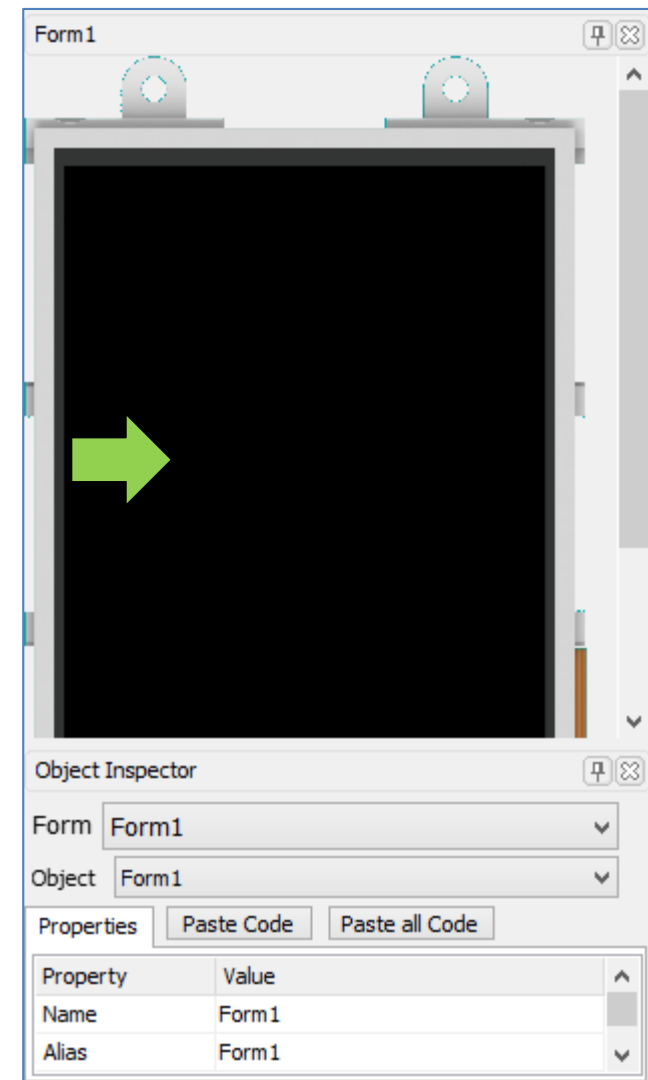


## Add an Image Object

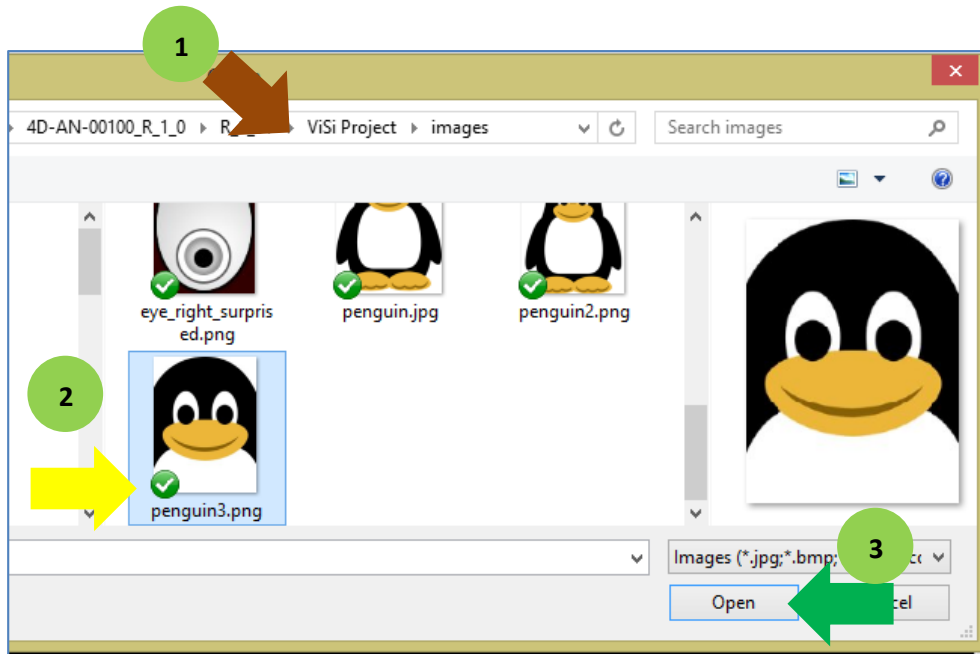
We will add the image for the face of the penguin. The icon for the image object is found under the **Systems/Media** pane of the **Widgets** menu. Click on it.



