



# Serial Arduino uSD Card Access

## FAT16

DOCUMENT DATE: 7<sup>th</sup> MAY 2020  
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 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 Procedures</b> .....	<b>3</b>
<b>Program the Arduino Host</b> .....	<b>3</b>
<i>The uSD card Mounting Routine</i> .....	<b>4</b>
<i>Creating a Text File</i> .....	<b>4</b>
<i>Writing to the Created Text File</i> .....	<b>5</b>
<i>Reading from a Text File from the uSD Card</i> .....	<b>6</b>
<i>File Unmount</i> .....	<b>7</b>
<i>Check the Contents of the uSD Card</i> .....	<b>7</b>
<b>Connect the 4D Display Module to the Arduino Host</b> .....	<b>8</b>
<b>Proprietary Information</b> .....	<b>9</b>
<b>Disclaimer of Warranties &amp; Limitation of Liability</b> .....	<b>9</b>

## Application Overview

In this application note, the basic commands available for accessing a FAT16-formatted uSD card are discussed. The sample application uses the File Mount, File Open, File Write, File Read, and File Close commands to show the basics of writing to and reading from a uSD card.

## Setup Procedures

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 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.

## The uSD card Mounting Routine

```
Display.putstr("Mounting...\n"); //print a string
if(!(disk = Display.file_Mount()))
{
    while(!(disk = Display.file_Mount()))
    {
        Display.putstr("Drive not mounted...");
        delay(200);
        Display.gfx_Cls();
        delay(200);
    }
}
```

This code starts up the FAT16 disk file services and allocates a small 20 byte control block for subsequent use. When you open a file using the “File Open” command a further 512 + 44 = 556 bytes are attached to the FAT16 file control block. When you close a file using the “File Close” command, the 556 byte allocation is released leaving the 20 byte file control block. The File Mount command must be called before any other FAT16 file related functions can be used. The control block and all FAT16 file resources are completely released with the “File Unmount” command.

The program will not continue unless a uSD card is inserted in the display.

## Creating a Text File

```
//uSD create a text file and write
word handle;
handle = Display.file_Open("SAMPLE.txt", 'w');
```

The function

### file\_Open (filename, mode)

requires two parameters. The filename parameter is the name of the file to be opened (passed as a string). The mode parameter can be any of the following:

FILE_READ or 'r'
FILE_WRITE or 'w'
FILE_APPEND or 'a'

**Note:** If a file is opened for write mode 'w', and the file already exists, the operation will fail.

This code creates a new text file named “SAMPLE.txt” in the uSD card. The mode is 'w', which means that the file is opened for writing. The File Open returns handle if file exists. The file 'handle' that is created is now used as reference for 'filename' for further file commands such as “File Close”, etc.

For File Write and File Append modes ('w' and 'a') the file is created if it does not exist. If the file is opened for append and it already exists, the file pointer is set to the end of the file ready for appending, else the file pointer will be set to the start of the newly created file.

If the file was opened successfully, the internal error number is set to 0 (i.e. no errors) and can be read with the "File Error" command. For File Read mode ('r') the file must exist else a null handle (0x00, 0x00) is returned and the 'file not found' error number is set which can be read with the "File Error" command.

### Writing to the Created Text File

```
Display.putstr("Writing to USD... \n");
Display.putstr("HELLO WORLD to SAMPLE.txt \n");
delay(2000);
Display.file_Write(11,"HELLO WORLD",handle);
Display.file_Close(handle);
delay(2000);
Display.putstr("DONE!");
delay(1000);
Display.gfx_Cls();           //clear the screen
```

This code writes a string "HELLO WORLD" to the text file "SAMPLE.txt" using **file\_Write()**. After writing, it closes the file using **file\_Close()**. The File Write command returns the current value of the file pointer. The File Close command will close the previously opened file.

The function

#### **file\_Write (size, source, handle)**

requires three parameters. The size parameter is the number of bytes to be written. The source parameter is string of data without null terminator. And the handle parameter is the handle that references the file to write.

