APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Serial Connection to an Arduino Host

DOCUMENT DATE:	**20th May 2019**
DOCUMENT REVISION:	**1.1**

W W W . 4 D S Y S T E M S . C O M . A U

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.

- Any of the following 4D Goldelox display modules:

  uOLED-96-G2 uOLED-128-G2 uOLED-160-G2
  uLCD-144-G2

  or any superseded module that supports the Serial environment. Visit www.4dsystems.com.au to see the latest products using the Goldelox graphics processor.

- Any of the following Picaso display modules:

  uLCD-24PTU uLCD-28PTU uVGA-III
  gen4-uLCD-24PT gen4-uLCD-28PT gen4-uLCD-32PT

  and other superseded display modules which support the ViSi environment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

- 4D Programming Cable / µUSB-PA5/µUSB-PA5-II for non-gen4 displays (uLCD-xxx)
- 4D Programming Cable & gen4-IB / gen4-PA / 4D-UPA, for gen-4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- 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

In Serial Picaso Getting Started - The SPE Application the user was introduced to how a 4D display module is configured as a serial slave device and to the basics of the **Serial Protocol**.

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 Goldelox Getting Started - The SPE Application

Serial Picaso Getting Started - The SPE Application

Serial Diablo16 Getting Started - The SPE Application

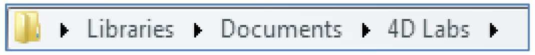
These application notes also introduce the user to the Serial Protocol thru the use of the Serial Commander.

Program the Arduino Host

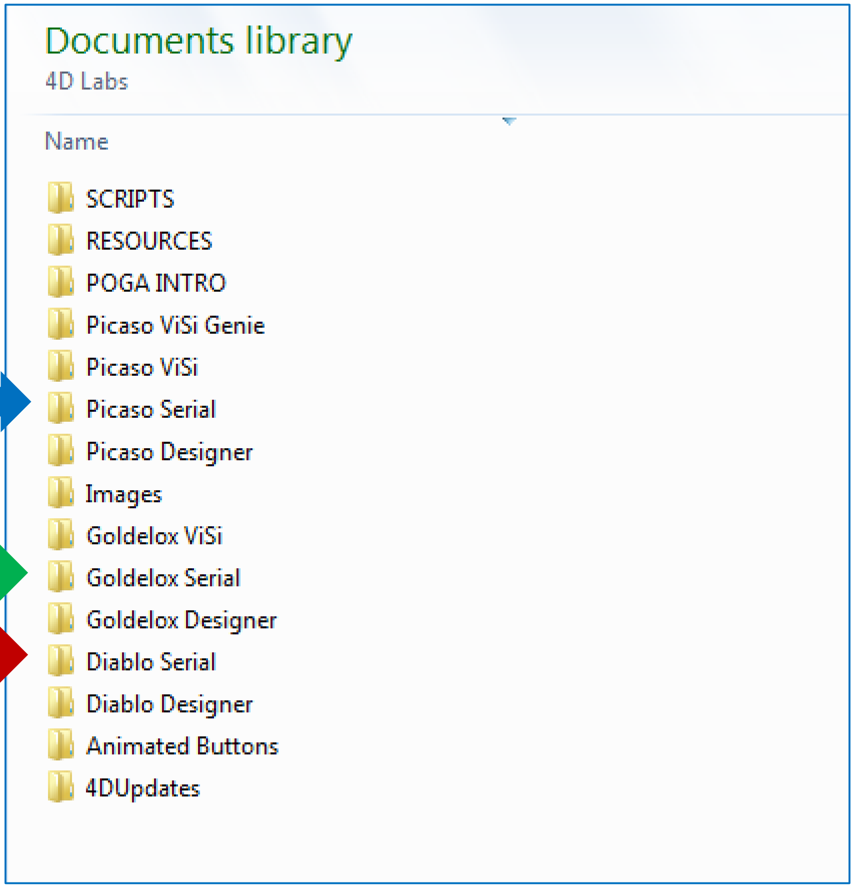
This section discusses the source code for the Arduino host for it to work with the display module. It is assumed that the user has a basic understanding of how the Arduino host works and how to program in the Arduino IDE. Inexperienced users may need to frequently refer to the Arduino website for more information.

Install the Library

The **Arduino-4D Serial Libraries** files, along with the **BigDemo** files, are copied to your PC during Workshop 4 installation. See the folder shown below (for Windows 8).



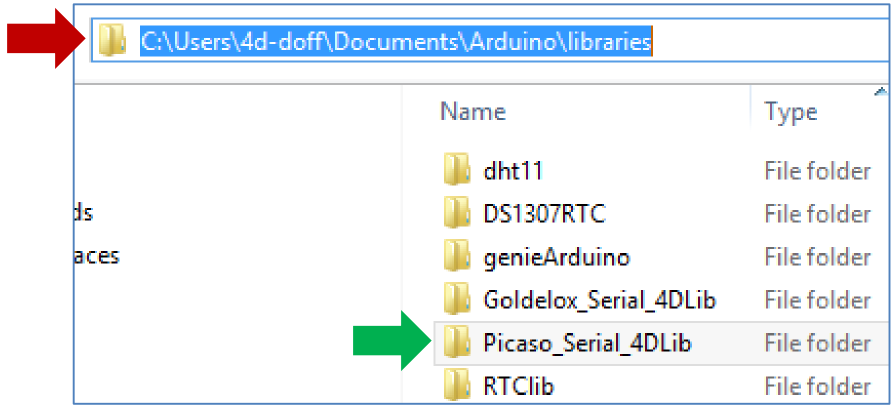
The **4D Labs** contain 3 Serial folder namely 'Picaso Serial', 'Goldelox Serial' and 'Diablo Serial'. Inside these folders are libraries for different microcontrollers/microprocessors.


Documents library
4D Labs
Name
SCRIPTS
RESOURCES
POGA INTRO
Picaso ViSi Genie
Picaso ViSi
Picaso Serial
Picaso Designer
Images
Goldelox ViSi
Goldelox Serial
Goldelox Designer
Diablo Serial
Diablo Designer
Animated Buttons
4DUpdates

Copythefolder**"Picaso_Serial_4DLib/Diablo_Serial_4DLib/Goldelox_Serial_4DLib"** to where additional Arduino libraries are saved. Here is a link to a tutorial on installing additional libraries in the Arduino IDE.

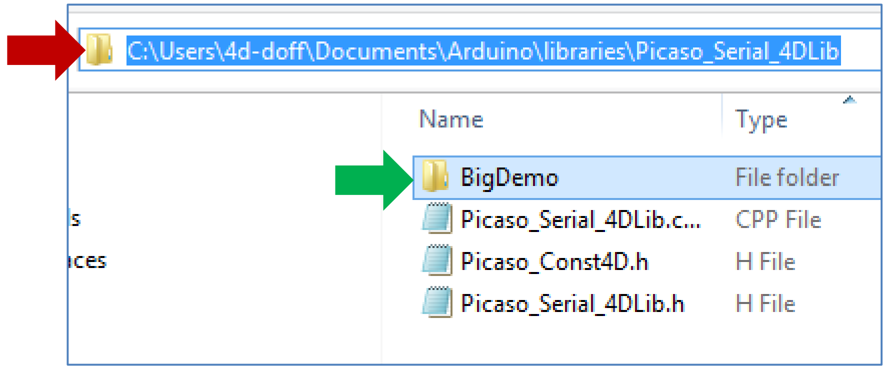
http://arduino.cc/en/Guide/Libraries

In Windows 8 for example, the library files will be saved here:


C:\Users\4d-doff\Documents\Arduino\libraries

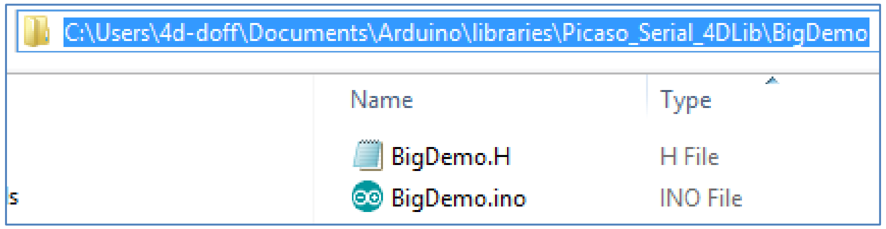
Name	Type
dht11	File folder
DS1307RTC	File folder
genieArduino	File folder
Goldelox_Serial_4DLib	File folder
Picaso_Serial_4DLib	File folder
RTClib	File folder

Also, create a copy of the folder **BigDemo** inside **Picaso_Serial_4DLib**.