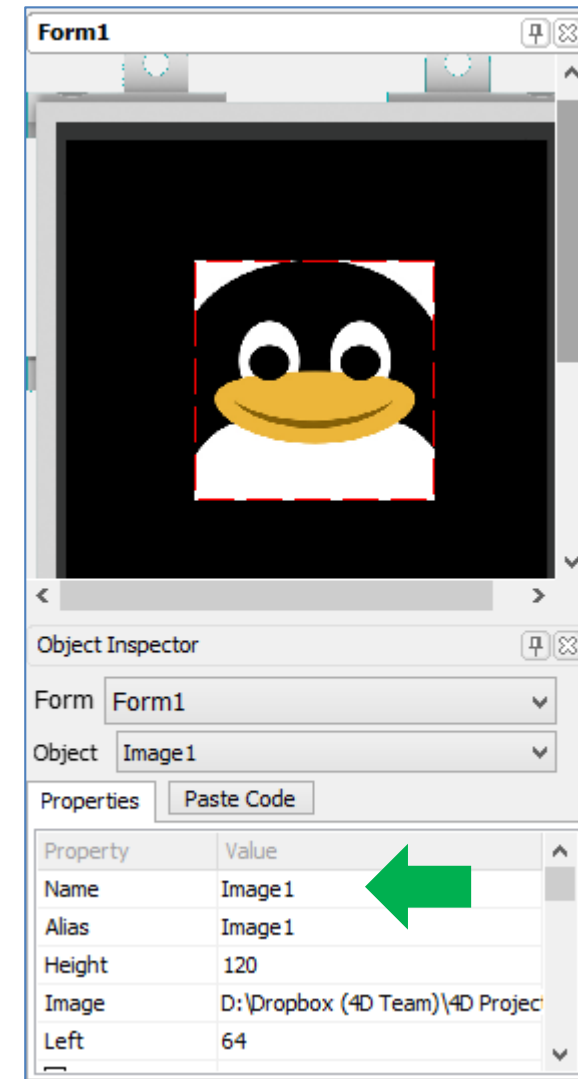
Click on the WYSIWYG (What-You-See-Is-What-You-Get) screen to place an image object.



Workshop will then ask for an image file. Here the file “**penguin3.png**” is selected. This file was resized to proportionally fit into the WYSIWYG screen. Look for this image file inside the folder “ViSi Project”.

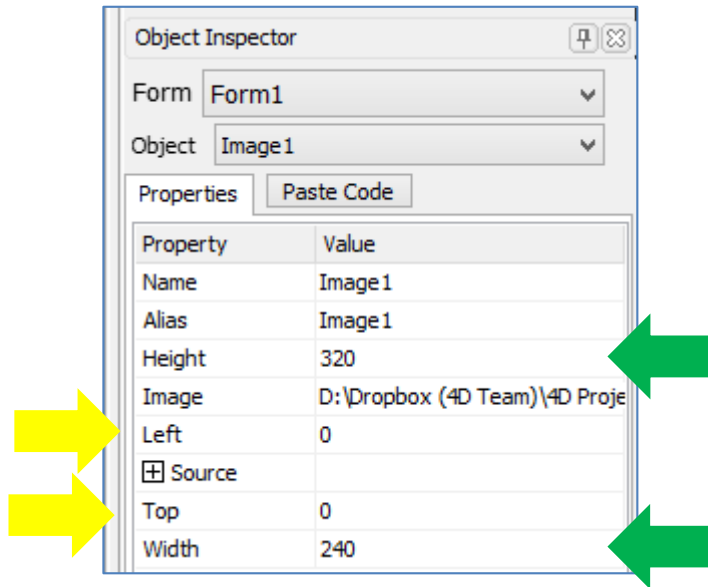


The WYSIWYG screen is updated accordingly with the image. This image object is Image1.



You can drag the edges of the object “Image1” to resize it. You can also input the width and height values directly to the Object Inspector. To make the

penguin image fill the entire WYSIWYG screen, input the width and height parameters below. Also, take note of the parameters “Left” and “Top”.



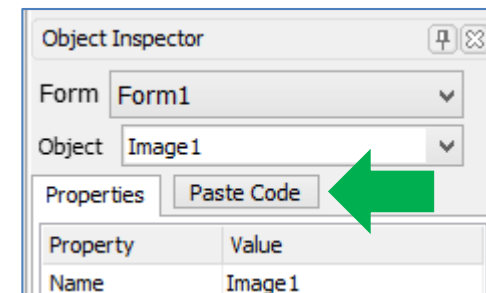
The 4DGL commands for displaying Image1 can be seen by pasting the code for it on the code area. Place the cursor on line 32 of the code area, as indicated below.

```

17  if (!(file_Mount()))
18      while (!(file_Mount()))
19          putstr("Drive not mounted...");
20          pause(200);
21          gfx_Cls();
22          pause(200);
23      wend
24  endif
25  //  gfx_TransparentColour(0x0020);    // uncomment
26  //  gfx_Transparency(ON);              // uncomment
27
28  //  hFontn := file_LoadImageControl("NoName1.dan", "NoName1.dan");
29  //  hstrings := file_Open("NoName1.txf", 'r') ; // Open text file
30  hndl := file_LoadImageControl("NoName1.dat", "NoName1.dat");
31  */
32  |
33
34  repeat
35      forever
36  endfunc
37

```

In the object inspector for Image1, click on the “Paste Code” button.



The code area is updated accordingly.

```

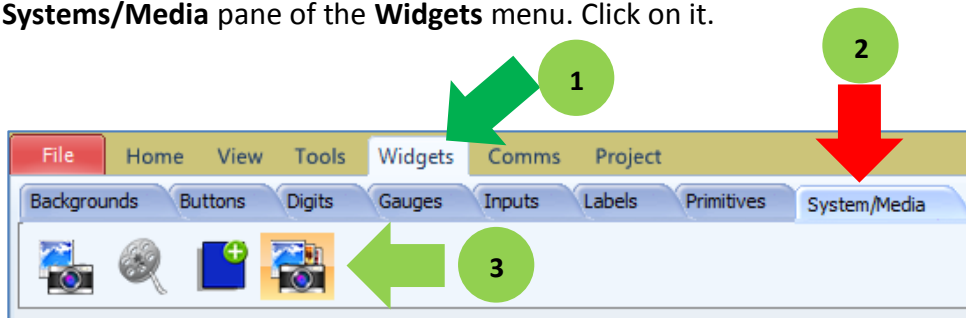
28 // hFontn := file_LoadImageControl("NoName1.dan", '
29 // hstrings := file_Open("NoName1.txf", 'r') ; //
30 hndl := file_LoadImageControl("NoName1.dat", "No
31 */
32
33 // Image1 1.0 generated 3/11/2015 2:25:36 PM
34 img_Show(hndl,iImage1) ;
35 |
36
37 repeat
38 forever

