

# **Serial Arduino Block Image Transfer**

DOCUMENT DATE:**28th May 2019**DOCUMENT REVISION:**1.1** 





## 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 of the following 4D Picaso display modules:

gen4-uLCD-24PT gen4-uLCD-28PT gen4-uLCD-32PT uLCD-24PTU uLCD-28PTU 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and other superseded modules which support the ViSi Genie environment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 <u>www.4dsystems.com.au/products</u> to see the latest display module products that use the Diablo16 processor.

- <u>4D Programming Cable</u> or <u>μUSB-PA5</u>
- <u>micro-SD (μSD)</u> memory card
- <u>Workshop 4 IDE</u> (installed according to the installation document)
- Any Arduino board with a UART serial port
- microSD Card Shield for Arduino
- <u>4D Arduino Adaptor Shield</u> (optional) or connecting wires
- <u>Arduino IDE</u>
- 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.

#### APPLICATION NOTES

## Content

Description2
Content3
Application Overview3
Setup Procedure3
Using the LCD Image Converter4
Program the Arduino Host6
Initialize SD shield6
Get Width and Height7
Perform BLIT (Block Image Transfer)7
Connect the 4D Display Module to the Arduino Host9
Proprietary Information10
Disclaimer of Warranties & Limitation of Liability10

# **Application Overview**

This application note shows how to perform Block Image Transfer (BLIT) using the Arduino Serial library. In this application note, an Arduino Uno board is used as the host, and a uLCD-32PTU is used as the slave display. First, the desired image is converted to an array, which contains all the hexadecimal values for the pixels of the image. A third party application can be used for this process. The array is then saved to a file. The file is loaded to a uSD card, which is then mounted to the uSD card shield attached to the uSD card, and will start sending the individual pixel bytes to the 4D display. The display will then show the pixels on the screen.

### Setup Procedure

The display must be configured as a slave device first 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 thru the use of the Serial Commander.

# Using the LCD Image Converter

A third party application is used to convert an image to a character array. The application can be downloaded here:

https://code.google.com/p/lcd-image-converter/.

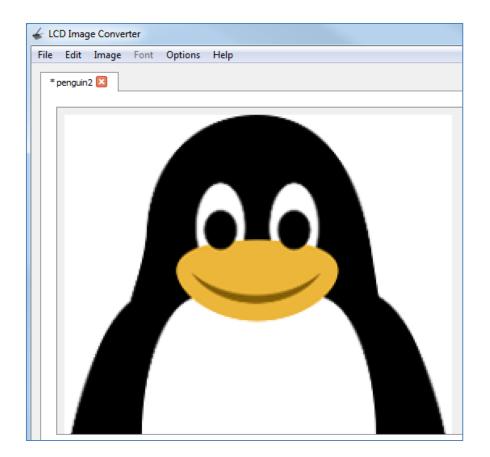
After the image is converted, the content of the array will then be copied to a text file. The text file has a format that must be followed:

```
[width, height]
```

```
Image character array content
```

This text file will be copied to the uSD card, which will be mounted to the uSD card shield of the Arduino.

In the LCD Image Converter application, you can load an existing image by going to File->Open. Here, an image of a penguin is opened.



#### APPLICATION NOTES

For this project, these were the conversion settings used under Options-> Conversion:

🖌 Options						? ×	
Preset: Color R5G6B5	Save As	Remove					
Prepare Matrix							
Split to rows	Prefix:						
Trailing bits:	0x						
Set to '1'	Suffix:						
Block size:							
8 bit 🔻	Delimiter:						
RLE Compression	RLE Compression ,						
Byte order:							
Little-Endian							
Big-Endian							
	1						
Show Preview						ОК	

If the setup is ok, go to File-> Convert or press "Ctrl+P". The converted output can be viewed using a text editor.

<pre>static const uint8_t image_data_penguin2[153600] = {</pre>								
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,
0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,	0xff,

Only the contents of the array are to be copied to the text file. The final text file should look like the image below:

IC	240,	320]							
000000000000000000000000000000000000000	xff, xff, xff, xff, xff, xff, xff, xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff,	0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0 0 0	x5a, x63, xff,	0xcb, 0x0c, 0xff,	0x42, 0x84, 0xff,	0x28, 0x30, 0xff,	0x31, 0xad, 0xff,	0x86, 0x55, 0xff,	0x21, 0xce, 0xff,	0x04, 0x79, 0xff,	

The first line contains the width and height of the image and the succeeding lines contain the characters of the array. The Arduino program will parse the text file and expects it to be in this format.