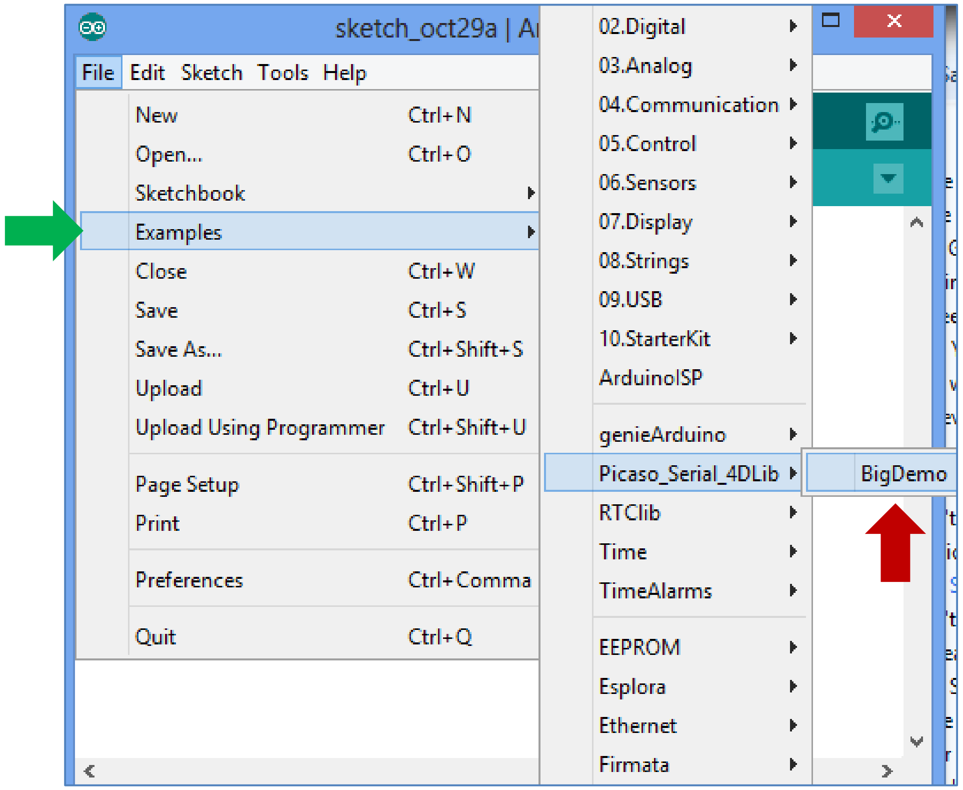

C:\Users\4d-doff\Documents\Arduino\libraries\Picaso_Serial_4DLib

Name	Type
BigDemo	File folder
Picaso_Serial_4DLib.c...	CPP File
Picaso_Const4D.h	H File
Picaso_Serial_4DLib.h	H File

The **BigDemo** folder contains the **BigDemo.ino** file, which demonstrates the use of all the serial commands. The sketch presented in this application note is a simplified version of the BigDemo sketch.

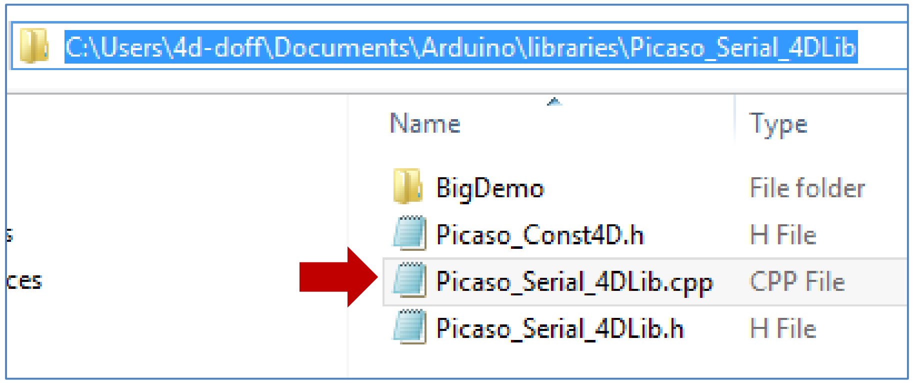

C:\Users\4d-doff\Documents\Arduino\libraries\Picaso_Serial_4DLib\BigDemo
Name
Type
BigDemo.H
H File
BigDemo.ino
INO File

Remember to restart the Arduino IDE after installing the library files. As a quick test, the **BigDemo** sketch should be accessible under the **File – Examples** menu.

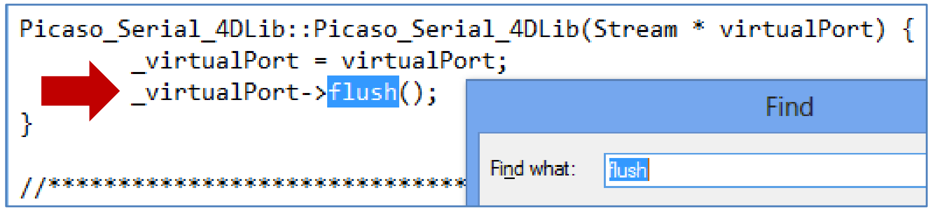


Modify the Library for the Arduino Due

If the host is an Arduino Due, one of the library files needs to be edited. For example open the file **"Picaso_Serial_4DLib.cpp"** (create a backup copy before editing).



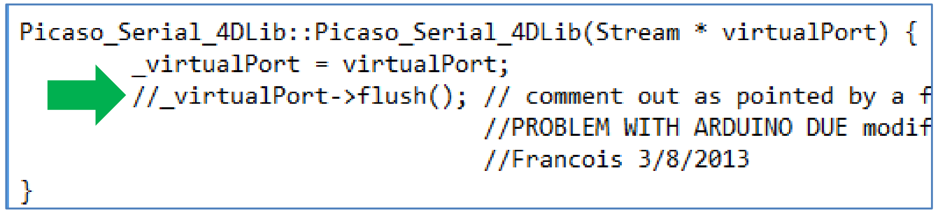
Find the line shown below.



```
Picaso_Serial_4DLib::Picaso_Serial_4DLib(Stream * virtualPort) {
        _virtualPort = virtualPort;
        _virtualPort->flush();
}

//******************************
```

Comment out the line so it becomes:



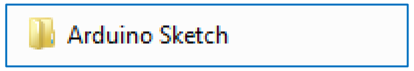
```
Picaso_Serial_4DLib::Picaso_Serial_4DLib(Stream * virtualPort) {
        _virtualPort = virtualPort;
        //_virtualPort->flush(); // comment out as pointed by a f
                                 //PROBLEM WITH ARDUINO DUE modif
                                 //Francois 3/8/2013
}
```

Save the file.

Note: This example is also applicable for Diablo Serial Library and Goldelox Serial Library.

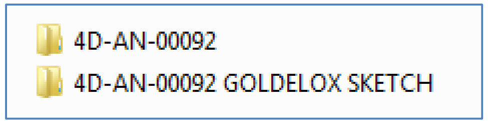
Open the Attached Arduino Sketch Files

There are two sketch files attached to this application note – a basic demo sketch **without** message logging and another version with message logging. Interested users can use the latter version for learning and debugging purposes. Since the serial monitor uses the port Serial0 for logging messages, either a software serial port or any of the other three hardware serial ports can be used for the display.



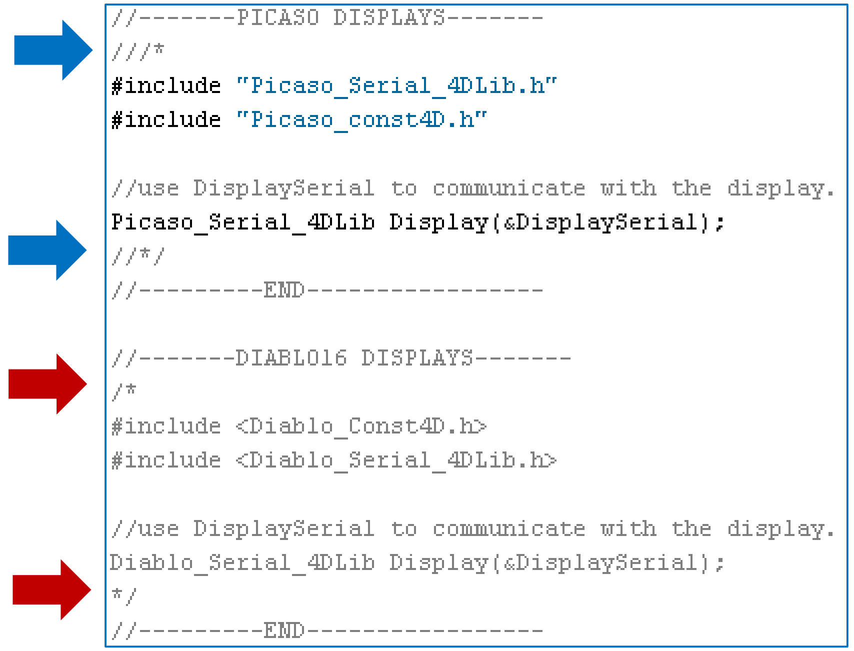
Understanding the Arduino Demo Sketch (w/o Message Logging)

Open, compile, and download to the Arduino host the attached sketch shown below. Note that comments have been added to the code for the benefit of the user. Additional explanations now follow.



Include the Library Files

For Picaso and Diablo Arduino Sketch:



```
//-------PICASO DISPLAYS-------
///*
#include "Picaso_Serial_4DLib.h"
#include "Picaso_const4D.h"

//use DisplaySerial to communicate with the display.
Picaso_Serial_4DLib Display(&DisplaySerial);
//*/
//---------END-----------------

//-------DIABLO16 DISPLAYS-------
/*
#include <Diablo_Const4D.h>
#include <Diablo_Serial_4DLib.h>

//use DisplaySerial to communicate with the display.
Diablo_Serial_4DLib Display(&DisplaySerial);
*/
//---------END-----------------
```

If Diablo16 Displays are going to be used, comment the lines that is pointed by the red arrows to include the Diablo Serial Library and then remove the comments '//' pointed by the blue arrows to comment the inclusion of the Picaso Serial Library.

Note: Goldelox example has a different Arduino sketch attached.

Define the Serial Port to be Used

Here **Serial0** will be used for communicating with the display module.

```
//1. Hardware Serial - choose a serial port
//-----------------------------------------
//default serial port for talking to the dis
#define DisplaySerial Serial
//#define DisplaySerial Serial1
//#define DisplaySerial Serial2
//#define DisplaySerial Serial3
//-----------------------------------------
```

To use a software serial port, comment out the line for the hardware port definition and uncomment the software serial port lines.

```
//1. Hardware Serial - choose a serial port
//-----------------------------------------
//default serial port for talking to the dis
//#define DisplaySerial Serial
//#define DisplaySerial Serial1
//#define DisplaySerial Serial2
//#define DisplaySerial Serial3
//-----------------------------------------
```

```
//2. Software Serial - set the desired pins
//-----------------------------------------
#include <SoftwareSerial.h>
SoftwareSerial DisplaySerial(10,11) ; // pi
//SoftwareSerial is not yet supported by the
//-----------------------------------------
```

If using the 4D Arduino Adaptor Shield and pins 10 and 11 are used as a software serial port for communicating with the display, jumpers J3 and J4 (of the 4D Arduino Adaptor Shield) need to be set accordingly. See the section **"Connect the 4D Display Module to the Arduino Host"** for more details.