```

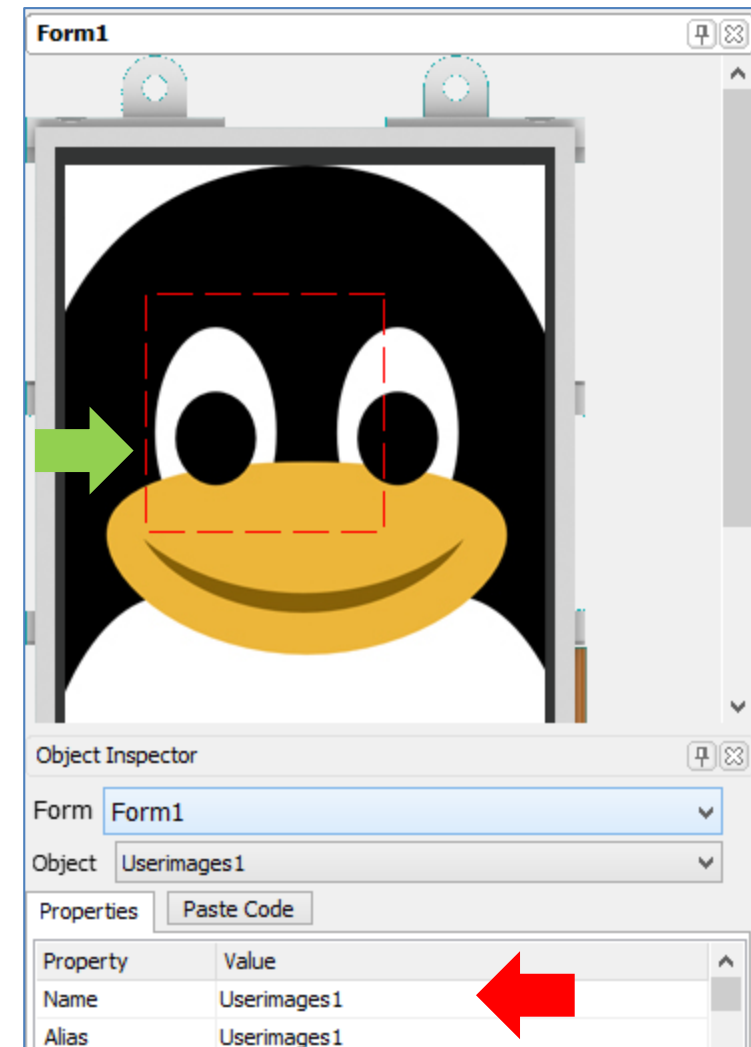
The 4DGL code is not needed in this application note. When writing the sketch for the Arduino host controller however, it is often useful to refer to the 4DGL code and functions.

## Add a User Images Object

A user images object is an object composed of several frames of images. With the correct serial command, the desired frame of a user images object can be displayed. For the eyes of the penguin, we will use two user images objects. The icon for the user images object is found under the **Systems/Media** pane of the **Widgets** menu. Click on it.

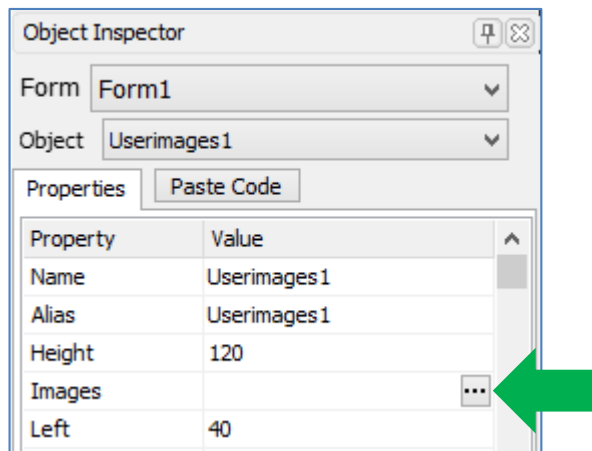


Click on the WYSIWYG screen to place a new user images object. Shown below is an empty user images object. This is Userimages1, which will be the left eye of the penguin.