The function

#### **file\_Close(handle)**

requires one parameter. The parameter handle is the file handle that was created by the "File Open" command which is now used as reference 'handle' for the filename. The handle can be used in other file functions such as in this function - to close the file.

Output:

```
Writing to uSD...
HELLO WORLD to SAMPLE.txt
DONE!
```

### Reading from a Text File from the uSD Card

```
//uSD read created txt file
handle = Display.file_Open("SAMPLE.txt",'r');
char Reader[11];
Display.putstr("READING SAMPLE.TXT: \n");
Display.file_Read(Reader,11,handle);
Display.putstr(Reader);
Display.file_Close(handle);
```

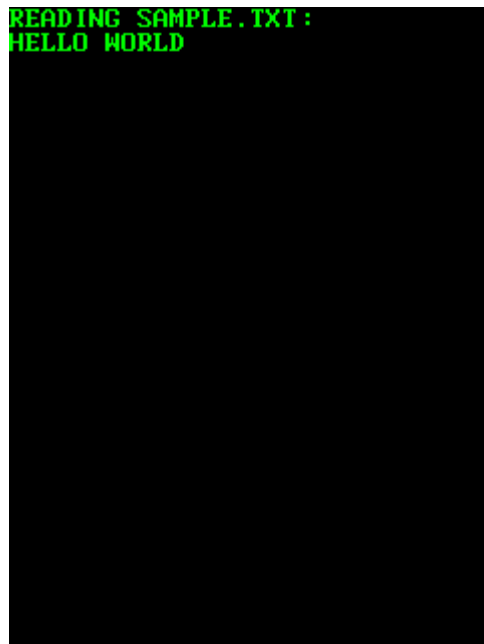
This code opens the file "SAMPLE.txt" previously created for reading and stores in a character array the contents of the text file using the function **file\_Read()**. The File Read reads the number of bytes specified by "size" from the file referenced by "handle" into a destination memory buffer.

The function

**file\_Read (destination, size, handle)**

requires three parameters. The destination parameter is the destination memory buffer, which is a normal word-aligned address. The size parameter is the number of bytes to be read. The handle parameter is the handle that references the file to be read.

Output:



```
READING SAMPLE.TXT:  
HELLO WORLD
```

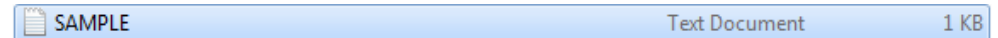
### File Unmount

```
Display.file_Unmount();
```

The “File Unmount” command releases any buffers for FAT16 and unmounts the Disk File System. This function is to be called to close the FAT16 file system.

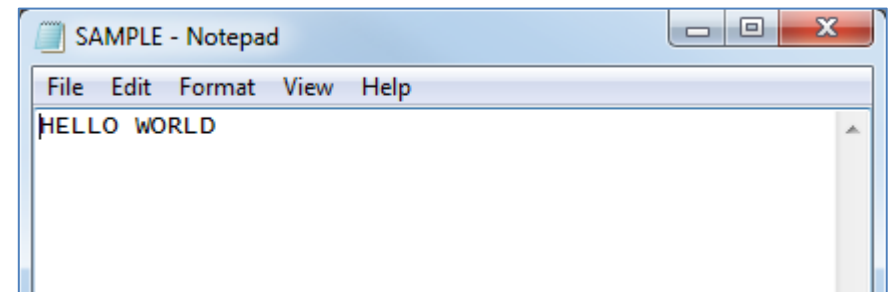
### Check the Contents of the uSD Card

After the execution of the program, the uSD card can now be removed from the display module and can be inserted to the PC for content verification. The file “SAMPLE.txt” should be present in the uSD card.



SAMPLE Text Document 1 KB

This is the text file created by the function **file\_Open()**. The contents of this file were written using the function **file\_Write()**. As seen in the image below, the string that was written by the Serial Arduino program is the same as the contents of the text file.



```
SAMPLE - Notepad  
File Edit Format View Help  
HELLO WORLD
```

## 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.