The line below tells the library that the serial port referred to by **DisplaySerial** will be used for talking to the display.

```
//use DisplaySerial to communicate with the display.
Picaso_Serial_4DLib Display(&DisplaySerial);
```

The Acknowledgment Byte

When the display module receives a command from the host, it executes the command and sends back an acknowledgment (and reply bytes if necessary). It is very important that the Arduino host waits for the acknowledgment (and reply bytes) before sending another command. Section **2.4 Introduction and Guidelines to the Serial Protocol** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual emphasizes this point.

Commands should only be sent and their response received, before another command is sent. If two commands are sent before the first response is received, incorrect operation may follow.

The built-in high-level commands of the **Arduino-Picaso Serial Library** are coded such that they will automatically wait for an acknowledgment (and reply bytes if there are) from the display. The simple command for clearing the screen is taken as an example.

```
Display.gfx_Cls();//clear the screen
```

Section **5.2.1 Clear Screen** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual describes in detail the command for clearing the screen. Inside the library file **"Picaso_Serial_4DLib.cpp"**, the function **gfx_Cls( )** is defined as follows:

```
void Picaso_Serial_4DLib::gfx_Cls()
{
  _virtualPort->print((char)(F_gfx_Cls >> 8));
  _virtualPort->print((char)(F_gfx_Cls));
  GetAck();
}
```

Note that after sending off the command bytes, the routine now calls on the function **GetAck( )**, which will wait for the ACK byte from the display module. Many of the basic commands (i.e., those that do not necessarily require a specific reply from the display module besides the ACK byte) are coded in a manner similar to this.

Now the function **GetAck( )** is also defined in **Picaso_Serial_4DLib.cpp**. Essentially, **GetAck( )** waits for the ACK byte from the display for a certain period of time. If the ACK byte is received within the time limit, the function exits and the program goes back to the main loop. If nothing is received within the specified waiting time limit, **GetAck( )** calls on the error-handling routine, passing to it two arguments – an integer for indicating that a timeout has occurred and a character which holds an insignificant value. If a byte other than the ACK byte is received, **GetAck ( )** calls on the error handler, passing to it two arguments – an integer for indicating that a NAK condition has occurred and a character which holds the value of the invalid byte received.

The user now has the option of writing the routine for handling errors. The function **mycallback( )** of the demo sketch is an example of such a routine. One important thing to remember is that when the library calls on the error-handling routine, it will pass to it two arguments – an integer and a character. To let the library know which function to call when an error occurs, write the line shown below at the start of the program.

```
void setup() {
  //assign the function for handling errors
  Display.Callback4D = mycallback ;
```

The Error-handling Routine

The function **mycallback( )** will execute when an error occurs (i.e., the display has taken too long to respond or it has sent back a reply other than the expected ACK byte).

```
void mycallback(int ErrCode, unsigned char Errorbyte)
{
  // Pin 13 has an LED connected on most Arduino boar
  int led = 13;
  pinMode(led, OUTPUT);
  while(1)
  {
    digitalWrite(led, HIGH);   // turn the LED on (HI
    delay(200);               // wait for a second
    digitalWrite(led, LOW);    // turn the LED off by
    delay(200);               // wait for a second
  }
}
```

Note that **mycallback( )** will only blink the LED at pin 13 indefinitely. The user may also use this function to reset the display and re-establish communications. When an error occurs, it would be improper for the program to proceed further without resetting the display. Again,

Commands should only be sent and their response received, before another command is sent. If two commands are sent before the first response is received, incorrect operation may follow.

In the other version of this sketch (that which has a message logging feature), the error handler uses the integer and character arguments passed to it to specify the exact nature of the error.

Set the Timeout Limit

Set how long the host will wait for replies coming from the display by writing the line shown below at the start of the program.

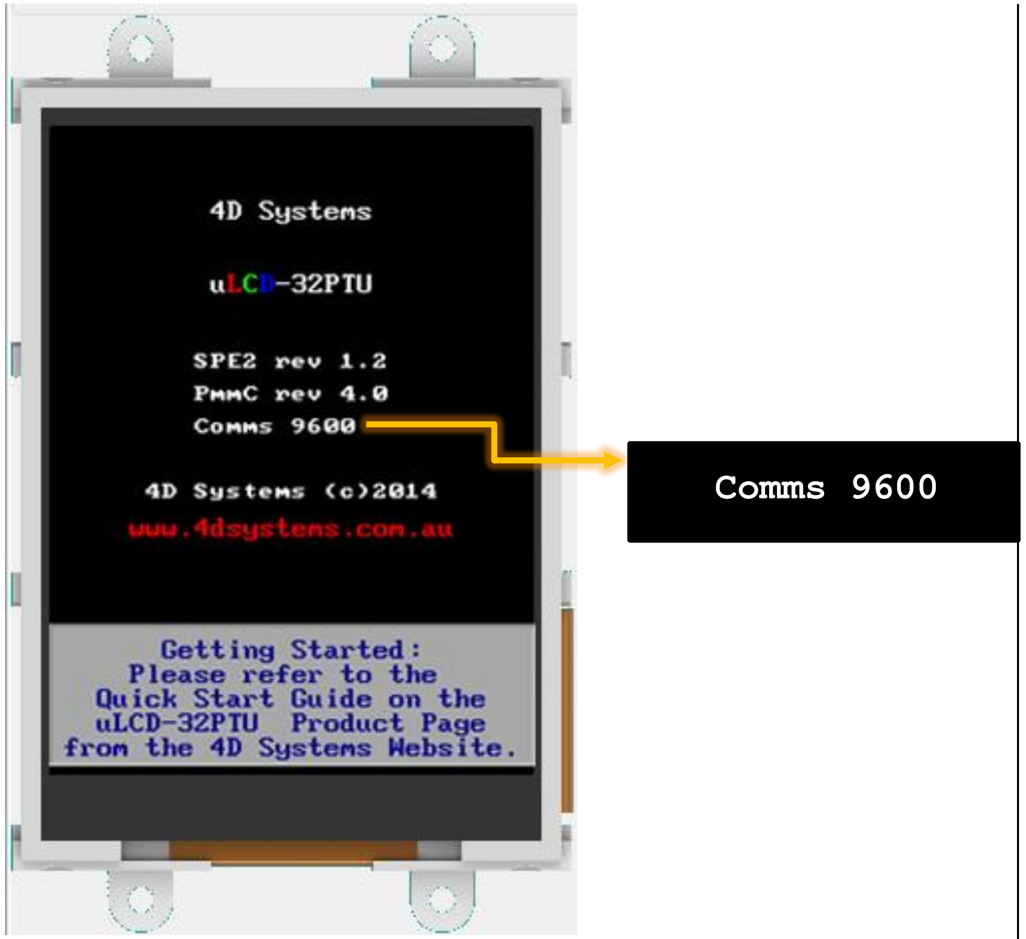
```
//5 second timeout on all commands
Display.TimeLimit4D   = 5000 ;
```

Set the Baud Rate

Communication with the display is set at 9600 bps.

```
//start
DisplaySerial.begin(9600) ;
```

Logically, the 4D display should also communicate with the Arduino host at the same baud rate. To check the baud rate of the display module check the baud rate indicated on the Serial Platform Environment (SPE) application splash screen.



4D Systems

uLCD-32PTU

SPE2 rev 1.2
PmmC rev 4.0
Comms 9600

4D Systems (c)2014
www.4dsystems.com.au

Getting Started:
Please refer to the
Quick Start Guide on the
uLCD-32PTU Product Page
from the 4D Systems Website.

Comms 9600

Reset the Arduino Host and the Display

To make resetting more convenient, the code below resets the display module when the program is restarted.

```
//Reset the Display (change D4 to D2 if you have o
pinMode(4, OUTPUT);  // Set D4 on Arduino to Outpu
digitalWrite(4, 1);  // Reset the Display via D4
delay(100);
digitalWrite(4, 0);  // unReset the Display via D4
```

If using the new 4D Arduino Adaptor Shield (Rev 2)

Note that the GPIO pin D4 of the Arduino host is used here for resetting the display. When using the new 4D Arduino Adaptor Shield (Rev 2.00 written on the PCB), make sure that pin RES is connected to pin AR in jumper J1. See the section **"Connect the 4D Display Module to the Arduino Host"**. If using the old 4D Arduino Adaptor Shield (Rev 1), simply change the code above. Use pin 2 instead of pin 4.

```
//Reset the Display (change D4 to D2 if you have o
pinMode(2, OUTPUT);  // Set D4 on Arduino to Output
digitalWrite(2, 1);  // Reset the Display via D4
delay(100);
digitalWrite(2, 0);  // unReset the Display via D4
```

If using the old 4D Arduino Adaptor Shield (Rev 1)

If using jumper connecting wires, connect the RESET pin of the display module to the D4 pin of the Arduino with a 1kohm series resistor in between (see the section **"Connect the 4D Display Module to the Arduino Host"**), and modify the code as shown below.

```
//Reset the Display (change D4 to D2 if you have or
pinMode(4, OUTPUT);  // Set D4 on Arduino to Output
digitalWrite(4, 0);  // Reset the Display via D4
delay(100);
digitalWrite(4, 1);  // unReset the Display via D4
```

If using jumper wires

Note that the logic state for resetting the display is now 0 instead of 1. This is because the display module's RESET pin is directly connected to D4 via a 1kohm resistor. If using a 4D Arduino Adaptor Shield, the display module's RESET pin is switched by the D4 pin via a transistor.