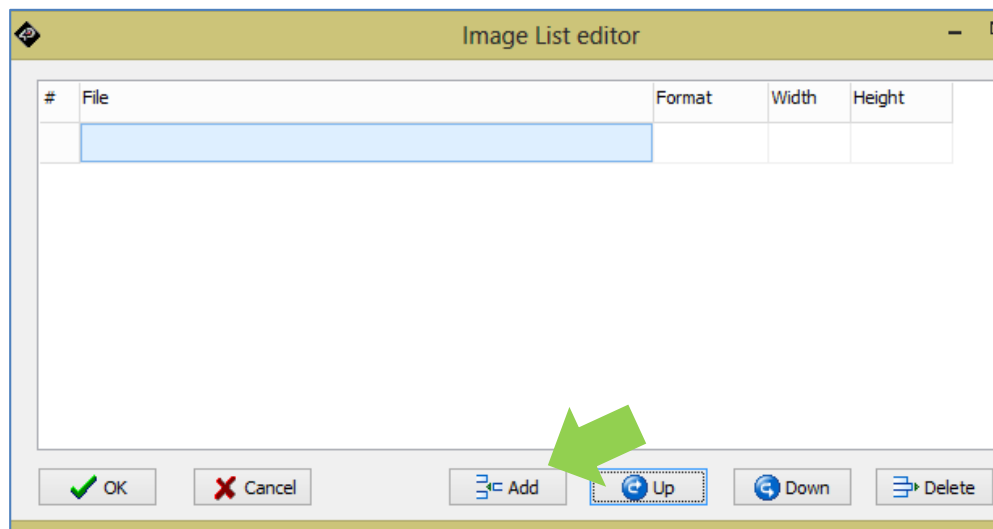




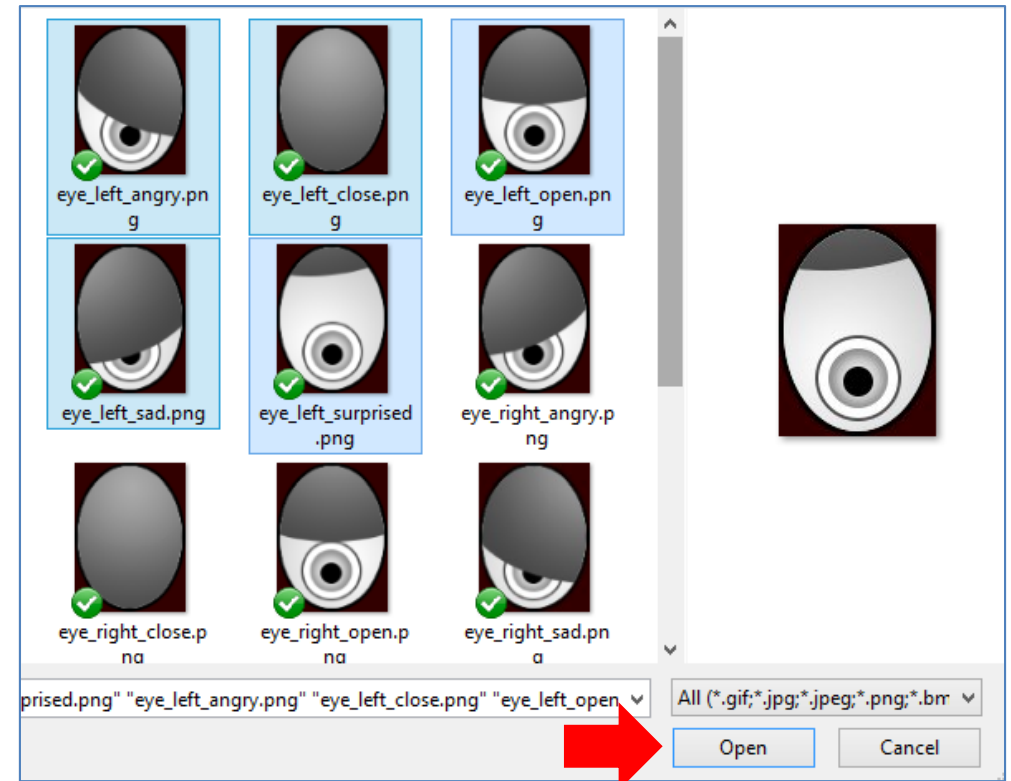
To populate Userimages1, click on the ellipsis symbol of the property “Images” of the Object Inspector.



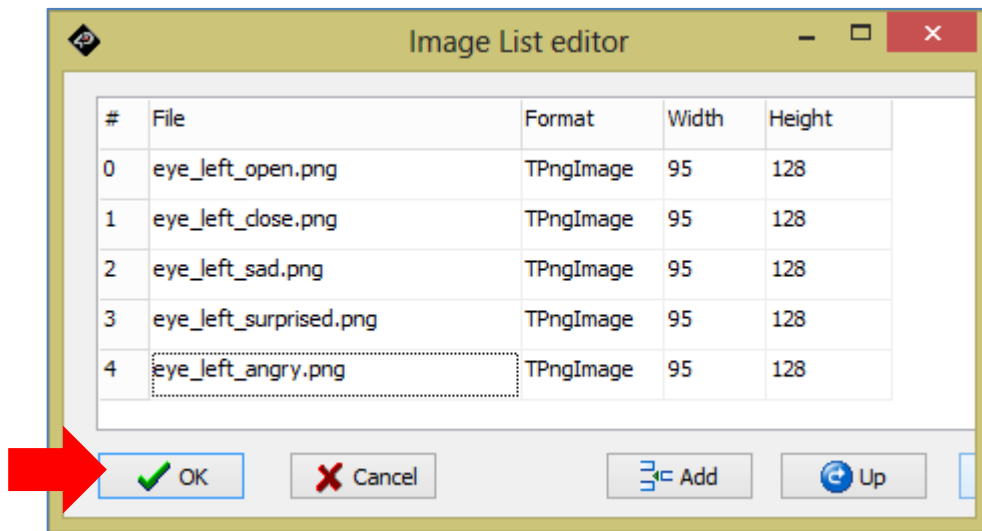
The image list editor window appears. Click on the “Add” button.