# Program the Arduino Host

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

- Understanding the basic Arduino Demo Sketch (without Message Logging)
- How to download and install the Serial-Arduino library
- How to open a serial port for communicating with the display and how to set the baud rate
- The Acknowledgement Byte
- The Error Handling Routine
- How to set the Timeout Limit
- How to set the baud Rate
- 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
- Understanding the Arduino Demo Sketch (with Message Logging)

Discussion of any of these topics is avoided in other Serial-Arduino application notes unless necessary. Users are encouraged to read <u>Serial</u> <u>Connection to an Arduino Host</u> first.

#### Initialize SD shield

```
if (!SD.begin(10)) {
   Serial.println("initialization failed!");
   return;
}
Serial.println("initialization done.");
```

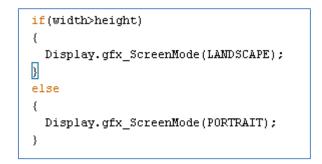
This code initializes the uSD card. The function **begin(cspin)** of the SD library requires one parameter. The **cspin** parameter is the pin connected to the chip select line of the SD card. This defaults to the hardware SS line of the SPI bus.

#### **Get Width and Height**

```
myFile = SD.open("BLIT.txt");
if (myFile)
{
 myChar = myFile.read();
 if(myChar == '[')
  {
   while(myChar != ',')
      myChar = myFile.read();
      if(myChar != ',')
      w[counter] = myChar;
      counter++;
   counter = 0;
   if(myChar == ',')
    {
      while(myChar != ']')
        myChar = myFile.read();
        if(myChar != ']')
        h[counter] = myChar;
        counter++;
      counter = 0;
    -}
```

This code parses the width and height data contained in the string "[width, height]" of the text file and stores them in variables.

The width and height can be used to determine the screen mode or the orientation of the image. If the width is greater than the height then the image is landscape. If the height is greater than the width then the image is portrait.



#### Perform BLIT (Block Image Transfer)

Use **blitFlash()** to perform block image transfer.

```
blitFlash(0,0,width,height,myFile);
myFile.close();
```

The first two parameters are the pixel coordinates of the top-left corner of the image. The **width** and **height** parameters are the values parsed from the text file. The **myFile** variable is the filename of the text file that contains the equivalent array of the image.

void blitFlash(word X, word Y, word Width, word Height,File filename)
{
 DisplaySerial.print((char)(F\_blitComtoDisplay >> 8)) ;
 DisplaySerial.print((char)(F\_blitComtoDisplay)) ;
 DisplaySerial.print((char)(X >> 8)) ;
 DisplaySerial.print((char)(X) ;
 DisplaySerial.print((char)(Y) ;
 DisplaySerial.print((char)(Y)) ;
 DisplaySerial.print((char)(Width >> 8)) ;
 DisplaySerial.print((char)(Width) >> 8)) ;
 DisplaySerial.print((char)(Height)) ;
 Display.GetAck() ;

Inside the routine **blitFlash()**, the function **blitFromFlash()** is called. Below is the definition of this function.

```
void blitFromFlash(File Source, long int Size)
{
        int wk ;
        long int i ;
        File myFile;
        myFile = Source;
    while (myFile.available()) {
        myChar = myFile.read();
        if(myChar == '0')
          myChar = myFile.read();
          if(myChar =='x')
            test[0] = myFile.read();
            test[1] = myFile.read();
            test[2] = 0;
          3
          result = asciihexTohex(test);
          if(result != -1)
            wk = result;
            DisplaySerial.write(wk) ;
        -}
```

The function **blitFromFlash()** is derived from **blitComtoDisplay()**, which is a function defined in the Arduino-Picaso-Serial and Arduino-Diablo16-Serial libraries. The prototype for **blitComtoDisplay()** is:

# void blitComtoDisplay(word X, word Y, word Width, word Height, t4DByteArray Pixels)

The function **blitComtoDisplay()** will BLIT (Block Image Transfer) pixel data to the screen thru the serial port. This function requires five parameters. The **x** and **y** parameters specify the horizontal and vertical position of the topleft corner of the image. The **width** parameter is the width of the image to be displayed. The **height** parameter is the height of the image to be displayed. The parameter "**Pixels**" is the starting address of an array containing the pixel data to be plotted on the display. For larger images, the equivalent array that contains the pixel data can become very large, such that it exceeds the available program memory space of an Arduino Uno or a Mega2560. To solve this issue, the array containing the pixel data of the image to be displayed is instead stored in a file saved on a uSD card. The Arduino program will then just have to open the file and start the process of "blitting" the contained data. To easily access the uSD card, the functions **blitFlash()** and **blitFromFlash()** were created.

# 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 <u>Serial Connection to an Arduino Host</u> 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.