Let the Display Start Up

The five second – delay below waits for the display module to start up.

```
delay (5000);          //let the display start up
```

In section **2.5 Power-Up and Reset** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual, it says:

When the PICASO Display Module comes out of a power-up or external reset, a sequence of events is executed internally. The user should wait at least 3 seconds for the start-up to take place before attempting to communicate with the module.

Set the Screen Orientation

This is the first command sent to the display module.

```
Display.gfx_ScreenMode(LANDSCAPE_R);
```

It sets the orientation of the display to **"reversed landscape"**. Refer to section **5.2.34/5.2.24(GOLDELOX) Screen Mode** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual for more information. Note that the function returns the previous screen mode value. Thus, the user can write:

```
orientationPrev = Display.gfx_ScreenMode(LANDSCAPE_R);
```

Clear the Screen

```
Display.gfx_Cls();//clear the screen
```

Section **5.2.1 Clear Screen** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual describes in detail the command for clearing the screen.

uSD Card Mount Routine

```
//uSD card mount 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 routine is optional but included in the program for practice. The program won't go further unless a uSD card is inserted in the LCD. This routine can be commented out and won't affect the final output.

Send a String

```
Display.putstr("HELLO WORLD\r");
```

For more information, refer to section **5.1.3 Put String** of the Picaso/Diablo/Goldelox Serial Command Set Reference Manual. Note that the function returns the length of the string printed.

Understanding the Arduino Demo Sketch (with Message Logging)

In this section, the Serial Monitor of the Arduino IDE is used to send messages to the PC. This is an effective method of debugging a serial program and/or getting acquainted with the **Arduino Serial Library**. For the Arduino Uno, the user can use a software serial port only to talk to the display since the hardware port Serial0 is used exclusively by the Serial Monitor. The following images show the results for testing the basic demo sketch files at 9600 bps on three Arduino boards. The display module was a uLCD-32PTU.

	Basic Demo with out message logging				
	Serial0	Serial1	Serial2	Serial3	Sofware (10, 11)
Uno	OK	NA	NA	NA	OK
Mega	OK	OK	OK	OK	OK
Due	OK	OK	OK	OK	NA

	Basic Demo with message logging				
	Serial Monitor	Serial1	Serial2	Serial3	Sofware (10, 11)
Uno	OK	NA	NA	NA	OK
Mega	OK	OK	OK	OK	OK
Due	OK	OK	OK	OK	NA

Enable Message Logging

The line

```
//enable message logging
#define LOG_MESSAGES
```

enables message logging. If message logging is enabled, Serial0 is now used to communicate with the Serial Monitor.

```
#ifdef LOG_MESSAGES
  #define HWLOGGING Serial
```

To disable message logging, simply comment out line 41 of the sketch as shown below.

```
//enable message logging
//#define LOG_MESSAGES
```

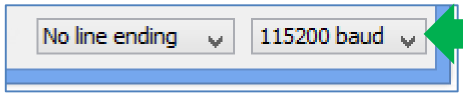
All message-logging-related lines throughout the sketch will now be disregarded.

Set the Baud Rate

Communication with the Serial Monitor is set at 115200 bps.

```
#ifdef LOG_MESSAGES
  HWLOGGING.begin(115200);
  HWLOGGING.println(F("\nArduino-4D Display Serial
#endif
```

Make sure that the Serial Monitor is configured properly.


No line ending
115200 baud

Displaying Returned Values

Note that when the display module receives the put string command and parameters, it prints the string and sends back an acknowledgment byte and two more bytes – the MSB and LSB values of the length of the string just printed. Thus the function **Display.putstr( )** returns the string length. Here it is stored in a variable.

```
stringLength[0] = Display.putstr("Hello Picaso!\n");
```

The Error-handling Routine

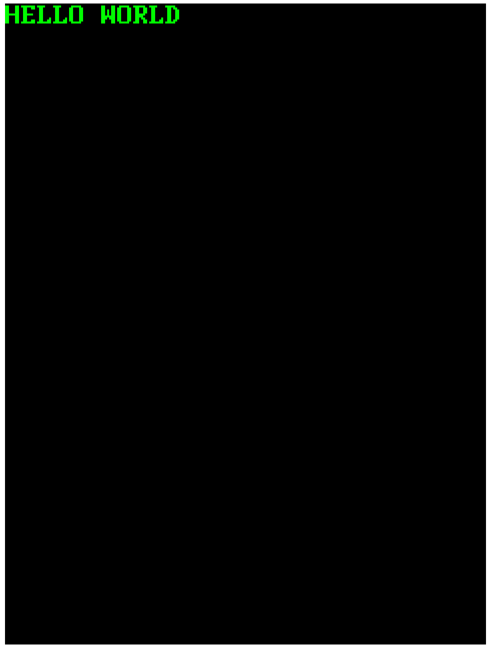
```
void mycallback(int ErrCode, unsigned char Errorbyte)
{
#ifdef LOG_MESSAGES
  const char *Error4DText[] = {"OK\0", "Timeout\0", "
  HWLOGGING.print(F("Serial 4D Library reports an err
  HWLOGGING.print(Error4DText[ErrCode]) ;
  HWLOGGING.print(F("\n"));

  if (ErrCode == Err4D_NAK)
  {
    HWLOGGING.print(F("returned data = ")) ;
    HWLOGGING.print(Errorbyte) ;
  }
  else
    HWLOGGING.println(F("")) ;