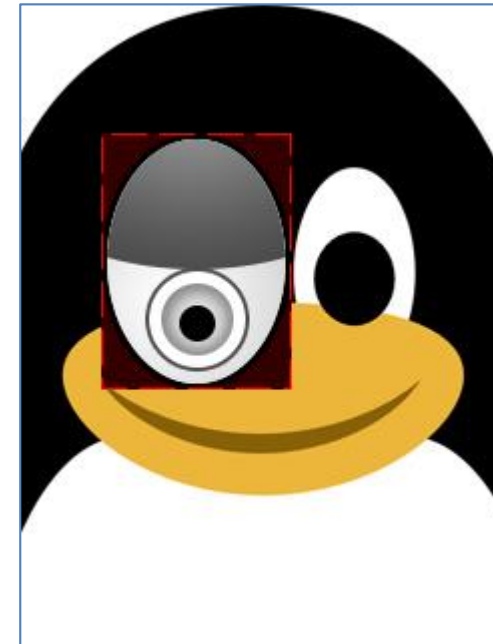
A standard open window appears. Use “Ctrl + Click” to select multiple files. Select all those for the left eye. The files are found inside the zip file attached to this application note. Then click on the open button.



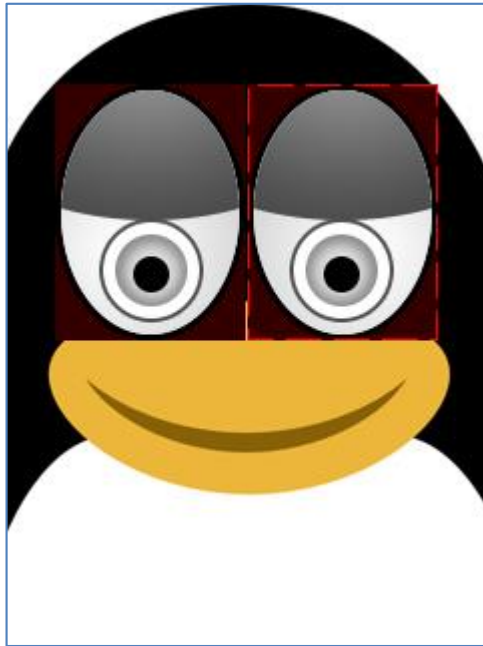
The image list editor is updated.



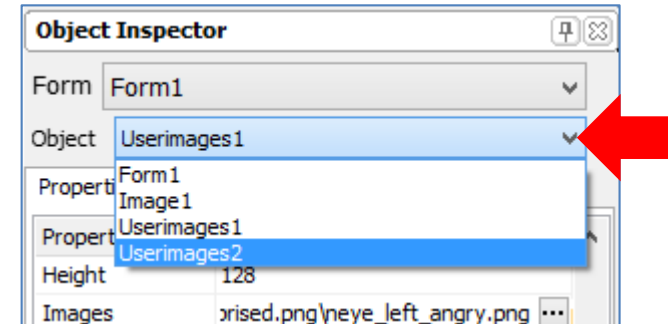
The arrangement above is important as each image frame is indexed. Use the “Up” and “Down” buttons to rearrange your images if necessary. Then click OK. The WYSIWYG screen is updated.



Repeat the same procedure for the right eye. The right eye object will be Userimages2.



Position the left and right eyes on the WYSIWYG screen such that they don't overlap. When objects disappear on the WYSIWYG screen, use the Object Inspector to navigate between different objects.



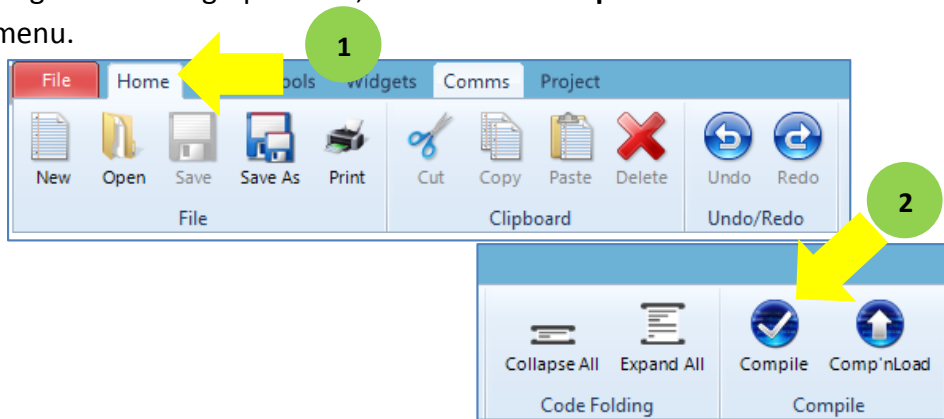
The 4DGL code for a user images object is:

```
// Userimages1 1.0 generated 3/11/2015 3:54:47 PM
img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
img_Show(hndl, iUserimages1) ;
```

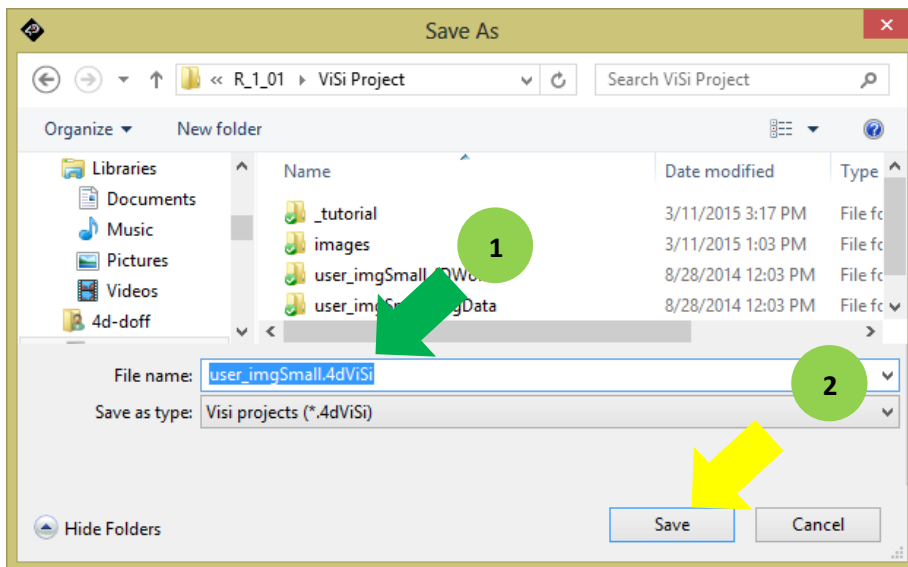
The last parameter, "**frame**", of the function "**img\_SetWord()**" is the index of the frame to be shown. In this case, frame can be any value between 0 and 4, inclusive.

## Save and Compile

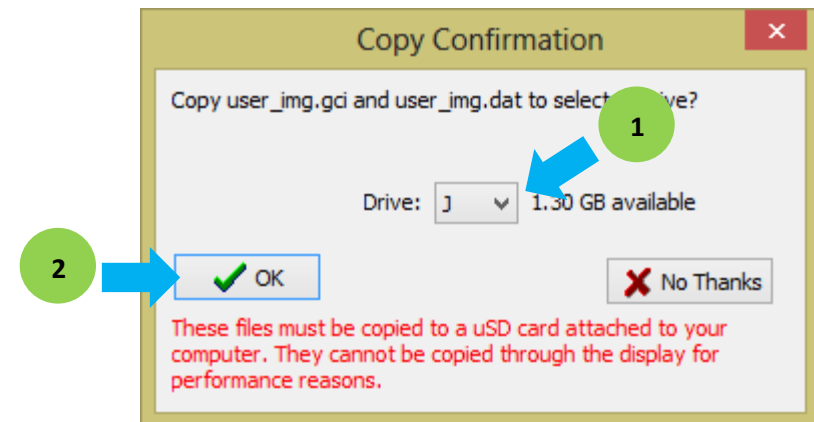
To generate the graphics file, click on the **“Compile”** button under the **Home** menu.



Workshop will ask for a filename for the project. Enter a name then click on the Save button. Here the filename is **“user\_imgSmall”**.



Workshop now builds the graphics file and copies it to the uSD card. The Copy Confirmation window appears. Make sure that your uSD card is properly mounted to your PC. You will be prompted to choose the correct drive for the memory card. Choose the correct drive by clicking on the drop down arrow. Then click OK.



Workshop now copies the graphics file to the  $\mu$ SD card. Properly unmount the uSD card from your PC then mount it to the display module. **Note that the filename of the graphics file is derived from that of the project filename. Check the contents of the uSD card for the exact filename of the generated files.** There are two files actually – the GCI file and the DAT file. The GCI file contains the actual graphics. The DAT file contains a list of the images inside the GCI file.

## Load the SPE Application

The display must be configured as a slave device before it can be controlled by a host. For instructions on how to launch Workshop 4, how to connect the display module to the PC, and how to configure the display as a slave device, kindly refer to the section “**Setup Procedure**” of any of the application notes below. Choose according to your display module’s processor.

### [Serial Picaso Getting Started - The SPE Application](#)

### [Serial Diablo16 Getting Started - The SPE Application](#)

These application notes also introduce the user to the Serial Protocol through the use of the Serial Commander.

## Program the Arduino Host

A thorough understanding of the application note [Serial Connection to an Arduino Host](#) is required before attempting to proceed further beyond this point. [Serial Connection to an Arduino Host](#) provides all the basic information that a user needs to be able to get started with the Serial Environment and Arduino. The following is a list of the topics discussed in [Serial Connection to an Arduino Host](#).

- How to download and install the Serial-Arduino library (choose a library according to your display module’s processor)

- How to modify the library for Arduino Due (due to a Due bug reported by a forum user)
- How to define the serial port to be used for talking to the display
- How to set the baud rate
- The Error Handling Routine
- How to set the Timeout Limit
- How to reset the Arduino Host and the Display
- How to let the Display Start Up
- How to set the Screen Orientation
- How to Clear the Screen
- The uSD Card Mount Routine
- How to enable message logging to the Serial Monitor of the Arduino IDE

Discussion of any of these topics is avoided in other Serial-Arduino application notes unless necessary. Users are encouraged to read [Serial Connection to an Arduino Host](#) first.

## Load Image Control

When compiling a project in the ViSi environment, Workshop uses the Graphics Composer Software tool to build the graphics file. This graphics file contains all the objects added to the project by the programmer with the use of the WYSIWYG screen. The graphics file has an extension of “.gci”, which stands for “Graphics Composer image”. Another file (with an extension of “.dat”) is generated along with the graphics file. This file contains a list of all the objects contained in the “.gci” file. These two files are copied to the uSD card, which is then mounted to the display module.