  HWLOGGING.println(F("Program cannot proceed further
  while(1) ; // you can return here, or you can loop
```

The error-handler above is an improved and complete version of the **mycallback ( )** routine. It uses the arguments passed to it to inform the user of the exact nature of the error. Take note that aside from the **GetAck( )** function, there are other functions in the **Arduino Serial Library** that calls on the error handler when an error occurs. See the file **"Picaso/Diablo/Goldelox_Serial_4DLib.cpp"** for more information.

Final Output



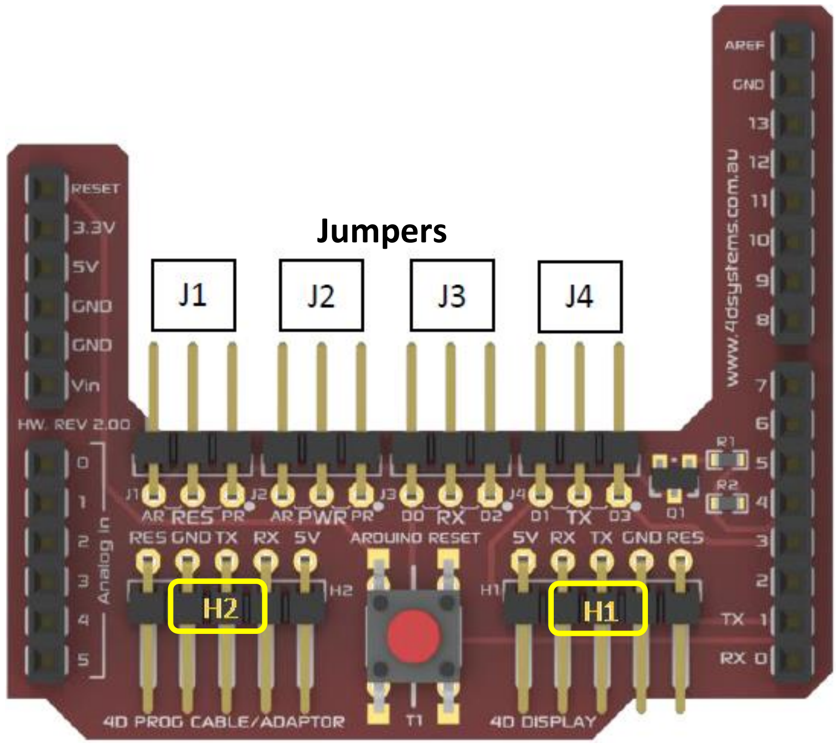
```
HELLO WORLD
```

Connect the 4D Display Module to the Arduino Host

This section discusses several ways of connecting the display module to the Arduino host. The user has the option of using a 4D Arduino Adaptor Shield (there are two versions of this – the old and the new) or jumper wires.

Using the New 4D Arduino Adaptor Shield (Rev 2.00)

Definition of Jumpers and Headers



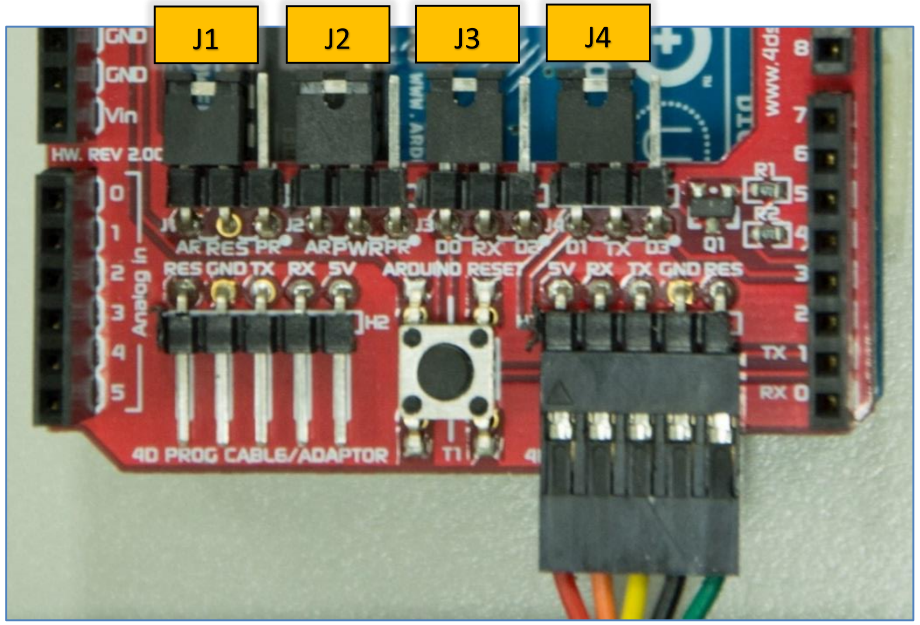
The 5-way cable coming from the display should be connected to **H1**. When the Arduino host cannot supply enough power to the display, the display can be powered separately thru **H2** (jumper **J2** should be configured accordingly).

J1 is for choosing which pin resets the display – either the RES pin of H2 or pin D4 of the Arduino host. **J2** is for choosing the power supply source for the display – either the Arduino host or the programming module connected to H2 (if the Arduino host power supply is inadequate). The middle pin of **J3**, RX, goes to the TX pin of the display and must be tapped to the correct RX pin of the Arduino host. The middle pin of **J4**, TX, goes to the RX pin of the display and must be tapped to the correct TX pin of the Arduino host.

Default Jumper Settings

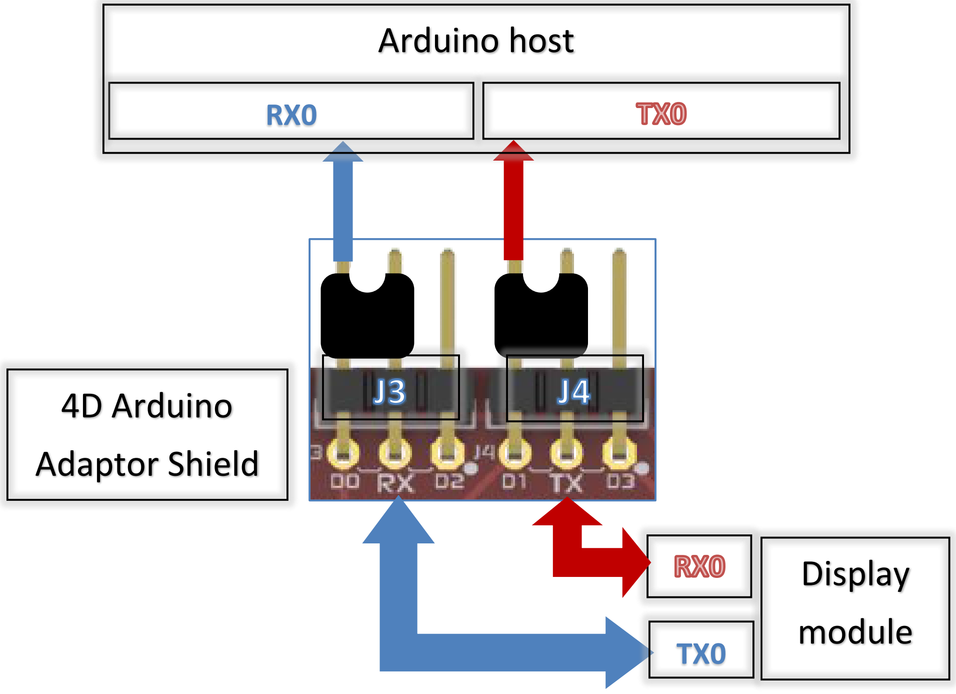
The image on the right column shows the default settings for jumpers J1 to J4.

- Pin D4 of the Arduino host resets the display (J1 shorts pins RES and AR).
- The Arduino host powers the display (J2 shorts pins PWR and AR).
- The Arduino host talks to the display thru port Serial0.



Default Settings for J3 and J4

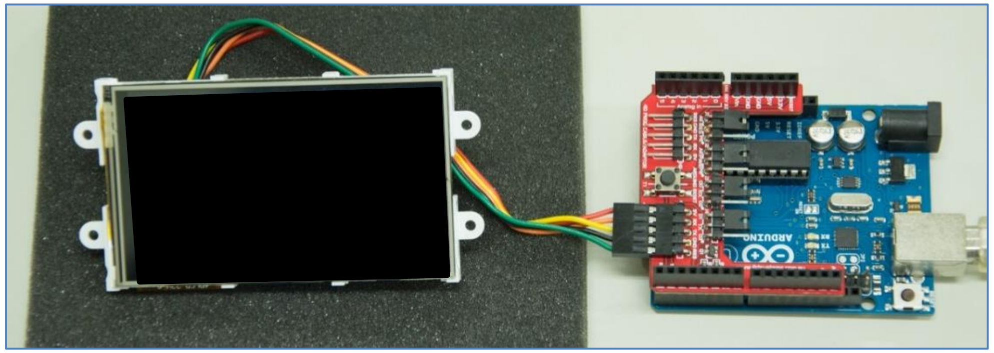
Jumpers J3 and J4 are configured, by default, to connect RX (TX0 of the display module) to D0 (RX0 of the Arduino) and TX (RX0 of the display module) to D1 (TX0 of the Arduino). Communication in this case is thru Serial0 of the Arduino host and COM0 of the display module.