To access the objects, we now need to load these files, using the Load Image Control function.

```
//load the graphics files  
hndl = Display.file_LoadImageControl("user_img.dat", "user_img.gci", 1) ;
```

The Load Image Control function reads a control file to create an image list. This function returns a handle (pointer to the memory allocation), to the image control list that has been created. This handle can then be used for accessing and showing the desired object.

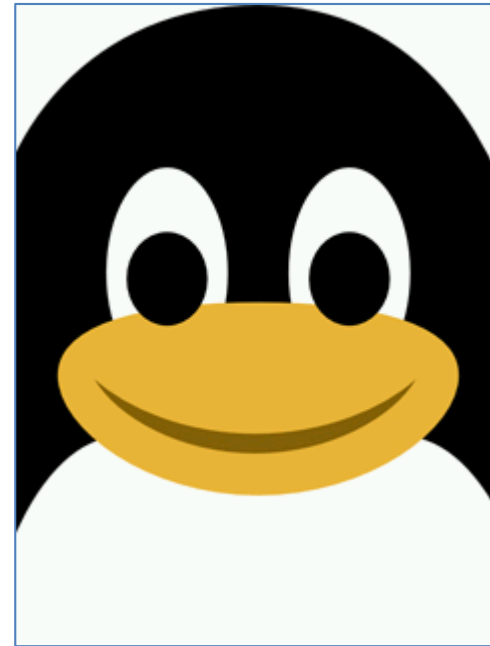
## Image Show

```
Display.img_Show(hndl,iImage1) ; //show the penguin
```

The Show Image command requires that an image control has been created with the “Load Image Control” command. This enables the displaying of the image entry in the image control and returns a non-zero value if successful. The return value is usually ignored.

**ilimage1** is the index for Image1 (the image for the penguin). Since Image1 is the third object added to the project, its index, iImage1, has a value of 2. Userimages1 (left eye of the penguin), the first object added to the project, has an index value of 0. Userimages2 (right eye of the penguin), the second object added to the project, has an index value of 1. Workshop automatically creates a constant for each image index and generates the value.

The object **Image1**.



## Setting Image Parameters

This function requires that an image control has been created with the “Load Image Control” command. It is used to set image parameters in an image entry.

The function

**Img\_SetWord(handle,index,offset,value)**

requires four parameters. The **handle** parameter is a pointer to the Image List. The **index** parameter is the index of the images in the list. The **offset** parameter is the offset of the required word in the image entry.

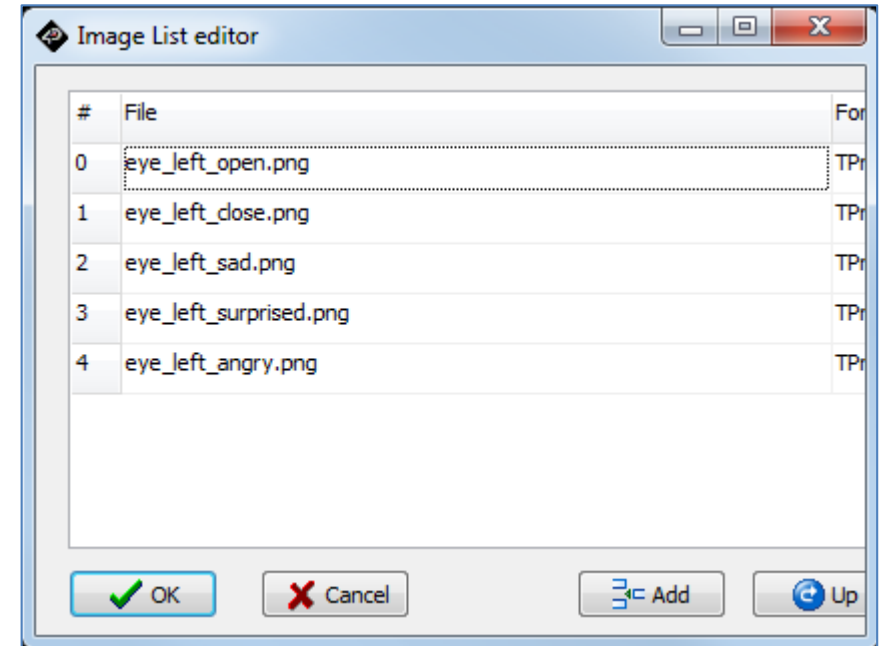
Offset	Constant name	Description
2	IMAGE_XPOS	// WORD image location X
3	IMAGE_YPOS	// WORD image location Y
6	IMAGE_FLAGS	// WORD image flags
7	IMAGE_DELAY	// WORD inter frame delay
9	IMAGE_INDEX	// WORD current frame

The **value** parameter is the word to be written to the entry.

**Note:** The “Show Image” command will show an error box for out of range frame values (IMAGE\_INDEX). Also, if the frame is set to -1, a rectangle will be drawn in background colour to blank an image.

### Set the IMAGE\_INDEX Parameter

We will now discuss how to set the value of the IMAGE\_INDEX word. Modifying the value of this word will enable us to show the desired frame of a multi-frame object. User images and video objects are some examples of multi-frame objects. Going back to the attached ViSi project, the image below shows the Image List editor.



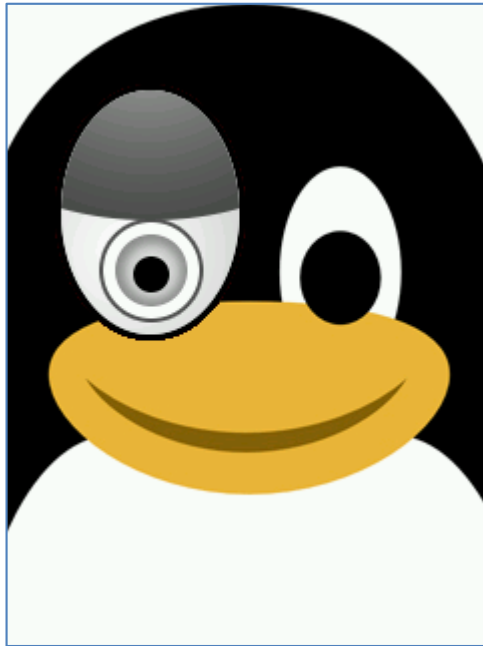
The Image List editor enumerates the images inside a user images object. Here, the images inside Userimages1 are shown. Userimages1 contains the images for the left eye while iUserimages2 contains the images for the right eye. This list can be used as a reference when showing a particular frame (IMAGE\_INDEX) using the function img\_SetWord.

To illustrate,

```
frame = 0;    //eyes open

Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
```

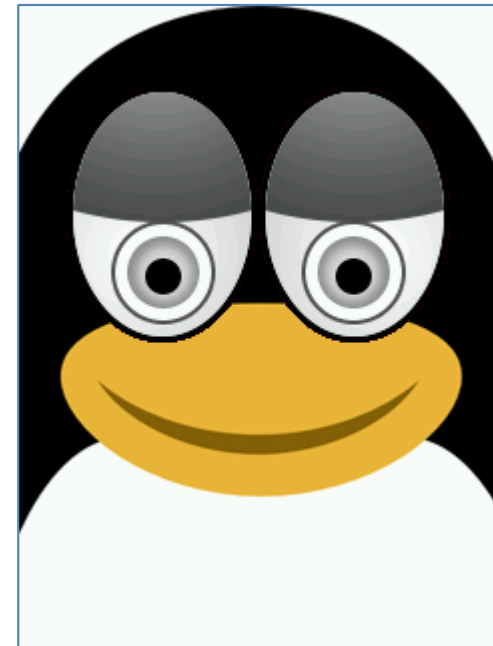
This code will show frame 0 of the left eye of the penguin. The image below shows the actual output.



Now, to display frame 0 of the right eye, write

```
frame = 0; //eyes open  
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;  
Display.img_Show(hndl,iUserimages2) ;
```

The image below shows the actual output.



Note that frame 0 of the left eye is also shown as a result of the previous commands.

### Main Loop

With the foregoing discussions on how to set the value of the IMAGE\_INDEX word, we can now proceed to writing codes for displaying the different frames of objects.



### Display Frame 0

```
frame = 0;    //eyes open
Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages2) ;
Display.putstr("  Robot is happy.  \r");
```

This code will show frame 0 of both the left and right eyes and will print the string "Robot is happy."



### Display Frame 1

```
frame = 1;    //eyes closed
Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages2) ;
```

This code will show frame 1 of both the left and right eyes.



### Display Frame 2

```
frame = 2; //eyes sad
Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages2) ;
Display.putstr(" Robot is sad. \r");
```

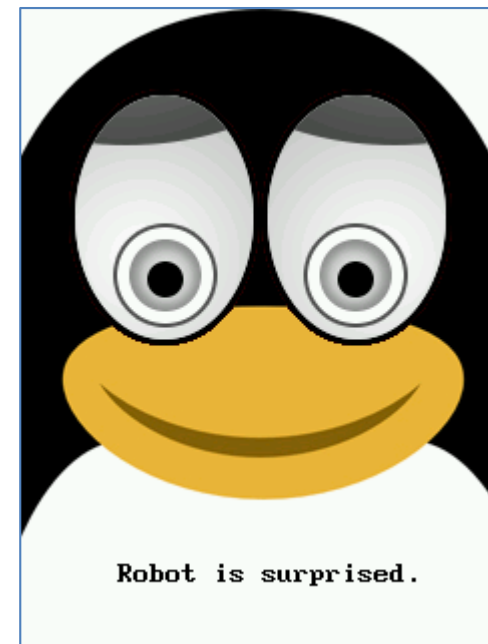
This code will show frame 2 of both the left and right eyes and will print the string "Robot is sad."



### Display Frame 3

```
frame = 3; //eyes surprised
Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages2) ;
Display.putstr("Robot is surprised.\r");
```

This code will show frame 3 of both the left and right eyes and will print the string "Robot is surprised."



## Display Frame 4

```
frame = 4;    //eyes angry
Display.img_SetWord(hndl, iUserimages1, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages1) ;
Display.img_SetWord(hndl, iUserimages2, IMAGE_INDEX, frame) ;
Display.img_Show(hndl,iUserimages2) ;
Display.putstr(" Robot is angry. \r");
```

This code will show frame 4 of both the left and right eyes and will print the string "Robot is angry."



## Connect the 4D Display Module to the Arduino Host

Refer to the section **"Connect the Display Module to the Arduino Host"** of the application note [Serial Connection to an Arduino Host](#) for the following topics:

- Using the New 4D Arduino Adaptor Shield (Rev 2.00)
  - Definition of Jumpers and Headers
  - Default Jumper Settings
  - Change the Arduino Host Serial Port
  - Power the Arduino Host and the Display Separately
- Using the Old 4D Arduino Adaptor Shield (Rev 1)
- Connection Using Jumper Wires

## 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.