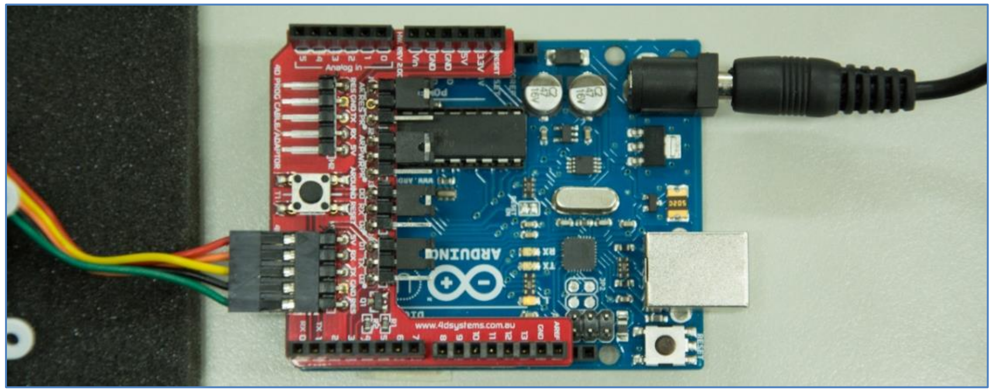
The Arduino Host Powers the Display

The following are images wherein the display module is powered by the Arduino host. Note that the power supply must be able to provide enough current for both the display module and the Arduino host. Refer to your display module's datasheet for the specified supply current.

Using the USB cable (the Arduino host powers the display):

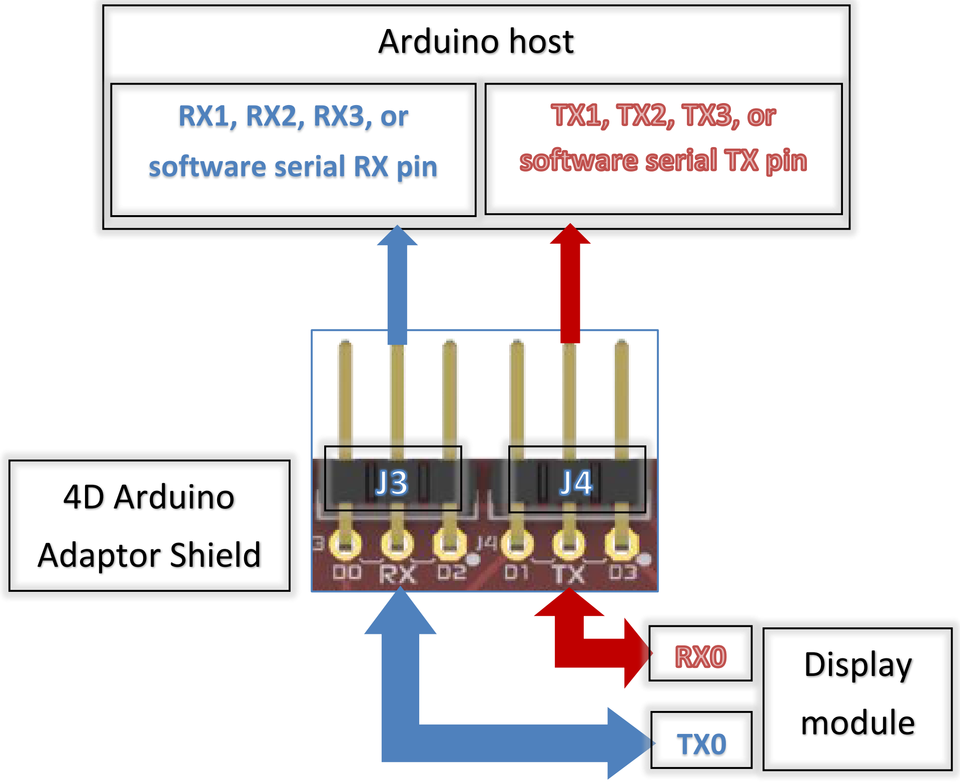


Using the jack(the Arduino host powers the display):



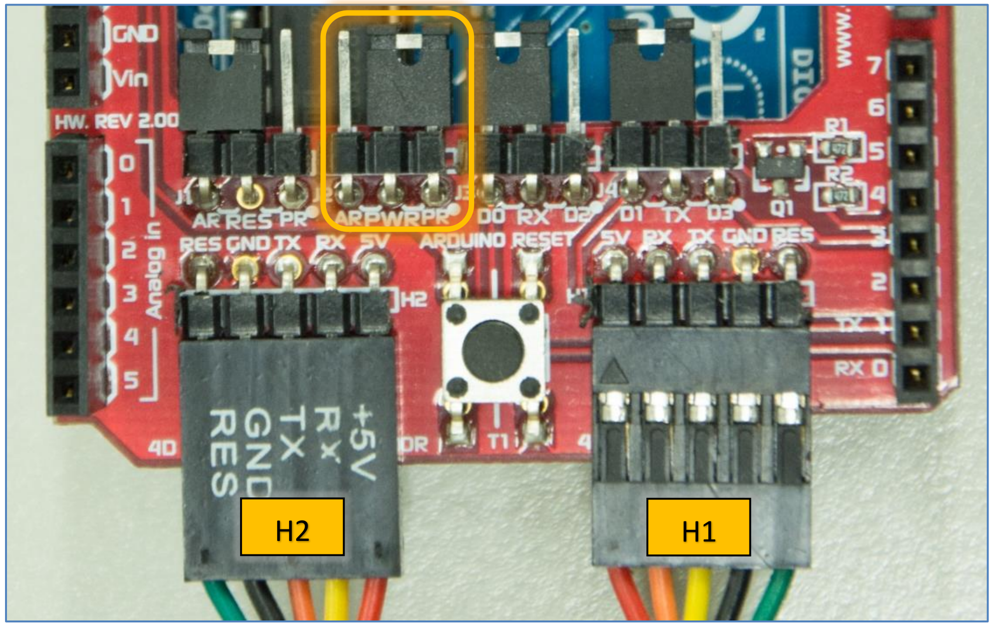
Change the Arduino Host Serial Port

To use the other hardware serial ports of the Mega or Due, remove the jumper connectors of J3 and J4 and connect the display TX0 and RX0 pins to the desired Arduino serial port TX and RX pins using jumper wires.


Arduino host
RX1, RX2, RX3, or software serial RX pin
TX1, TX2, TX3, or software serial TX pin
4D Arduino Adaptor Shield
J3
J4
D0 RX D2
D1 TX D3
RX0
TX0
Display module

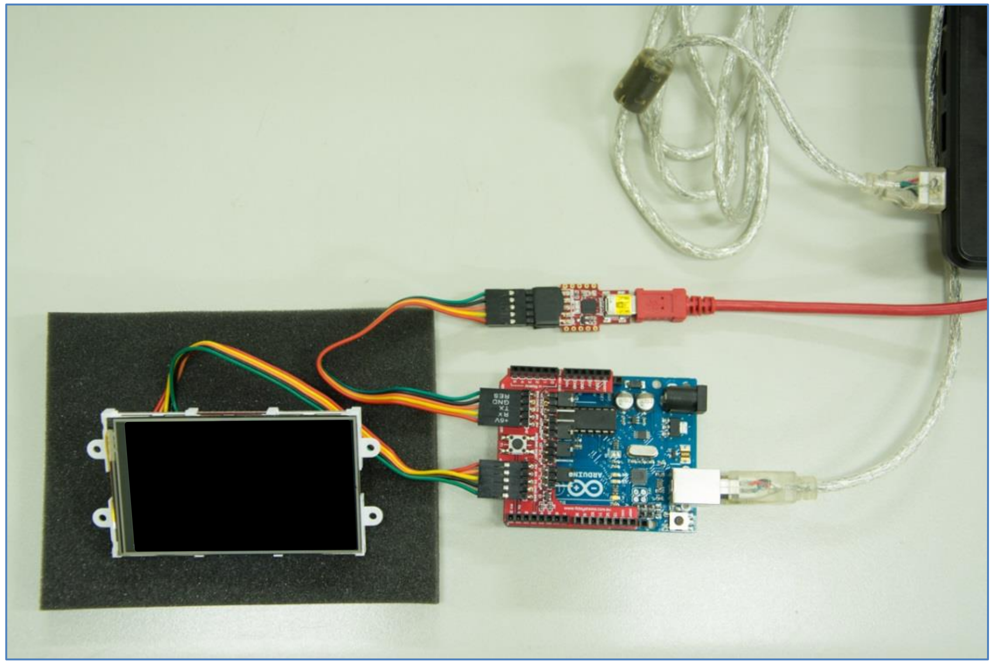
Power the Arduino Host and the Display Separately

If the display requires a higher current to operate (the uLCD-70DT for instance), it is not advisable to power it off the 5V out of the Arduino host. To power the display separately from the Arduino board, set J2 as shown below. Power will then be supplied to the display thru H2.



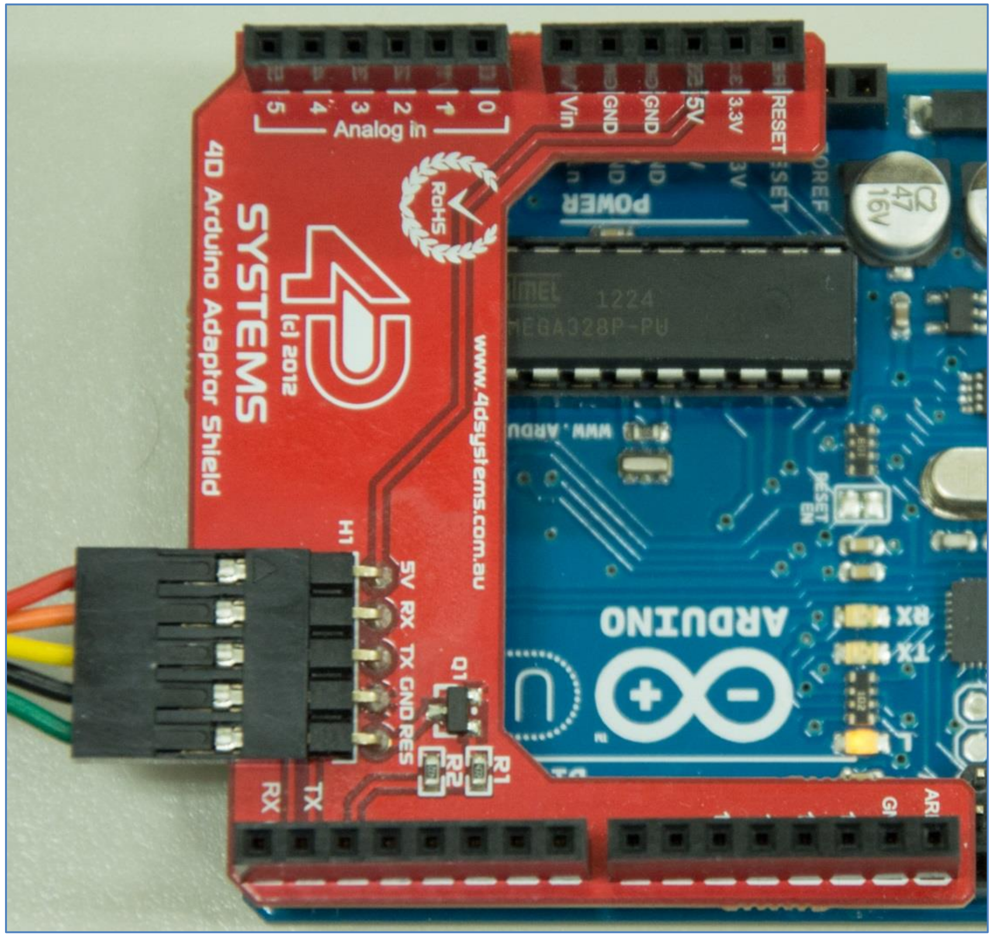
H2 is for the 4D USB Programming Cable or µUSB-PA5 (power supply source), and H1 is for the display module. The following image shows how the Arduino host and the 4D display are connected when they are powered separately.

Complete setup (host and display are powered separately):



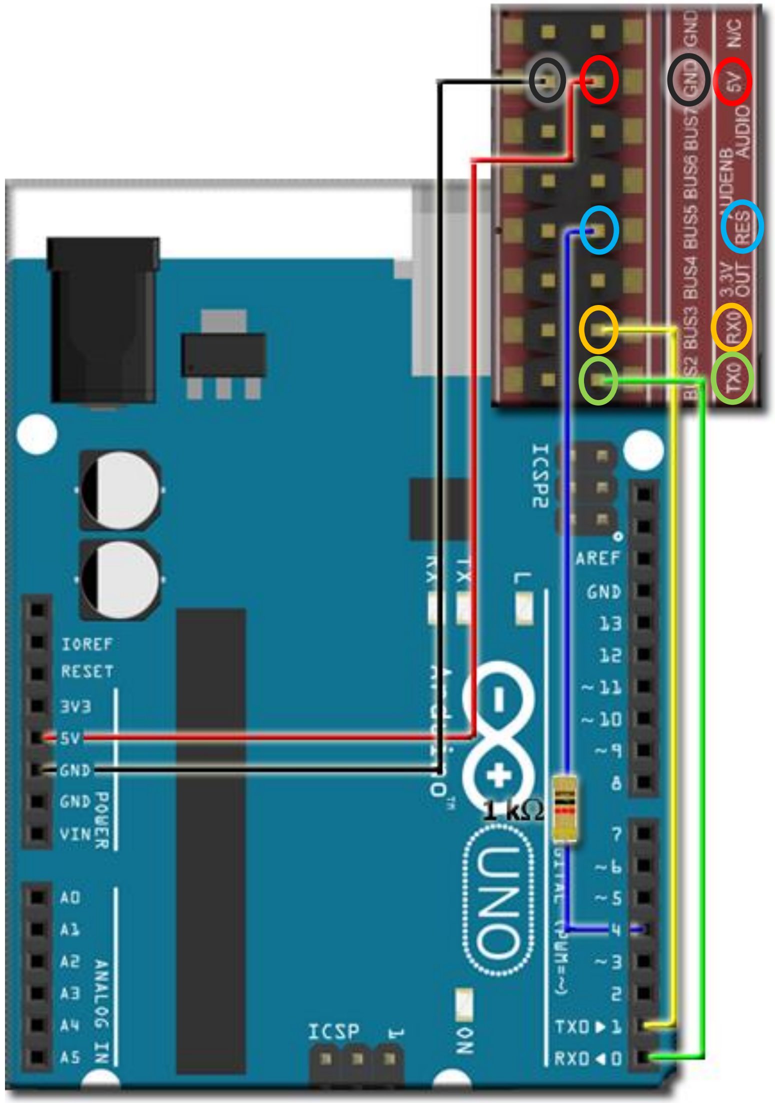
Note that the display module cannot be programmed thru the µUSB-PA5 in this setup since H2 transfers power only. Before programming the display module, disconnect it first from the Arduino Adaptor Shield. Likewise, before programming the Arduino host, make sure that it is not connected to the display module. Do this when the communication is thru **Serial0** (Arduino host) and **COM0** (4D display). Always double check the orientation of the connections.

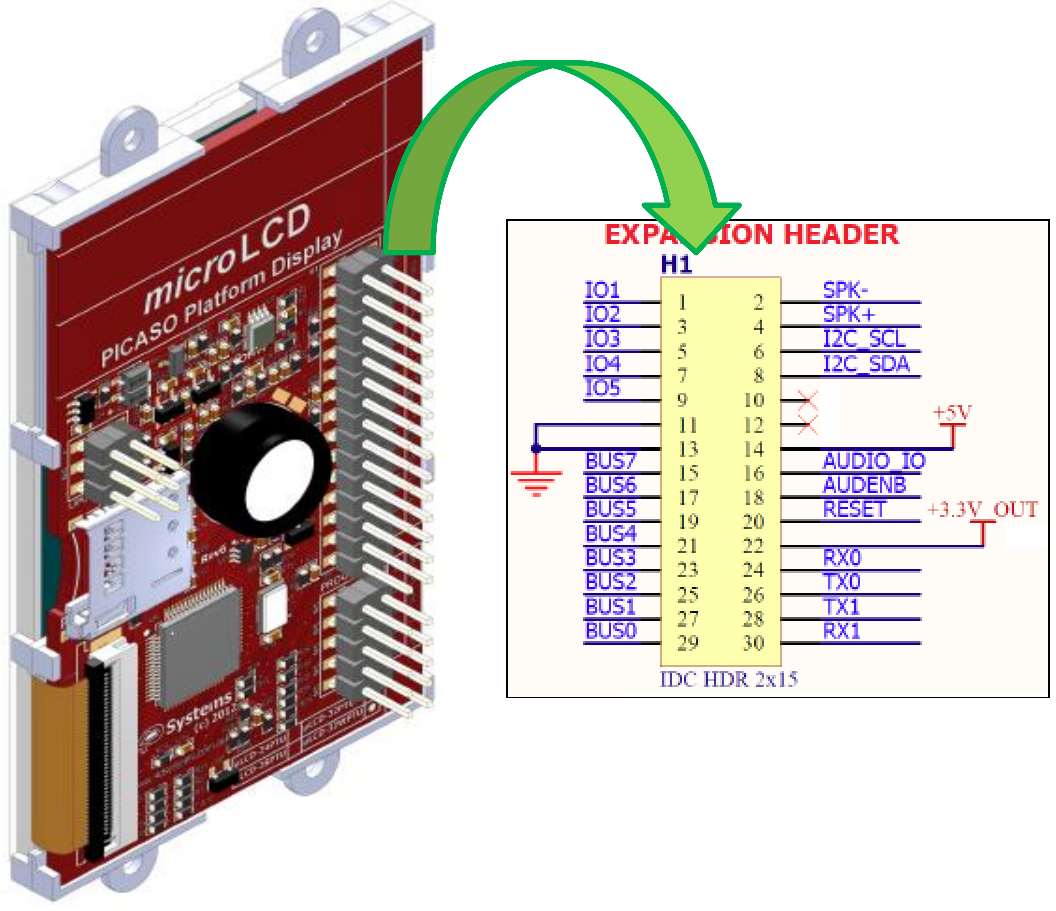
Using the Old 4D Arduino Adaptor Shield (Rev 1)



The old 4D Arduino Adaptor Shield (Rev1) uses digital pin **D2** for resetting the display. The reset routine of the Arduino sketch must be modified accordingly.

Connection Using Jumper Wires

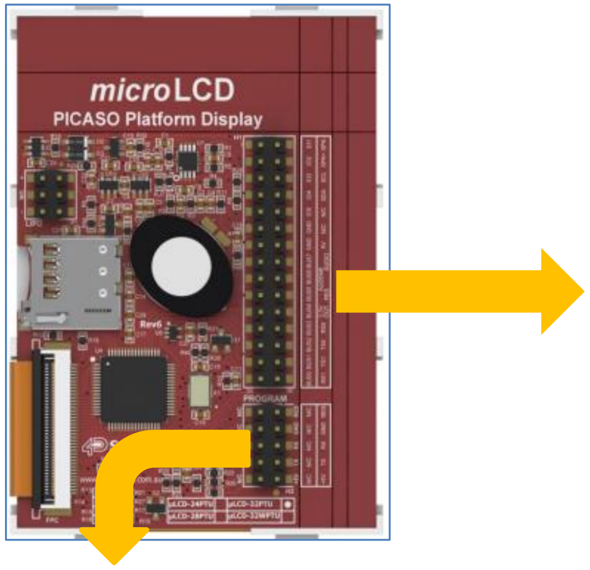


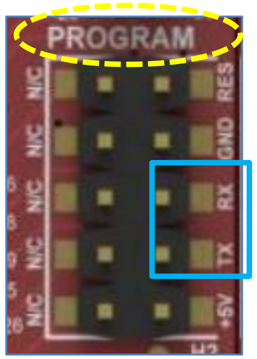


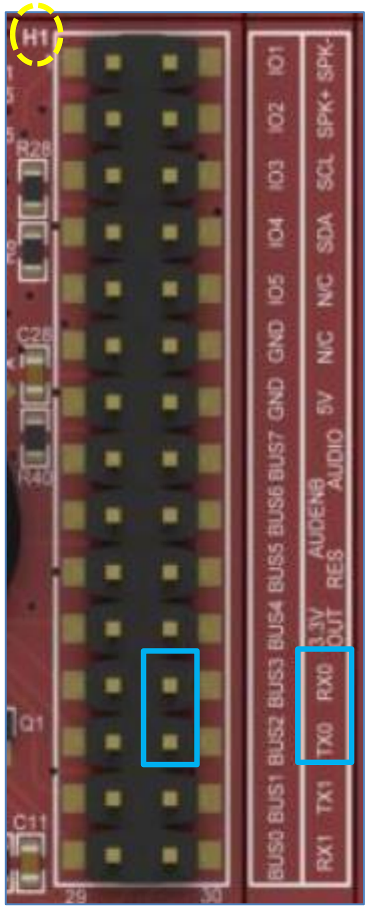
Note that the display here is powered off the **5V out** of the Arduino board. Pin **D4** of the host will also reset the display (logic of the reset routine must be inverted). Connect the **5V** and **GND** pins of the display to an external **5V** power supply source if a separate supply is needed. The reset pin, **RES**, of the display can also be connected to another GPIO pin of the Arduino host and the sketch can be modified accordingly.

Changing the Serial Port of the Genie Program

A ViSi-Genie program uses the serial port COM0 by default. This is also the serial port through which the display is programmed by Workshop. The datasheet for the uLCD-32PTU, for example, shows the H1 I/0 Expansion header and the programming header.

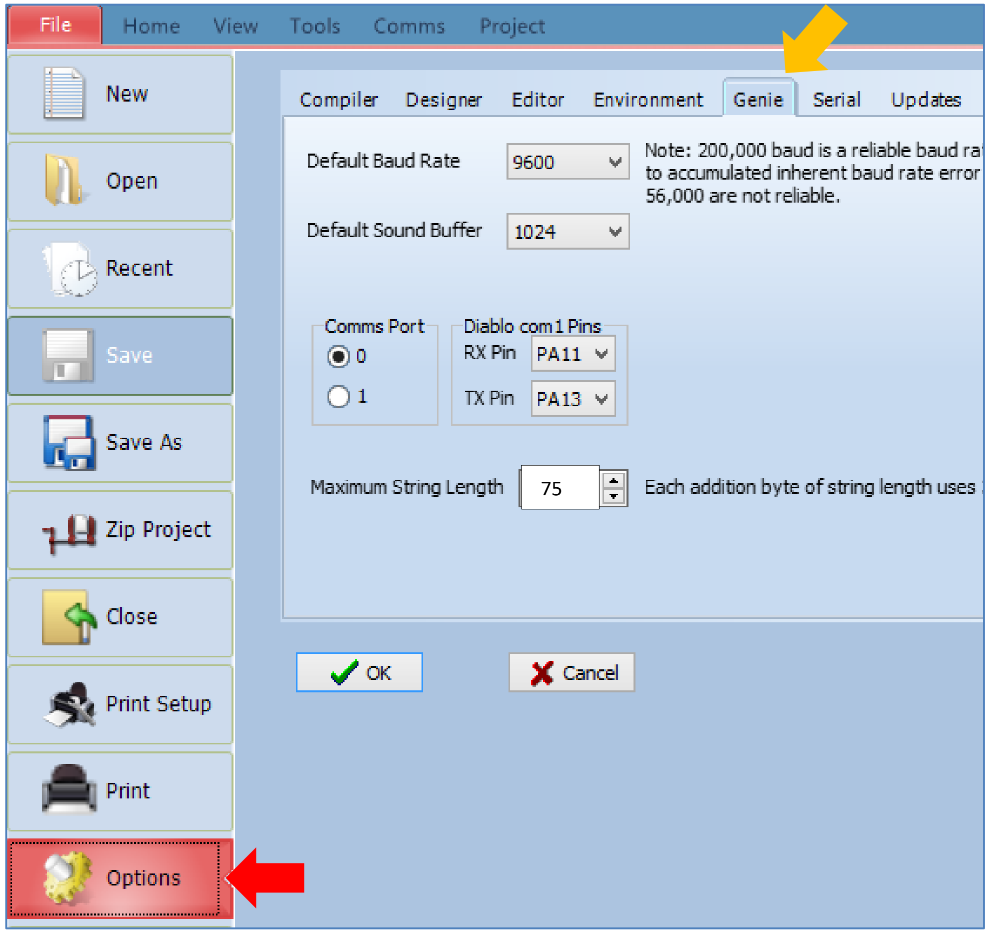






As the reader may have already perceived, the TX and RX pins on the programming header are the same pins as TX0 and RX0 on the H1 I/O expansion header. In Workshop it is possible to change the serial port being used by a ViSi-Genie program. Instructions for doing this are as follows.

Under the **File** menu, select **Options** then select the **Genie** tab.



Default Baud Rate: 9600

Note: 200,000 baud is a reliable baud ra to accumulated inherent baud rate error 56,000 are not reliable.

Default Sound Buffer: 1024

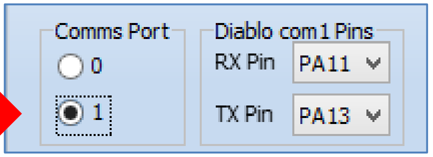
Comms Port: ● 0 ○ 1

Diablo com1 Pins: RX Pin PA11, TX Pin PA13

Maximum String Length: 75

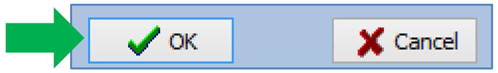
Each addition byte of string length uses

For Picaso displays there are only two available serial ports – COM0 and COM1. To use COM1, click on the button next to it then click OK.

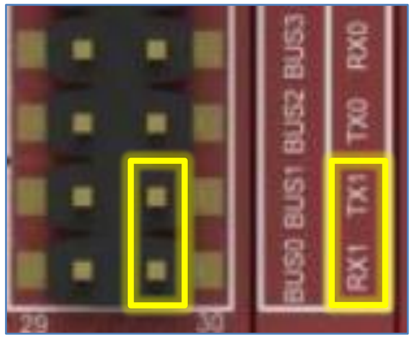


Comms Port: ○ 0 ● 1

Diablo com1 Pins: RX Pin PA11, TX Pin PA13



Compile and download the program to the display. All subsequent ViSi-Genie programs will now use COM1. Also, the TX and RX pins of the host shall now be connected to the RX and TX pins of COM1 instead of COM0.

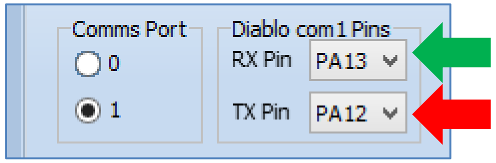


The Diablo16 processor has four serial ports – COM0, COM1, COM2, and COM3. The TX and RX pins of COM0 are fixed and are used for programming the processor. Again, COM0 is also the default serial port used by a ViSi-Genie program. The TX and RX pins for COM1, COM2, and COM3, on the other hand, are 'mappable' – that is, they can be configured to be 'mapped' out to any (but not all) of the GPIO pins. The table below shows the GPIO pins that can be used as TX and RX pins for COM1, COM2, and COM3. This table is taken from the Diablo16 datasheet.

DIABLO16 Serial TTL Comm Port Configuration Options

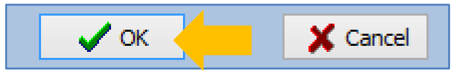
	TX1	RX1	TX2	RX2	TX3	RX3
PA0		✓		✓		✓
PA1	✓	✓	✓	✓	✓	✓
PA2		✓		✓		✓
PA3	✓	✓	✓	✓	✓	✓
PA4	✓	✓	✓	✓	✓	✓
PA5	✓	✓	✓	✓	✓	✓
PA6	✓	✓	✓	✓	✓	✓
PA7	✓	✓	✓	✓	✓	✓
PA8	✓	✓	✓	✓	✓	✓
PA9	✓	✓	✓	✓	✓	✓
PA10		✓		✓		✓
PA11		✓		✓		✓
PA12	✓	✓	✓	✓	✓	✓
PA13	✓	✓	✓	✓	✓	✓
PA14						
PA15						

Workshop, however, only provides the option of using COM1 as an alternative to COM0. To use the GPIO pins PA13 and PA12 as RX and TX pins respectively, specify them under **Diablo com1 Pins** then click OK.

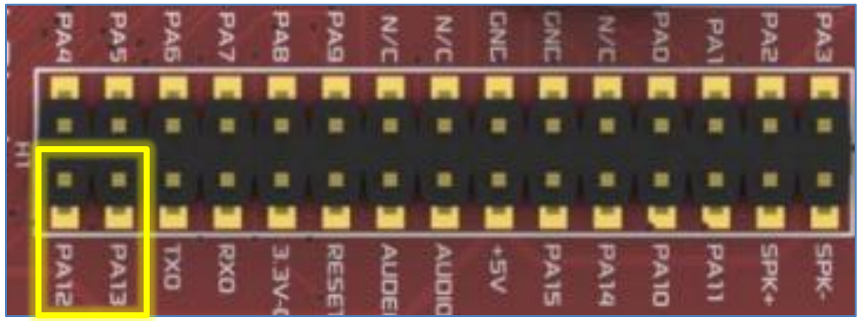


Comms Port: 0 [✓ 1]

Diablo com1 Pins
RX Pin: PA13
TX Pin: PA12



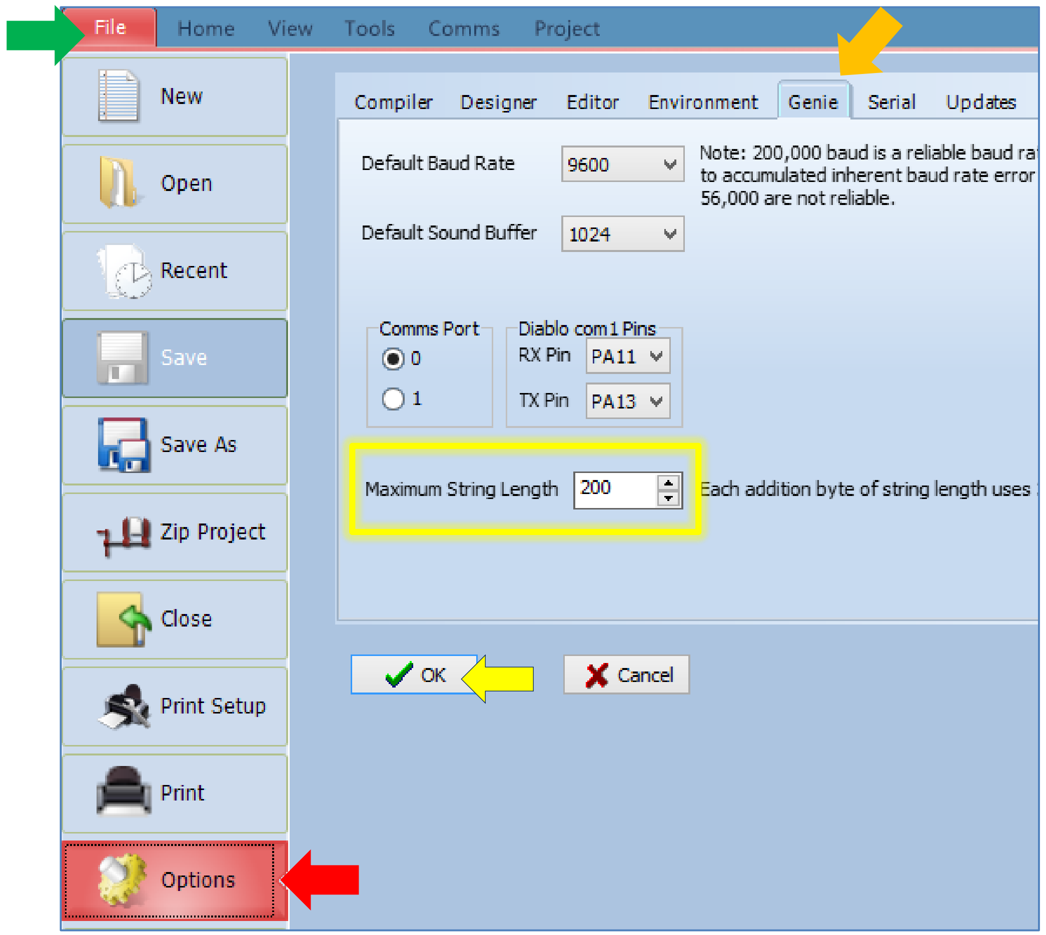
Compile and download the program to the display. All subsequent ViSi-Genie programs with a Diablo16 target display will now use COM1 with the specified TX and RX pins. Also, the TX and RX pins of the host shall now be connected to the specified RX and TX pins of COM1. If using a uLCD-35DT for example,



Consult the datasheet of your display for more information.

Changing the Maximum String Length

The host can dynamically write to the strings object of a ViSi-Genie program. The default maximum length of a character array that can be dynamically written to a strings object is 75 characters (excluding the overhead bytes). Worskhop provides an option for increasing this limit.
Under the **File** menu, select **Options** then select the **Genie** tab. Here the maximum length is set to 200 characters. Click OK.



Default Baud Rate 9600

Note: 200,000 baud is a reliable baud ra
to accumulated inherent baud rate error
56,000 are not reliable.

Default Sound Buffer 1024

Comms Port
(●) 0
( ) 1

Diablo com1 Pins
RX Pin PA11
TX Pin PA13

Maximum String Length 200

Each addition byte of string length uses

Compile and download the program to the display. All subsequent ViSi-Genie programs will now have this configuration.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability