



ViSi-Genie Connecting a 4D Display to an Arduino Host

DOCUMENT DATE: 15th JANUARY 2019
DOCUMENT REVISION: 1.02



Description

This Application Note explores the possibilities provided by the ViSi-Genie environment in Workshop to work with an Arduino host. In this example, the host is an AVR ATmega328 microcontroller-based 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 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and other superseded modules which support the ViSi Genie environment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](#)

- [4D Programming Cable / \$\mu\$ USB-PA5/uUSBPA5-II for non-gen4 displays \(uLCD-xxx\)](#)
- [4D Programming Cable & gen4-IB / 4D-UPA / gen4-PA for gen4 displays \(gen4-uLCD-xxx\)](#)
- [micro-SD \(\$\mu\$ SD\)](#) 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

Description	2	Interrogate the Display for the Status of the Slider	17
Content	3	REPORT_EVENT vs. REPORT_OBJ	17
Application Overview	4	Program the Arduino Host	18
Setup Procedure	6	<i>Download and Install the ViSi-Genie-Arduino Library</i>	18
Create a New Project	6	<i>Understanding the Arduino Sketch Demo</i>	19
Design the Project	7	Open a Serial Port and Set the Baud Rate	20
<i>Add a Cool Gauge Object</i>	7	genieAttachEventHandler()	22
<i>Naming of Objects</i>	8	Reset the Arduino Host and the Display	22
<i>Add a Text String Object</i>	8	Set the Screen Contrast	23
<i>Add a Slider Object</i>	10	Send a Text String	23
<i>Report Event</i>	11	The Main Loop	23
<i>Add a LED Digits Object</i>	12	Receiving Data from the Display	24
<i>Add a User LED Object</i>	13	The Use of a Non-blocking Delay	24
<i>Add a Static Text Object</i>	13	How to Change the Status of an Object	26
Build and Upload the Project	14	How to Know the Status of an Object	27
Identify the Messages	14	The User's Event Handler	27
<i>Use the GTX Tool to Analyse the Messages</i>	14	Connect the 4D Display Module to the Arduino Host	28
Launch the GTX Tool	14	<i>Using the New 4D Arduino Adaptor Shield (Rev 2.00)</i>	28
<i>The Slider Object</i>	15	Definition of Jumpers and Headers	28
Change the Status of the Slider	15	Default Jumper Settings	29
Message from a Slider	16	Change the Arduino Host Serial Port	30
		Power the Arduino Host and the Display Separately	31
		<i>Using the Old 4D Arduino Adaptor Shield (Rev 1)</i>	32

Connection Using Jumper Wires 33

Changing the Serial Port of the Genie Program..... 34

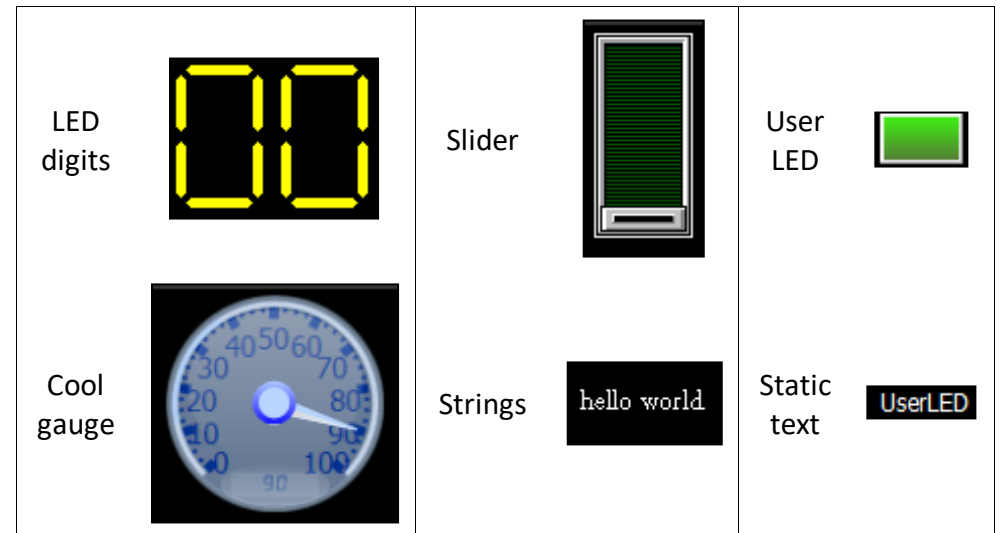
Changing the Maximum String Length..... 36

Proprietary Information 38

Disclaimer of Warranties & Limitation of Liability 38

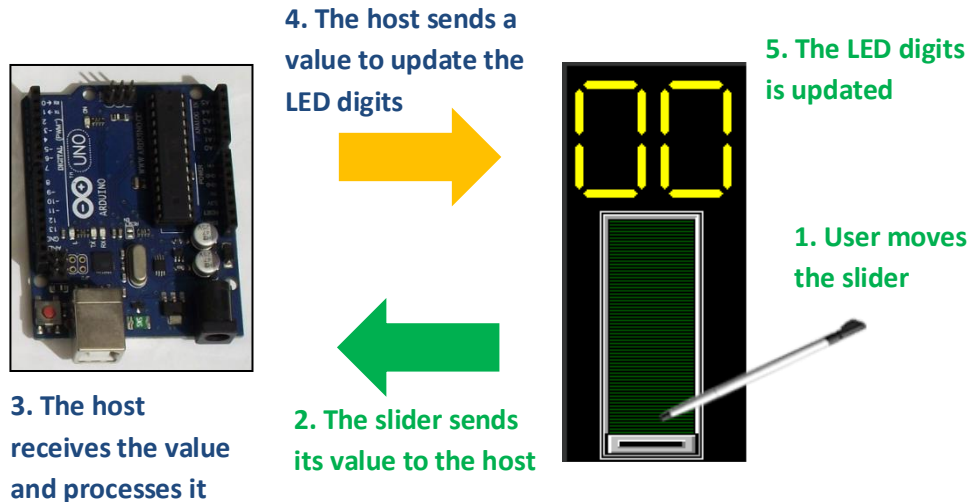
Application Overview

It is often difficult to design a graphical display without being able to see the immediate results of the application code. ViSi-Genie is the perfect software tool that allows users to see the instant results of their desired graphical layout with this large selection of gauges and meters (called widgets) that can simply be dragged and dropped onto the simulated module display. The following are examples of widgets or objects used in this application note.



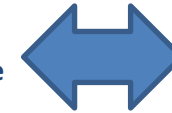
This application note shows how to create a ViSi Genie program and how to use the ViSi Genie library for the Arduino IDE. To achieve these objectives, a simple project is developed. This consists of a 4D Picaso module displaying six objects – a LED digits, a slider, a cool gauge, a string, a user LED, and a

static text (label). Several of these objects interact with an Arduino host in a manner illustrated below.



The user LED changes its state according to the value written to it by the host

The host constantly monitors and toggles the state of the user LED

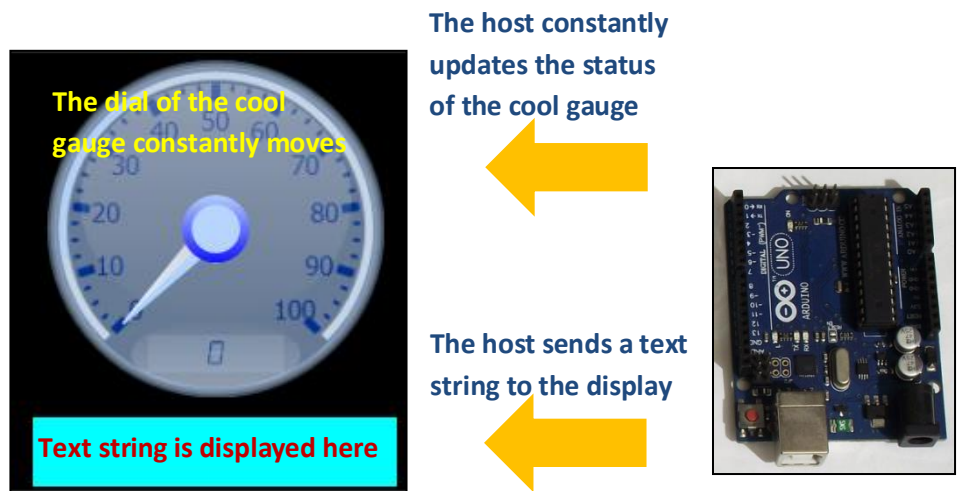


To recreate the application described in this demo, the user first creates a ViSi Genie program in the 4D Workshop IDE and downloads it to a 4D display module. The user can also download and open the already-completed ViSi Genie program from this Github repository:

<https://github.com/4dsystems/ViSi-Genie-Arduino-Library>

The Arduino host, on the other hand, is programmed using the Arduino IDE. The latest ViSi-Genie-Arduino library files are in also in the Github repository referred to above.

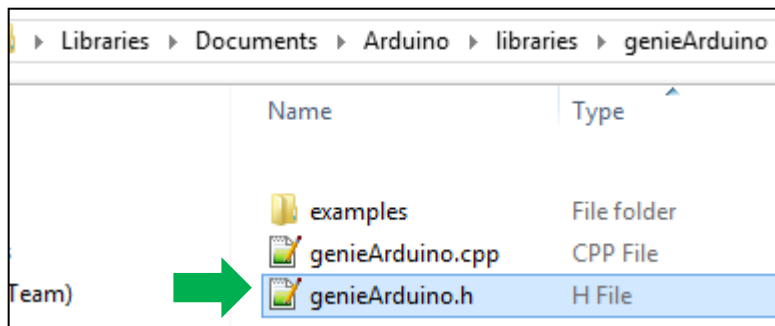
This application note was written to guide the user in setting up a working system in the shortest possible time and to cover the important aspects in the simplest possible manner. For topics that require further explanations, references are pointed out to the user for further study.



Note: The latest version of the ViSi-Genie-Arduino library (as of July 2014) is

```
50 #define GENIE_VERSION "GenieArduino 20-JUL-2014"
```

The library version can be checked by opening the library file “**genieArduino.h**”.



Instructions for installing the library are in the section “**Program the Arduino Host**”. **Users are encouraged to always update to the latest version.**

Setup Procedure

The user can download the ViSi-Genie project example from:

<https://github.com/4dsystems/ViSi-Genie-Arduino-Library>

For instructions on how to launch Workshop 4, how to open a ViSi-Genie project, and how to change the target display, kindly refer to the section “**Setup Procedure**” of the application note

[ViSi-Genie Getting Started - First Project for Picaso Display Modules](#) (for Picaso)

or

[ViSi-Genie Getting Started - First Project for Diablo16 Display Modules](#) (for Diablo16).

Create a New Project

For instructions on how to create a new ViSi-Genie project, please refer to the section “**Create a New Project**” of the application note

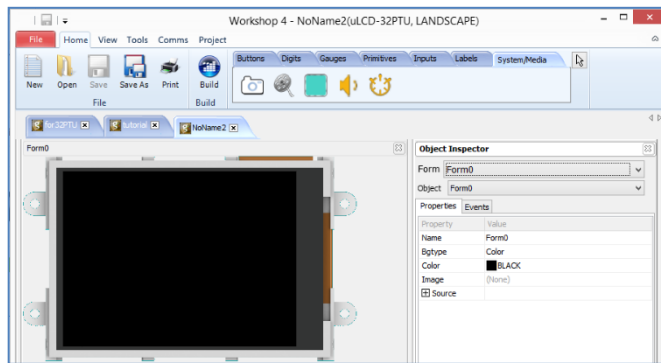
[ViSi-Genie Getting Started - First Project for Picaso Display Modules](#) (for Picaso)

or

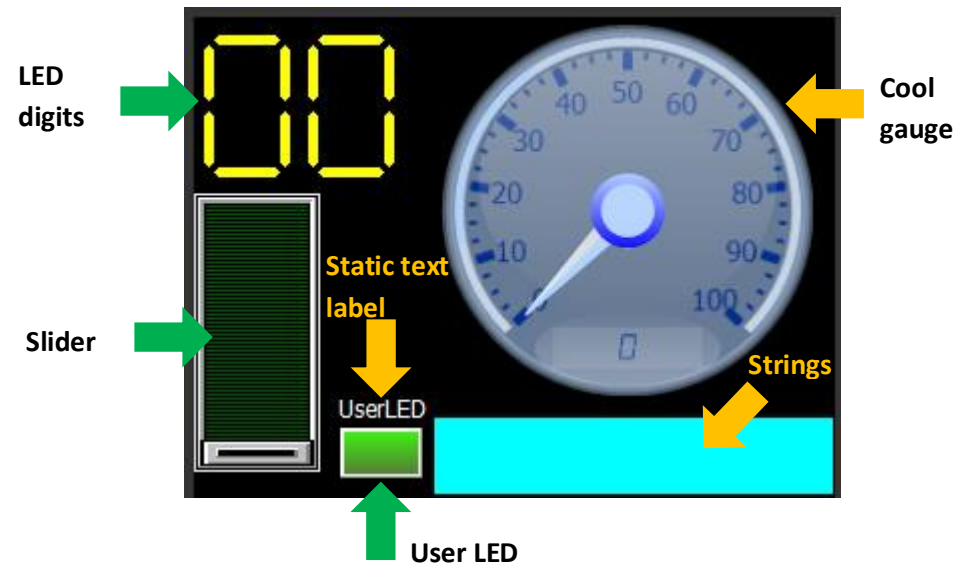
[ViSi-Genie Getting Started - First Project for Diablo16 Display Modules](#) (for Diablo16).

Design the Project

Everything is now ready to start designing the project. **Workshop 4** displays an empty screen, called **Form0**. A **form** is like a page on the screen. The form can contain **widgets** or **objects**, like sliders, displays or keyboards. Below is an empty form.



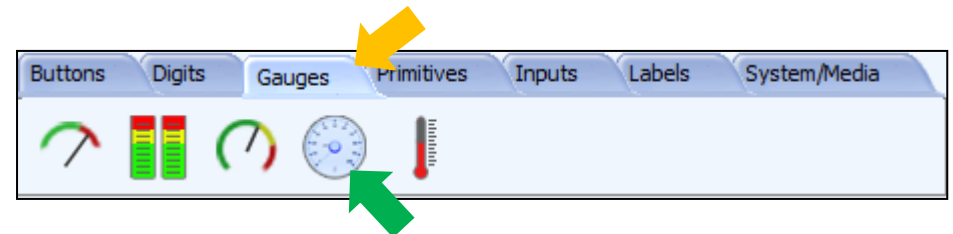
At the end of this section, the user will be able to create a form with six objects. The final form will look like as shown below, with the labels excluded.



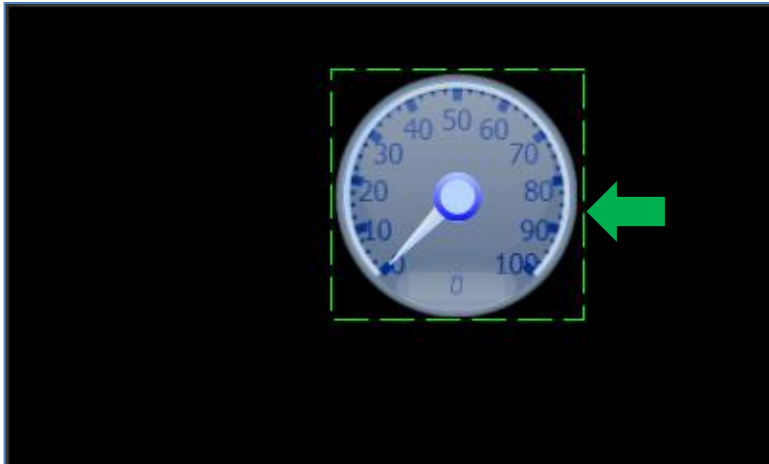
The procedures for adding each of these objects will now be discussed.

Add a Cool Gauge Object

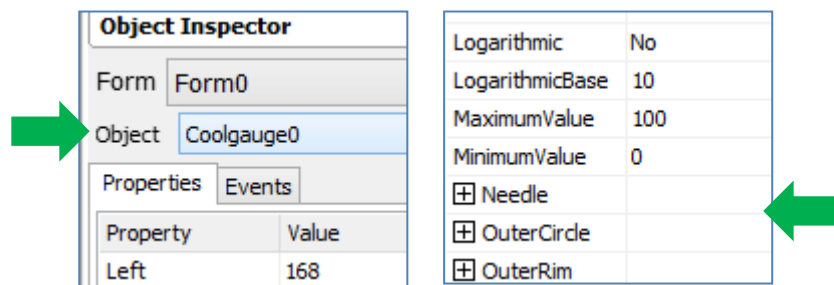
The cool gauge constantly receives values from the Arduino host. The dial of the cool gauge will constantly move to correspond with the received values. To create a cool gauge, go to the **Gauges** pane then click on the **cool gauge** icon.



Click on the **WYSIWYG** (What-You-See-Is-What-You-Get) screen to place the cool gauge. The WYSIWYG screen simulates the actual appearance of the display module screen.



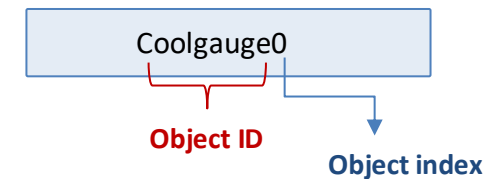
The object can be dragged to any desired location and resized to the desired dimensions. The **Object Inspector** on the right part of the screen displays all the properties of the newly created cool gauge object named **Coolgauge0**.



Feel free to experiment with the different properties. Take note of the maximum and minimum values. These determine the range of values that can be sent from the Arduino host to the cool gauge. To know more about gauges, refer to [ViSi-Genie Gauges](#).

Naming of Objects

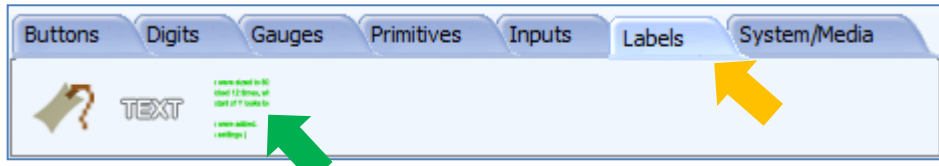
Naming is important to differentiate between objects of the same kind. For instance, suppose the user adds another cool gauge object to the WYSIWYG screen. This object will be given the name Coolgauge1 – it being the second cool gauge in the program. The third cool gauge will be given the name Coolgauge2, and so on. An object's name therefore identifies its kind and its unique index number. It has an ID (or type) and an index.



It is important to take note of an object's ID and index. When programming in the Arduino IDE, an object's status can be polled or changed if its ID and index are known. The process of doing this will be shown later.

Add a Text String Object

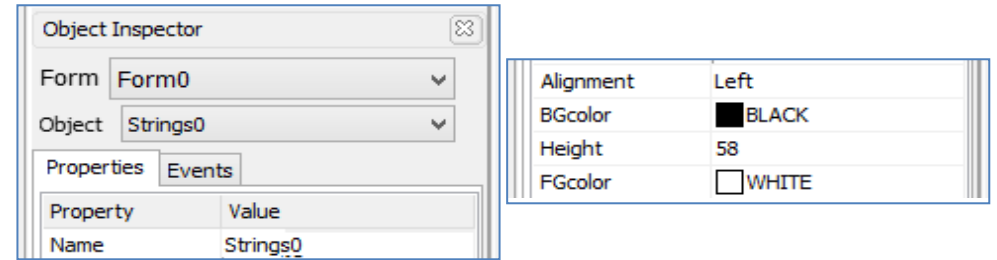
The display module can print text strings received from the Arduino host on the screen. To create a text string object, go to the **Labels** pane then click on the **strings** icon.



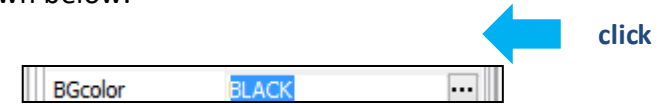
Click on the **WYSIWYG** screen to place the string. Again, the WYSIWYG screen simulates the actual appearance of the display module screen.



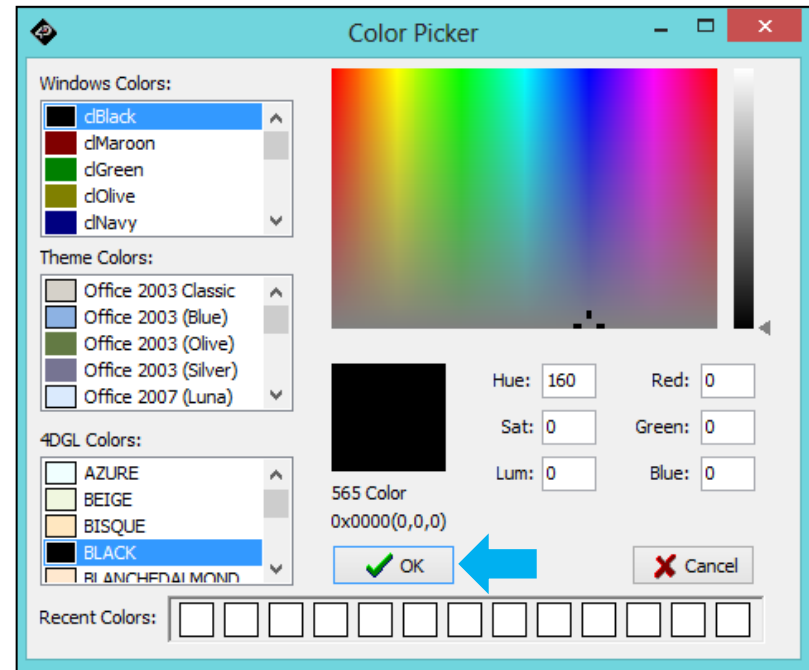
The space inside the red box will be the space occupied by the text string to be displayed. The object can be dragged to any desired location and resized to the desired dimensions. The **Object Inspector** on the right part of the screen displays all properties of the newly created strings object named **Strings0**.



To add background and foreground colours for the text string, edit the properties as shown below.



Choose the desired colour and click OK.



Do the same for the foreground colour.

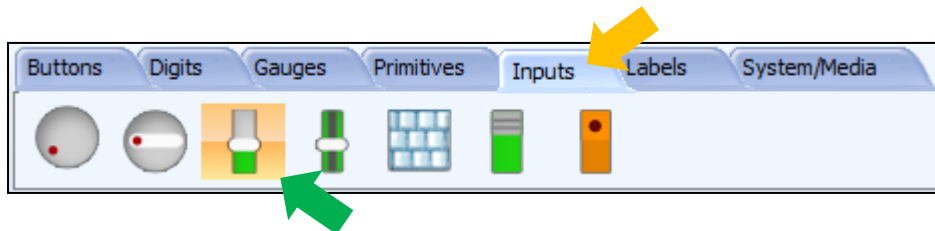


When done, the form should look similar to that shown below.



Add a Slider Object

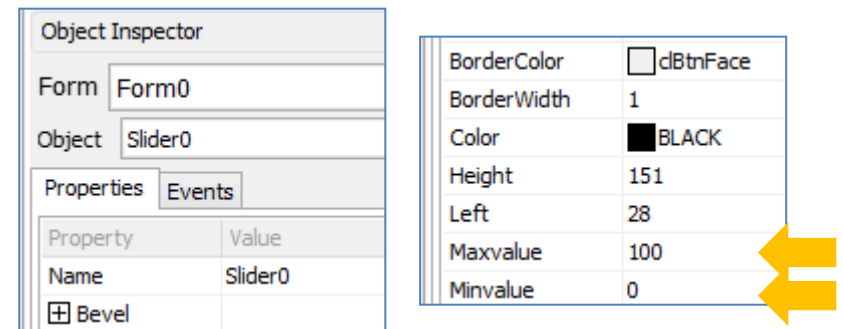
The slider sends a message to the Arduino host when its status has changed. To add a slider, go to the **Inputs** pane and click on the **slider** icon.




Click on the WYSIWYG screen to place a slider object. Drag the object to any desired location.

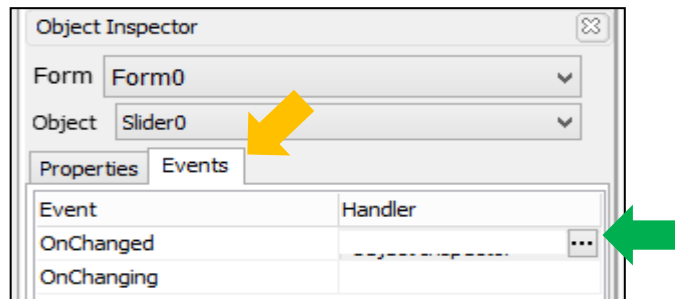


In the **Object Inspector**, the minimum value is 0 and maximum is 100 by default.

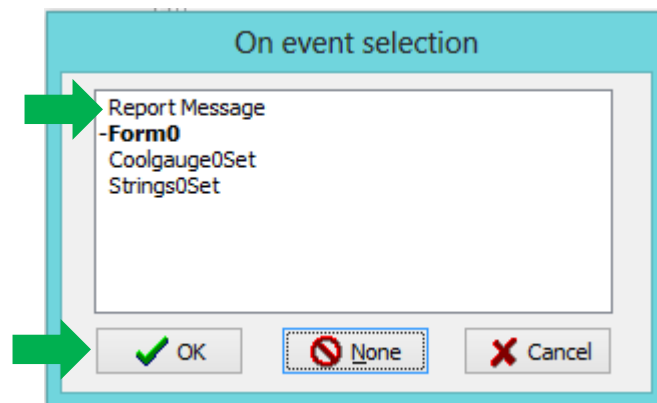


Report Event

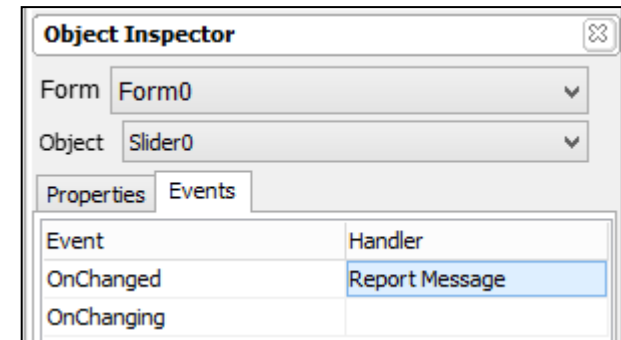
An object can report its current status independently without being asked by the Arduino host. A slider, for example, can be configured to report its current status to the host each time it is moved. To do this, click on the Events tab in the object inspector and click on the  symbol in the OnChanged line.



The **On event selection** window appears. Select **Report Message** and click **OK**.



The Events pane is now updated.

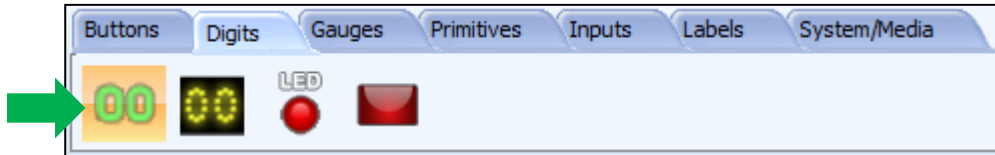


Now every time the slider is moved or its status has changed, it sends a **message** to the host controller. To be more exact, the slider will send a report message when the stylus or finger moving it is lifted off the screen. Selecting the **OnChanging** event, on the other hand, causes the slider to send messages while it is being moved (the moving finger or stylus is not lifted off yet). To learn more about the onChanged and OnChanging events, read the application note [ViSi-Genie onChanging-and-onChanged-Events](#).

The message or data being sent has a format which the Arduino host must understand. A section of this application note is dedicated to explaining this format (called the ViSi-Genie Communication Protocol) used by the display module. Advanced users may refer to the [ViSi-Genie User Reference Manual](#).

Add a LED Digits Object

The **LED digits** object will display values received from the Arduino host. To add a LED digits object, go to the **Digits** pane and select the first icon.



Click on the WYSIWYG screen to place it.

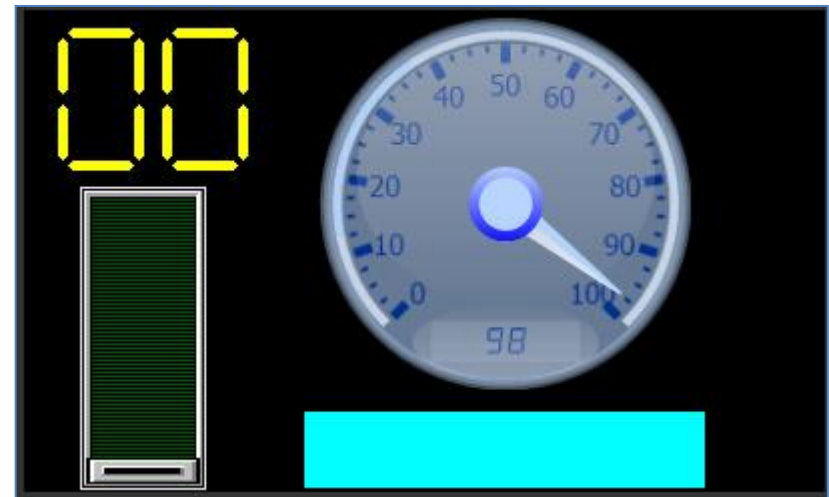


Go to the Object inspector and set the following property values.

Object Inspector	
Form	Form0
Object	Leddigits0
Properties Events	
Property	Value
Name	Leddigits0
Color	BLACK
Decimals	0
Digits	2

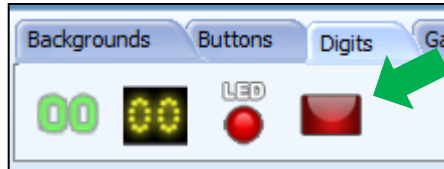
Height	73
LeadingZero	Yes
Left	4
OutlineColor	BLACK
Palette	
High	YELLOW
Low	BLACK
Top	8
Visible	Yes
Width	105

The updated appearance of the LED digits object is shown below.



Add a User LED Object

To add a user LED object, go to the **Digits** pane and select the user LED icon.



Click on the WYSIWYG screen to place it.

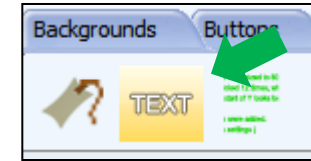


The properties of **Userled0** are:

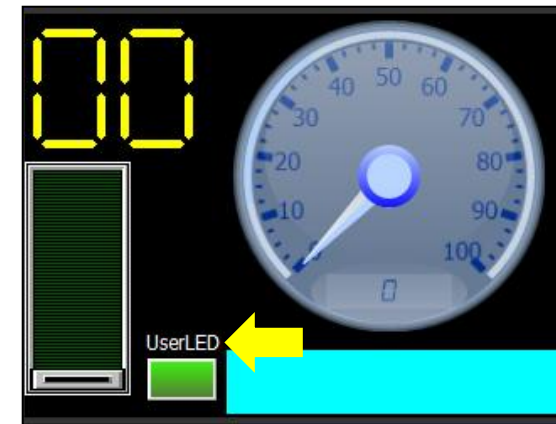
Object Inspector		Active		Yes		PaletteEx	
Form		Bevel				High1	
Form0		Height		27		0x14EC3F	
Object		Left		72		High2	
Userled0		OutlineColor		BLACK		0x377B56	
Properties		OutlineWidth		0		Low1	
Events						0x000051	
Property						Low2	
Value						BLACK	
Name				Userled0			

Add a Static Text Object

The **Static Text** object is under the Backgrounds pane.



Click on the WYSIWYG screen to place it.



The property **“Caption”** defines the text to be displayed by Statictext0. The properties of **Statictext0** are:

Object Inspector		AutoSize		Yes		Transparent		Yes	
Form		Caption		UserLED		Visible		Yes	
Form0		Color		BLACK		Width		45	
Object		Font		(WHITE, [], Te		WordWrap		Yes	
Properties		Height		14					
Events		Left		72					
Property		Top		188					
Value									
Name				Statictext0					

Build and Upload the Project

For instructions on how to build and upload a ViSi-Genie project to the target display, please refer to the section “**Build and Upload the Project**” of the application note

[ViSi-Genie Getting Started - First Project for Picaso Display Modules](#) (for Picaso)

or

[ViSi-Genie Getting Started - First Project for Diablo16 Display Modules](#) (for Diablo16).

The uLCD-32PTU and/or the uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

Identify the Messages

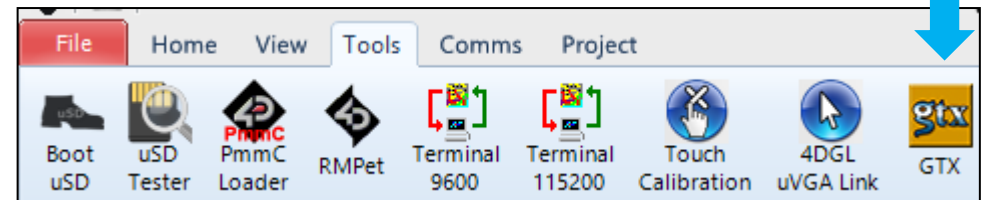
The display module is going to generate and send messages to the host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for the user to be able to properly program the host controller. The [ViSi-Genie User Reference Manual](#) is recommended for advanced users.

Use the GTX Tool to Analyse the Messages

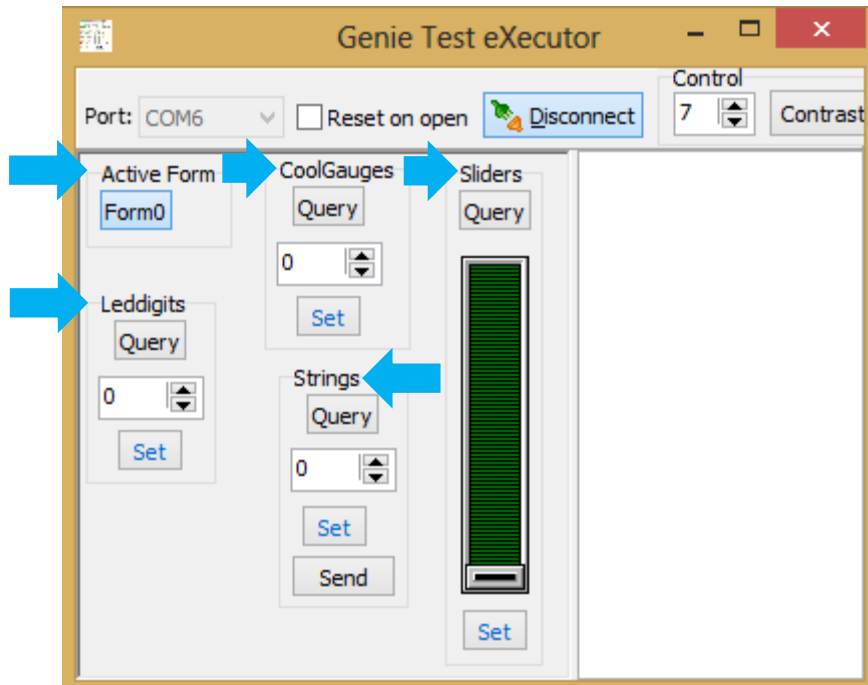
Using the GTX or **Genie Test eXecutor** tool is the first option to get the messages sent by the screen to the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential debugging tool.

Launch the GTX Tool

Under **Tools** click on the GTX tool button.



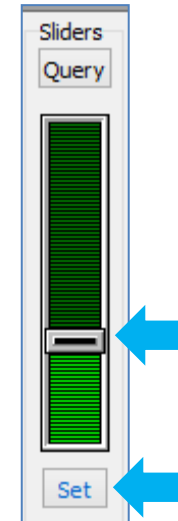
A new window appears, with the form and objects created previously.



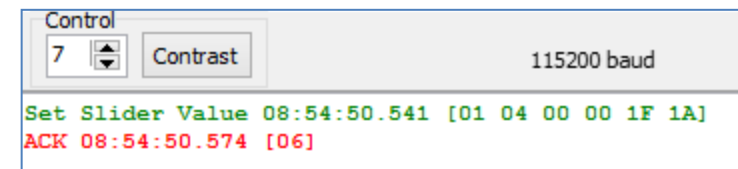
The Slider Object

Change the Status of the Slider

In the GTX tool window, move the slider and press **Set**. On the display module, note that the slider has moved.



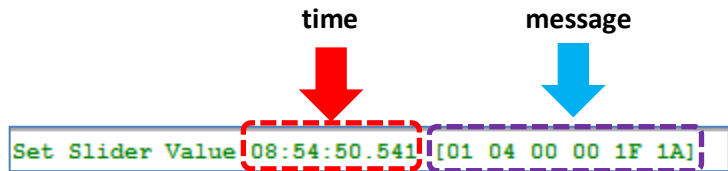
Also, messages are sent to and received from the display module.



The white area on the right displays

- in **green** the messages sent to the display module
- and in **red** the messages received from the display module

The actual message bytes are those inside the brackets. These values are in hexadecimal. The figure preceding the actual message is the computer time at which the message is sent. A label is also included to tell the observer what the message represents.



The message sent is formatted according to the following pattern:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
01	04	00	00	1F	1A
WRITE_OBJ	Slider	First	0x001F		

The message stands for “Write to the first slider object on the display module the value 0x001F.” Converting the hexadecimal value 0x001F to decimal will yield the value 31.

The checksum is a means for the host to verify if the message received is correct. This byte is calculated by XOR’ing all bytes in the message from (and including) the CMD or command byte to the last parameter byte. Then, the result is appended to the end to yield the checksum byte. If the message is correct, XOR’ing all the bytes (including the checksum byte) will give a result

of zero. Checking the integrity of a message using the checksum byte is handled automatically by the Arduino host thru the ViSi-Genie-Arduino library.

ACK = 0x06 as shown below

```
ACK 08:54:50.574 [06]
```

is an acknowledgment from the display module which means that it has understood the message.

Message from a Slider

Remember that the slider was configured to **Report a Message** when its status has changed. Now move the slider on the display module with a stylus or a finger. Observe the message sent by the display module to the PC.

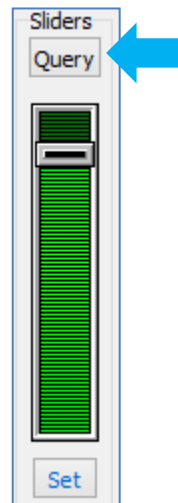
```
Slider Change 09:02:36.533 [07 04 00 00 42 41]
```

The message from slider object is formatted according to the following pattern:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
07	04	00	00	42	41
REPORT_EVENT	Slider	First	0x0042		

Interrogate the Display for the Status of the Slider

Suppose the slider object is not configured to report an event when it has moved. The host or the PC can ask the display module for the value of the slider by sending a message. Now on the display module move the slider randomly. In the GTX tool window press Query.



Observe the message area.

```
Request Slider Value 09:32:16.742 [00 04 00 04]
Slider Value 09:32:16.760 [05 04 00 00 59 58]
```

The PC sends a request message. The display module replies with the current value of the slider object. The messages sent and received are formatted according to the following patterns.

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
00	04	00	-	-	04
READ_OBJ	Slider	First	Not applicable		
05	04	00	00	59	58
REPORT_OBJ	Slider	First	0x0059		

REPORT_EVENT vs. REPORT_OBJ

It is important to take note of the difference between REPORT_EVENT and REPORT_OBJ. **REPORT_EVENT** occurs if the user selects the event of a widget in Workshop to be "Report Message". There is no need for the host to ask the display module for the value of the slider. The slider independently sends its current status since it was configured to do so. Whereas **REPORT_OBJ** is a result of the user doing a read of an object from the host, using the Read Object function.

Experimentation with the different objects using the GTX tool is now left to the user as an exercise. The following tables are shown below for reference. Consult the [ViSi-Genie User Reference Manual](#) for more information.

2.1.2 Command and Parameters Table							
Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
READ_OBJ	0x00	Object ID	Object Index	-	-	-	Checksum
WRITE_OBJ	0x01	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum
WRITE_STR	0x02	String Index	String Length	String (1 byte chars)			Checksum
WRITE_STRU	0x03	String Index	String Length	String (2 byte chars)			Checksum
WRITE_CONTRAST	0x04	Value	-	-	-	-	Checksum
REPORT_OBJ	0x05	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum
REPORT_EVENT	0x07	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum

This table is found in section 2.1 of the [ViSi-Genie User Reference Manual](#).

Object	ID
Dipswitch	0 (0x00)
Knob	1 (0x01)
Rockerswitch	2 (0x02)
Rotaryswitch	3 (0x03)
Slider	4 (0x04)
Trackbar	5 (0x05)
Winbutton	6 (0x06)
Angularmeter	7 (0x07)
Coolgauge	8 (0x08)
Customdigits	9 (0x09)
Form	10 (0x0A)
Gauge	11 (0x0B)
Image	12 (0x0C)
Keyboard	13 (0x0D)
Led	14 (0x0E)
Leddigits	15 (0x0F)

Meter	16 (0x10)
Strings	17 (0x11)
Thermometer	18 (0x12)
Userled	19 (0x13)
Video	20 (0x14)
Statictext	21 (0x15)
Sound	22 (0x16)
Timer	23 (0x17)
Spectrum	24 (0x18)
Scope	25 (0x19)
Tank	26 (0x1A)
UserImages	27 (0x1B)
PinOutput	28 (0x1C)
PinInput	29 (0x1D)
4Dbutton	30 (0x1E)
AniButton	31 (0x1F)
ColorPicker	32 (0x20)
UserButton	33 (0x21)

This table is found in section 3.3 of the [ViSi-Genie User Reference Manual](#).

Program the Arduino Host

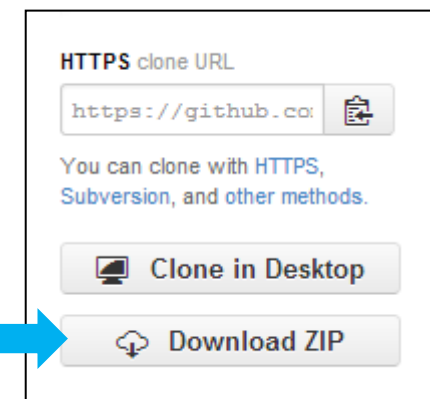
This section discusses how to program the Arduino host to make it work with the display module. It is assumed that the user has a basic understanding of how the Arduino host works and how to program in the Arduino IDE. Inexperienced users may need to frequently refer to the Arduino website for more information.

Download and Install the ViSi-Genie-Arduino Library

The ViSi-Genie-Arduino library files are located here:

<https://github.com/4dsystems/ViSi-Genie-Arduino-Library>

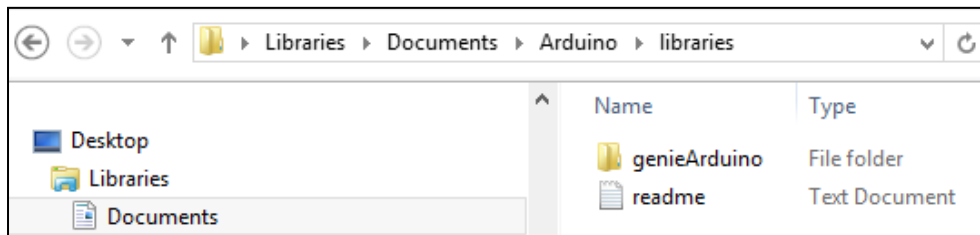
On the right side of the github page, click on the Download ZIP button. Save the zip file and extract its contents to the folder where additional Arduino libraries are saved.



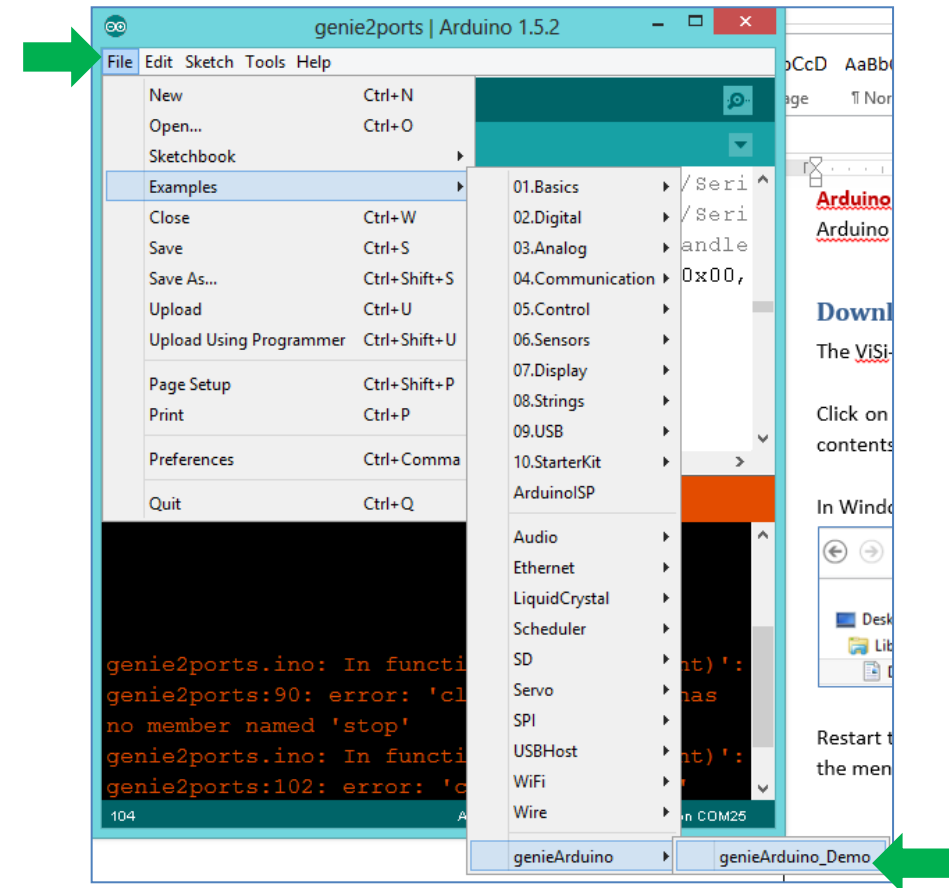
Here is a link to a tutorial on installing additional libraries in the Arduino IDE.

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

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



Remember to restart the Arduino IDE after installing the libraries. The genieArduino demo sketch should be accessible under the File – Examples menu.



Understanding the Arduino Sketch Demo

Open the genieArduino_Demo sketch. Note that comments have been added to the code to help the user. Additional explanations are now given below.

Open a Serial Port and Set the Baud Rate

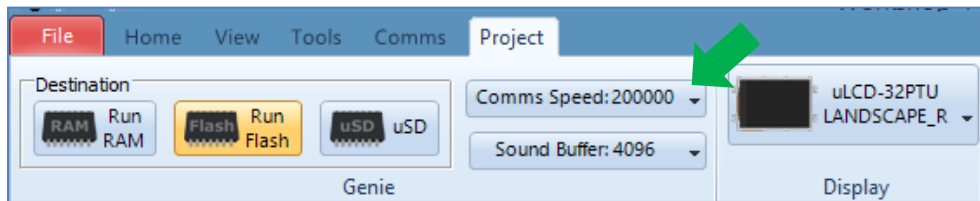
In the `setup ()`, the line shown below sets the data rate in bits per second (baud) for serial data transmission of the port **Serial0**.

```
Serial.begin(200000); // Serial0 @ 200000 (200
```

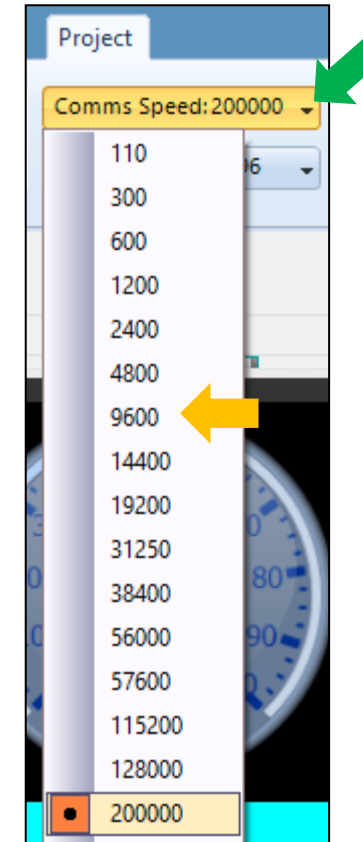
The next line indicates that the ViSi-Genie-Arduino library will use the port **Serial0** for communicating with the display.

```
genie.Begin(Serial); // Use Serial0 for talki
```

Port **Serial0** cannot be used therefore for any other purposes such as talking to the serial monitor. Logically, the 4D display should also communicate with the Arduino host at the same baud rate. To check the baud rate of the ViSi Genie program go to the project menu in Workshop.



To change the baud rate of the ViSi Genie program, simply click on the drop down arrow.



Choose the desired baud rate (9600 bps for instance) and download the program to the display module again. Now change the baud rate of the Arduino host as well.

```
Serial.begin(9600); Serial0 @ 9600 Baud
genie.Begin(Serial); // Use Serial0 for talking
```

For Arduino boards with four hardware serial ports (like the Due and Mega 2560), the user can choose another port to talk to the display. To use Serial2 at 9600 bps for example:

```
//Serial.begin(9600); // Serial0 @ 9600 (9600)
//Serial1.begin(9600); // Serial1 @ 9600 (9600)
Serial2.begin(9600); // Serial2 @ 9600 (9600)
//Serial3.begin(9600); // Serial3 @ 9600 (9600)

//genie.Begin(Serial); // Use Serial0 for talk
//genie.Begin(Serial1); // Use Serial1 for tal
genie.Begin(Serial2); // Use Serial2 for talki
//genie.Begin(Serial3); // Use Serial3 for tal
```

To use a software serial port,

```
#include <genieArduino.h>
#include <SoftwareSerial.h>
Genie genie;
SoftwareSerial DisplaySerial(10,11);
#define RESETLINE 4

void setup ()
{
  Serial.begin(57600); //Serial port for PC and Ard
  DisplaySerial.begin(9600);
  genie.Begin(DisplaySerial); // Use DisplaySerial
  genie.AttachEventHandler(myGenieEventHandler); //
```

Refer to the product page of your Arduino board to know the pins that can be used for a software serial port. The section “**Connect the 4D Display Module to the Arduino Host**” towards the end of this application note shows how the TX and RX pins of the host are connected to those of the display. The following are the SoftwareSerial baud rates that were tested to work with 4D displays using the basic application that comes with this application note.

Processor	Tested SoftwareSerial baud rates (bps)
Picasso	4800, 9600, 14400, 19200, 31250, 38400
Diablo16	14400, 19200, 31250

As of writing, the latest version of the SoftwareSerial is documented to work at 115200 bps. This is easily demonstrated to be false as any interrupt (e.g. the timer interrupt that updates millis()) will result in software serial being unable to ‘extract’ the current byte being read. Thus the maximum, possibly reliable baud rate is 57600 bps. Therefore, since the library currently uses ‘calculated’ baud rates, other than the earlier fixed lookup table, users should be able to successfully test and use valid baud rates (lower than 57600 bps) other than those specified in the table above. Note that if other interrupts are in use the maximum baud rate may need to be lowered further. **Users are encouraged to always use a hardware serial port of the Arduino host for talking to a 4D display.** Refer to the Arduino website for more information on the issues and limitations of the SoftwareSerial library.

N.B.: Again, remember that the baud rate of the display module should match that of the Arduino host. Also, when a serial port has been set to communicate with the display, it cannot be used for talking to other devices.

genieAttachEventHandler()

```
genie.AttachEventHandler(myGenieEventHandler);
```

This function is used to tell the library the name of the function used in the user's code space, so it knows what to call when an event is received and it needs processing. Full explanation is given in the section "Receiving Data from the Display".

Reset the Arduino Host and the Display

It is essential that the display is 'ready' before the host starts sending commands. To satisfy this condition, the host program can be made to reset the display and wait for some time (for the display to start up and initialize properly) before it starts sending commands. This sequence is ideally placed at the start of the host program. To illustrate:

```
#define RESETLINE 4 // Change this if you are r
```

```
pinMode(RESETLINE, OUTPUT); // Set D4 on Arduin
digitalWrite(RESETLINE, 1); // Reset the Displa
delay(100);
digitalWrite(RESETLINE, 0); // unReset the Disp
delay(3500); //let the display start up after t
```

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), simply connect RES to 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.

```
#define RESETLINE 2 // Change this if you are r
```

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.

```
pinMode(RESETLINE, OUTPUT); // Set D4 on Arduino
digitalWrite(RESETLINE, 0); // Reset the Display
delay(100);
digitalWrite(RESETLINE, 1); // unReset the Display
```

If using jumper wires

Note that the logic state for resetting the display is reversed if not using any of the 4D Arduino Adaptor Shields. It 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.

Set the Screen Contrast

To make sure that the LCD is turned on, write

```
genie.WriteContrast(1); // 1 = Display ON, 0 = OFF
```

A contrast value of 1 can be too low for displays that support contrast levels from 0 to 15. Use a higher contrast level value for these displays. Read further below for more information.

To turn off the display, write

```
genie.WriteContrast(0); // 1 = Display ON, 0 = OFF
```

Most Picaso display modules can only have a value of 1 or 0 (on or off) for contrast. The uLCD-43P/PT/PCT modules and Diablo16 display modules, however, support contrast values from 0 to 15, which makes power saving

possible. This function does not apply to uVGA-II/III modules. Check the datasheet of your display for more information.

Send a Text String

To write to a string object on the display, write,

```
genie.WriteString(0, GENIE_VERSION);
```

The first argument of the member function WriteStr() is the index of the string object to which the string will be sent. The second argument is a null-terminated character array. GENIE_VERSION is a string constant defined in the ViSi-Genie-Arduino library.

The Main Loop

This is now the main part of the program. Three variables are declared and used here – **waitPeriod** for checking how much time has elapsed, **gaugeAddVal** holds the increment or decrement value of the cool gauge, and **gaugeVal** holds the value to be sent to the cool gauge.

```
void loop()
{
    static long waitPeriod = millis();
    static int gaugeAddVal = 1;
    static int gaugeVal = 50;
```

Receiving Data from the Display

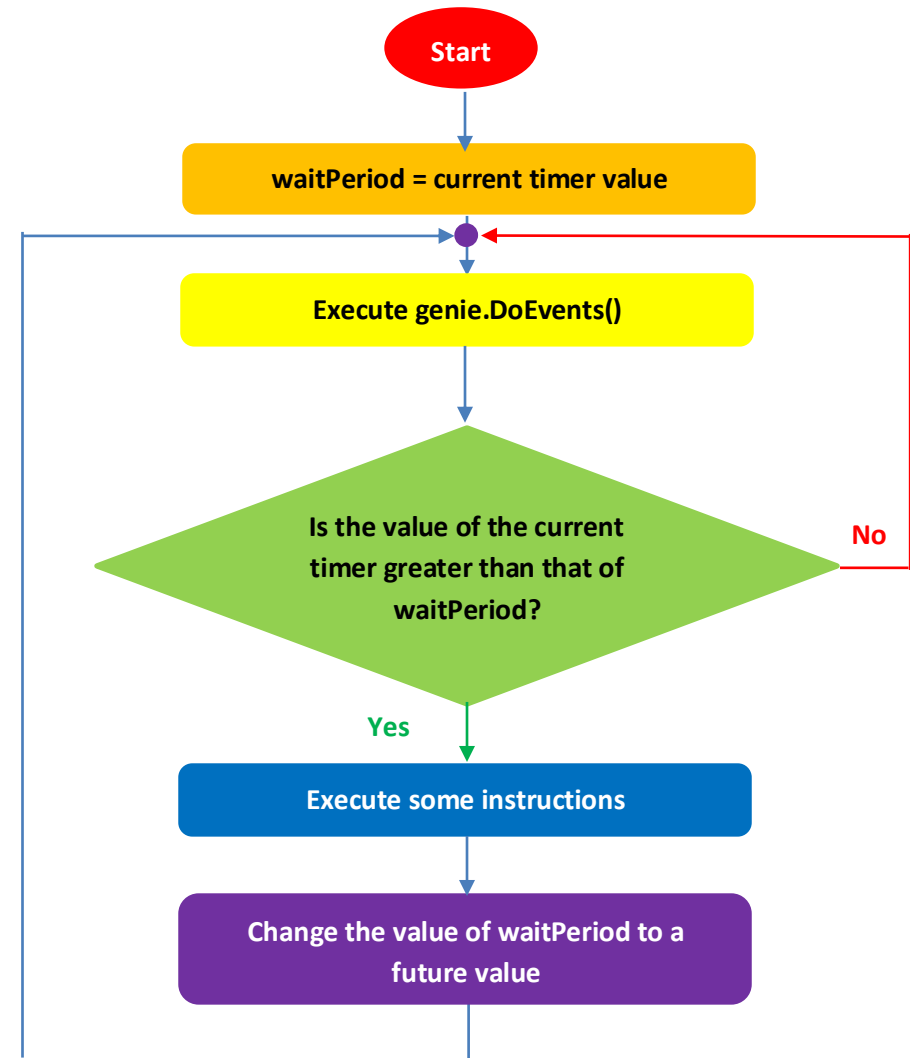
The function below receives and queues the data received from the display.

```
genie.DoEvents();
```

Another function (to be written by the user) is needed to process the received data. The function **genie.DoEvents()** will call this user-defined function internally. A diagram of how data from the display is received and processed is now shown on the next page.

The Use of a Non-blocking Delay

Note how the main loop uses a non-blocking delay to be able to execute other instructions besides **genie.DoEvents()**. Ideally, **genie.DoEvents()** is the main task and it should be called as often as possible such that all events from the display are processed immediately. The main loop can be represented with the flow chart in the right column. This flow chart is the recommended model for Arduino programs communicating with a 4D display. The use of the function **delay()** is discouraged since events from the display will not be processed in “real time”. Also, the time duration for the execution of other instructions/subroutines besides **genie.DoEvents()** should be kept as short as possible.





1

genie.DoEvents()

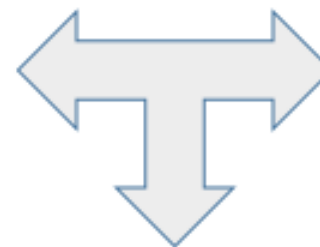
This function receives and queues the data received from the display. The data received can represent an event (a Reported Message sent from the Display due to a user interaction) or can be a result of a Read Object call (a manual poll of information from the user to the display, requesting information from a particular object). `genie.DoEvents()` is located and defined in the ViSi-Genie-Arduino library.

To process the data received, `genie.DoEvents()` calls an external function which is located in the user's code space, not in the library. This external function is called the "Event Handler" – a function written by the user.

2

myGenieEventHandler()

This is a user-defined function and can be named arbitrarily. This function enables the program to evaluate the data queued by `genie.DoEvents()`. This function takes one set or frame of data (one frame is for one event) from the event queue and evaluates it based on the code defined by the user. The user can break down the event into object type and index number, and extract the data from the event of a particular object when the event is received by the Arduino. At the start of this function, the event is removed from the queue using the `genie.DequeueEvent()` function.


**genie.AttachEventHandler()**

This function tells the ViSi-Genie-Arduino library the name of the external function (`myGenieEventHandler()` above) that needs to be called from the user's code. `genie.DoEvents()` does not know what external function to call for processing the received data since this external function is user-defined. There is therefore a need for the user to tell `genie.DoEvents()` which function to call. Do this by writing `genie.AttachEventHandler(name-of-your-function-here)` at the start of the program. By default, the user's function is called `myGenieEventHandler()`.

To sum it up, the display sends information to the Arduino, whether it is by result of a Reported Message or from a Read Object. The Arduino is executing its code in the main loop. It comes to the **genie.DoEvents()** function and calls the ViSi-Genie-Arduino library. The Arduino 'sees' there is some information available and stores it in a queue. When the Arduino processes the queue, the first event is taken out of the queue and processed based on the commands set out in the user-defined Event Handler function. The Arduino then returns to executing any other functions in its main loop, and then jumps to the start of the loop again.

How to Change the Status of an Object

To change the status of the cool gauge, use the function indicated below.



```
genie.WriteObject(GENIE_OBJ_COOL_GAUGE, 0x00, gaugeVal);
gaugeVal += gaugeAddVal;
if (gaugeVal == 99) gaugeAddVal = -1;
if (gaugeVal == 0) gaugeAddVal = 1;
```

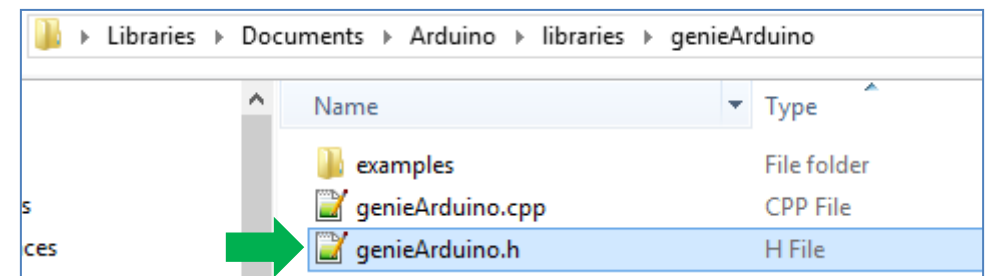
GENIE_OBJ_COOL_GAUGE is the object's ID or type, **0x00** is the index, and **gaugeVal** is the value to be written to the object. Note that **gaugeVal** is incremented or decremented by **gaugeAddVal**. Also **gaugeVal** is limited to a value between and including 0 and 99. Remember that the cool gauge in the display module has minimum and maximum values of 0 and 100.

It is possible to change the status of any object as long as the object ID and index are known. The image below lists the object types or IDs already

defined in the ViSi-Genie-Arduino library. All of the objects can be written to except the **GENIE_OBJ_KEYBOARD** and **GENIE_OBJ_STATIC_TEXT**.

#define GENIE_OBJ_DIPSW	0	GENIE_OBJ_THERMOMETER	18
#define GENIE_OBJ_KNOB	1	GENIE_OBJ_USER_LED	19
#define GENIE_OBJ_ROCKERSW	2	GENIE_OBJ_VIDEO	20
#define GENIE_OBJ_ROTARYSW	3	GENIE_OBJ_STATIC_TEXT	21
#define GENIE_OBJ_SLIDER	4	GENIE_OBJ_SOUND	22
#define GENIE_OBJ_TRACKBAR	5	GENIE_OBJ_TIMER	23
#define GENIE_OBJ_WINBUTTON	6	GENIE_OBJ_SPECTRUM	24
#define GENIE_OBJ_ANGULAR_METER	7	GENIE_OBJ_SCOPE	25
#define GENIE_OBJ_COOL_GAUGE	8	GENIE_OBJ_TANK	26
#define GENIE_OBJ_CUSTOM_DIGITS	9	GENIE_OBJ_USERIMAGES	27
#define GENIE_OBJ_FORM	10	GENIE_OBJ_PINOUTPUT	28
#define GENIE_OBJ_GAUGE	11	GENIE_OBJ_PININPUT	29
#define GENIE_OBJ_IMAGE	12	GENIE_OBJ_4DBUTTON	30
#define GENIE_OBJ_KEYBOARD	13	GENIE_OBJ_ANIBUTTON	31
#define GENIE_OBJ_LED	14	GENIE_OBJ_COLORPICKER	32
#define GENIE_OBJ_LED_DIGITS	15	GENIE_OBJ_USERBUTTON	33
#define GENIE_OBJ_METER	16		
#define GENIE_OBJ_STRINGS	17		

This list is found in the library file "**genieArduino.h**".



The latest library may include more objects. Thus, the user must always update to the latest version.

How to Know the Status of an Object

The Arduino can interrogate the display module for the status of a certain object. For instance, read the current status of the user LED using the command indicated below.

```
// The results of this call will be available to
// Do a manual read from the UserLEd0 object
genie.ReadObject(GENIE_OBJ_USER_LED, 0x00);
```

The status of any object can be read as long as the object ID and index are known. The display module will then reply with a message containing the object's current status. This message is received and queued by **genie.DoEvents()** and dequeued and evaluated by the user's event handler.

The User's Event Handler

First, the union **Event** of the **genieFrame** type is declared.

```
//LONG HAND VERSION, MAYBE MORE VISIBLE AND MORE LIKE
void myGenieEventHandler(void)
{
    genieFrame Event;
```

The **genieFrame** union type is defined in the ViSi-Genie-Arduino library. It contains the structure **reportObject**, which is of the **FrameReportObj** type.

```
union genieFrame {
    uint8_t        bytes[GENIE_FRAME_SIZE];
    FrameReportObj reportObject;
};
```

The **FrameReportObj** structure type contains five bytes, each representing a byte to be received from the display module. To illustrate:

```
struct FrameReportObj {
    uint8_t    cmd;
    uint8_t    object;
    uint8_t    index;
    uint8_t    data_msb;
    uint8_t    data_lsb;
};
```

The next step now is to take an event or a message from the event queue of the **genie.DoEvents()** function. Observe the correct syntax.

```
genie.DequeueEvent(&Event);
```

Then break down the message into its components. A part of the sketch demonstrates how this is done.

```
//If the cmd received is from a Reported Object, which occurs if
if (Event.reportObject.cmd == GENIE_REPORT_OBJ)
{
  if (Event.reportObject.object == GENIE_OBJ_USER_LED)
  {
    if (Event.reportObject.index == 0)
    {
      bool UserLed0_val = genie.GetEventData(&Event);
      UserLed0_val = !UserLed0_val;
      genie.WriteObject(GENIE_OBJ_USER_LED, 0x00, UserLed0_val);
    }
  }
}
}
```

Here is a list of Genie command type constant definitions taken from the **genieArduino.h** file of the ViSi-Genie-Arduino library.

GENIE_READ_OBJ	0
GENIE_WRITE_OBJ	1
GENIE_WRITE_STR	2
GENIE_WRITE_STRU	3
GENIE_WRITE_CONTRAST	4
GENIE_REPORT_OBJ	5
GENIE_REPORT_EVENT	7

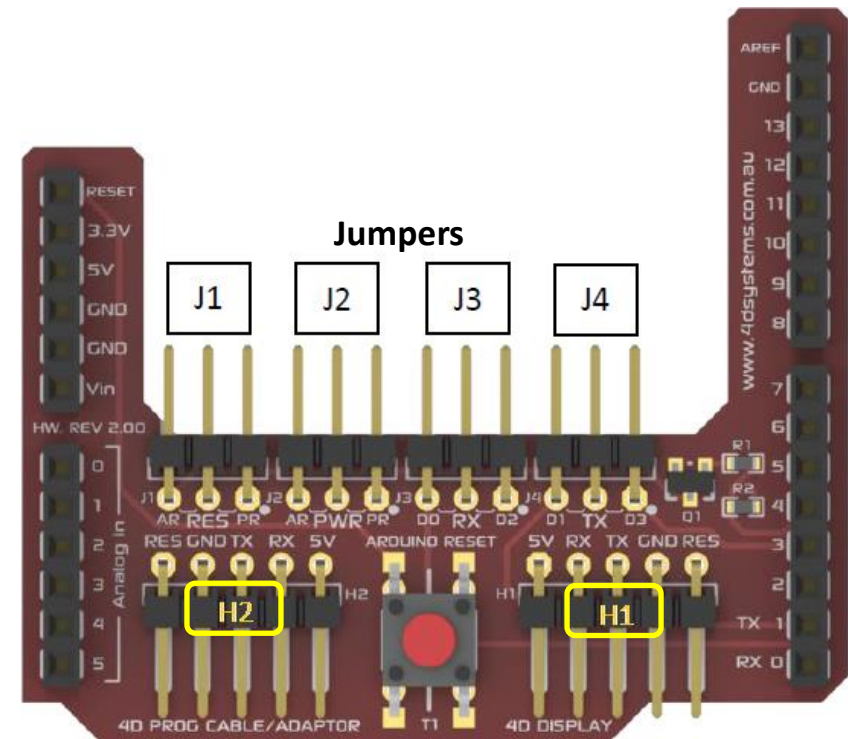
In conclusion the user's event handler evaluates each byte of a message (command, object, object index, and value), and then makes a decision according to the result of this evaluation.

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

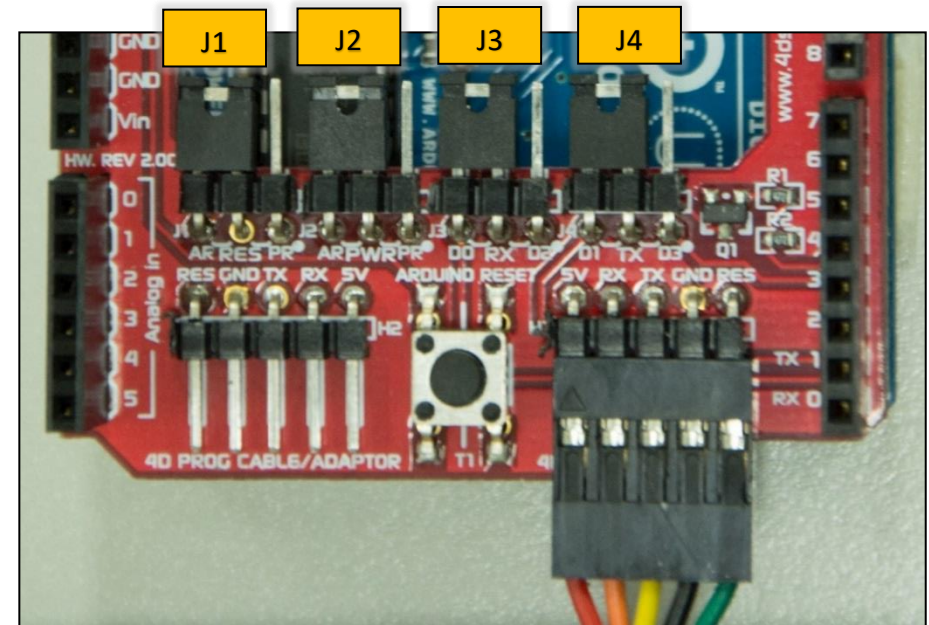
be powered separately thru **H2** (jumper **J2** should be configured accordingly).

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

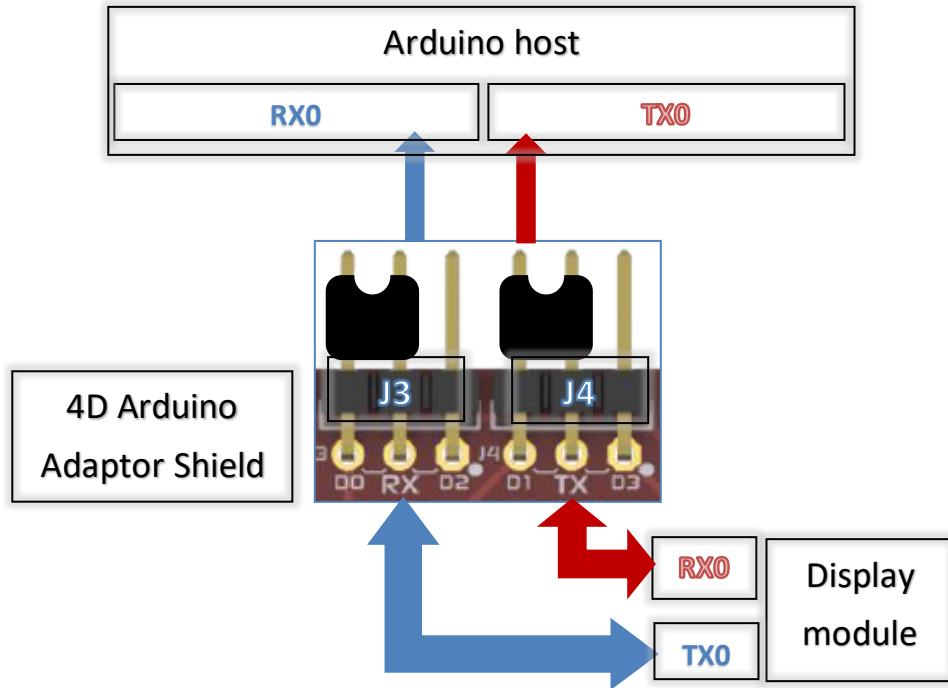
The image on the right column shows the default settings for jumpers J1 to J4.

- 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

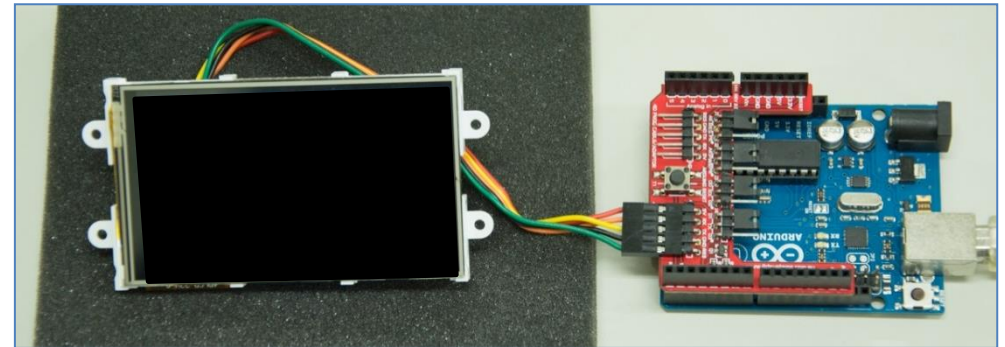
Jumpers J3 and J4 are configured, by default, to connect RX (TX0 of the display module) to D0 (RX0 of the Arduino) and TX (RX0 of the display module) to D1 (TX0 of the Arduino). Communication in this case is thru Serial0 of the Arduino host and COM0 of the display module.



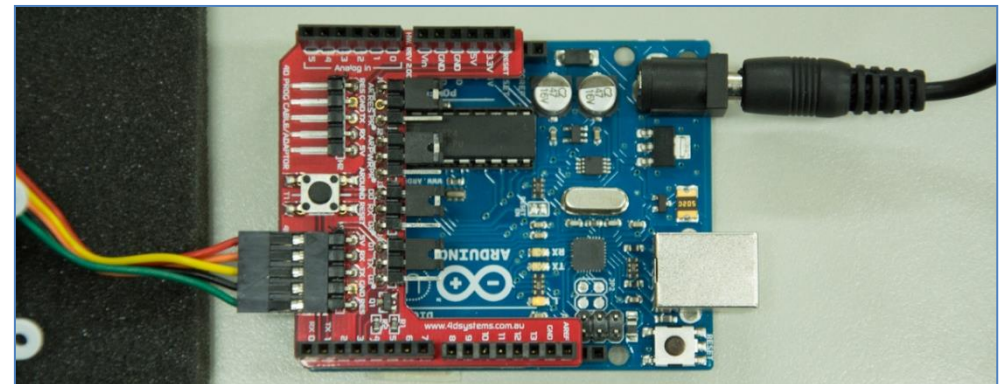
The Arduino Host Powers the Display

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):

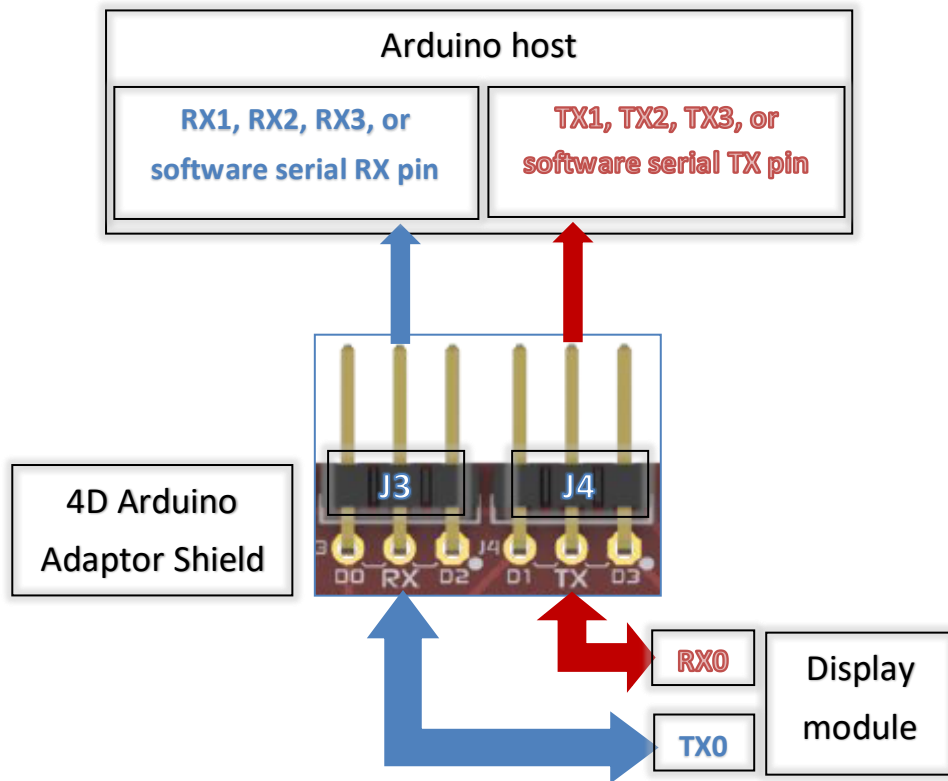


Using the jack (the Arduino host powers the display):



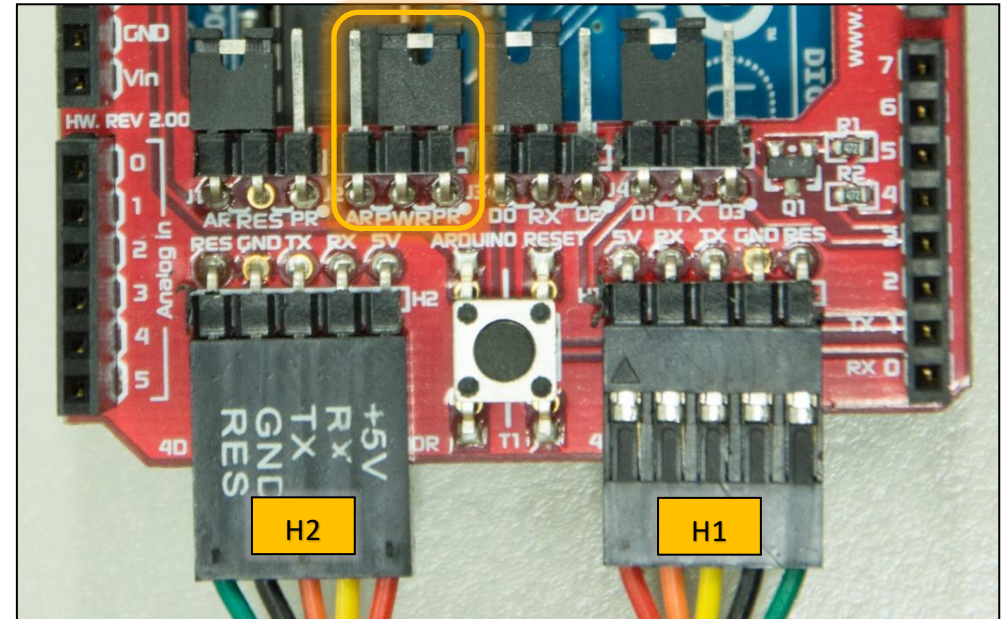
Change the Arduino Host Serial Port

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.

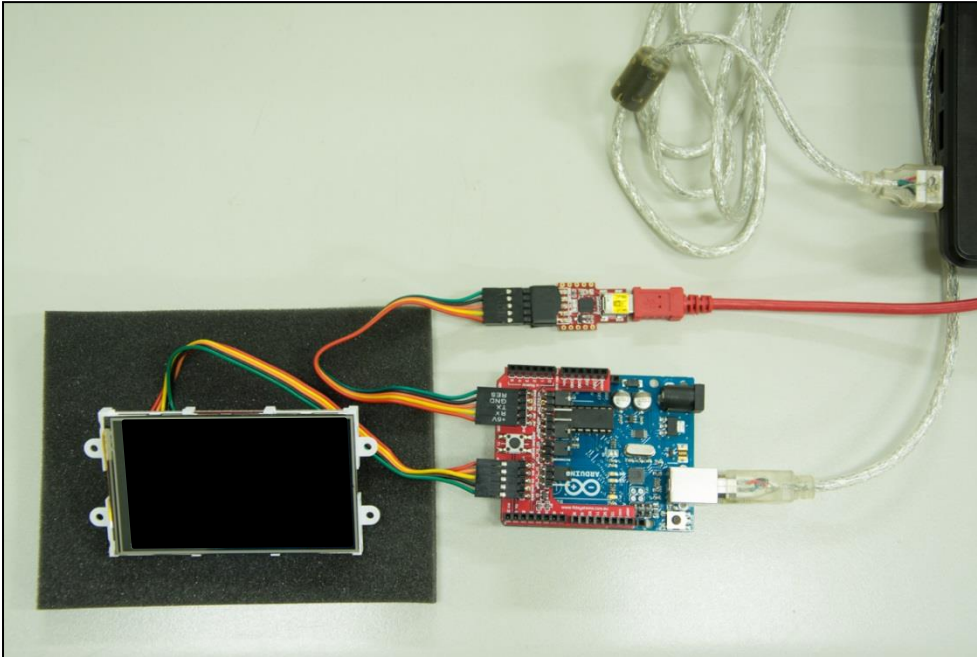


Power the Arduino Host and the Display Separately

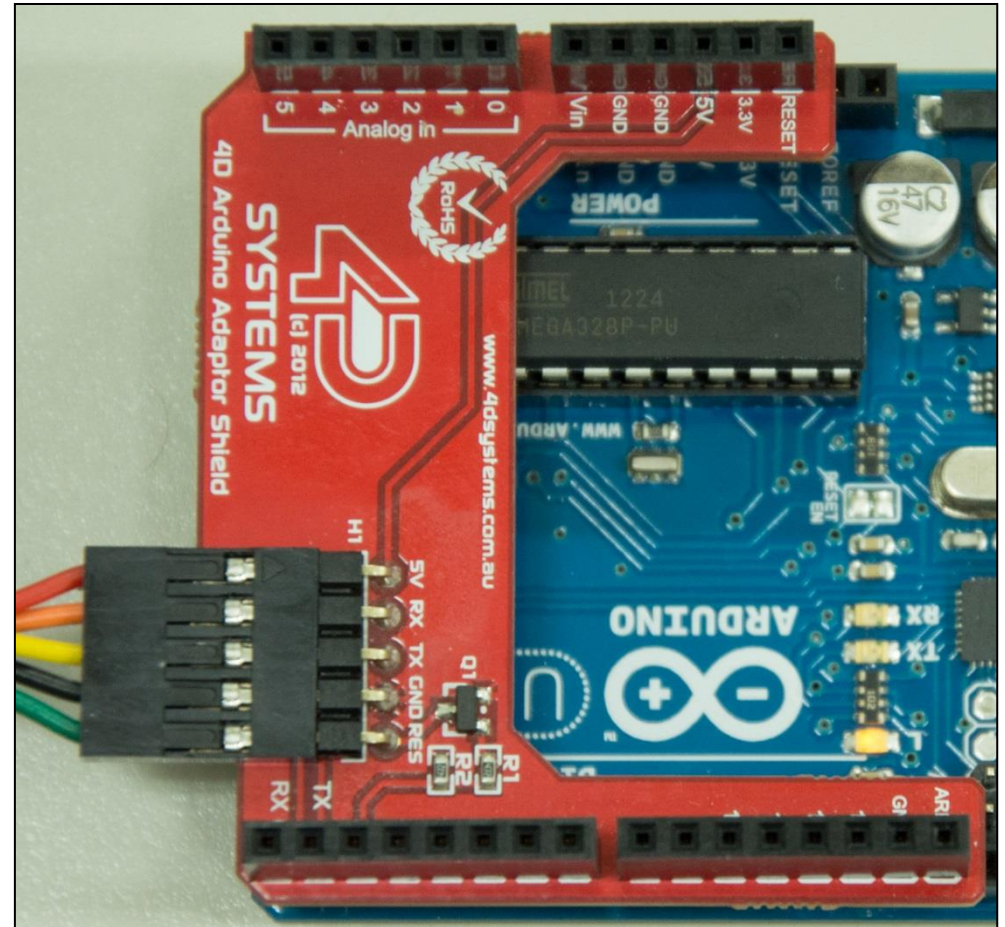
If the display requires a higher current to operate (the uLCD-70DT for instance), it is not advisable to power it off the 5V out of the Arduino host. To power the display separately from the Arduino board, set J2 as shown below. Power will then be supplied to the display thru H2.



H2 is for the 4D USB Programming Cable or μ USB-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.

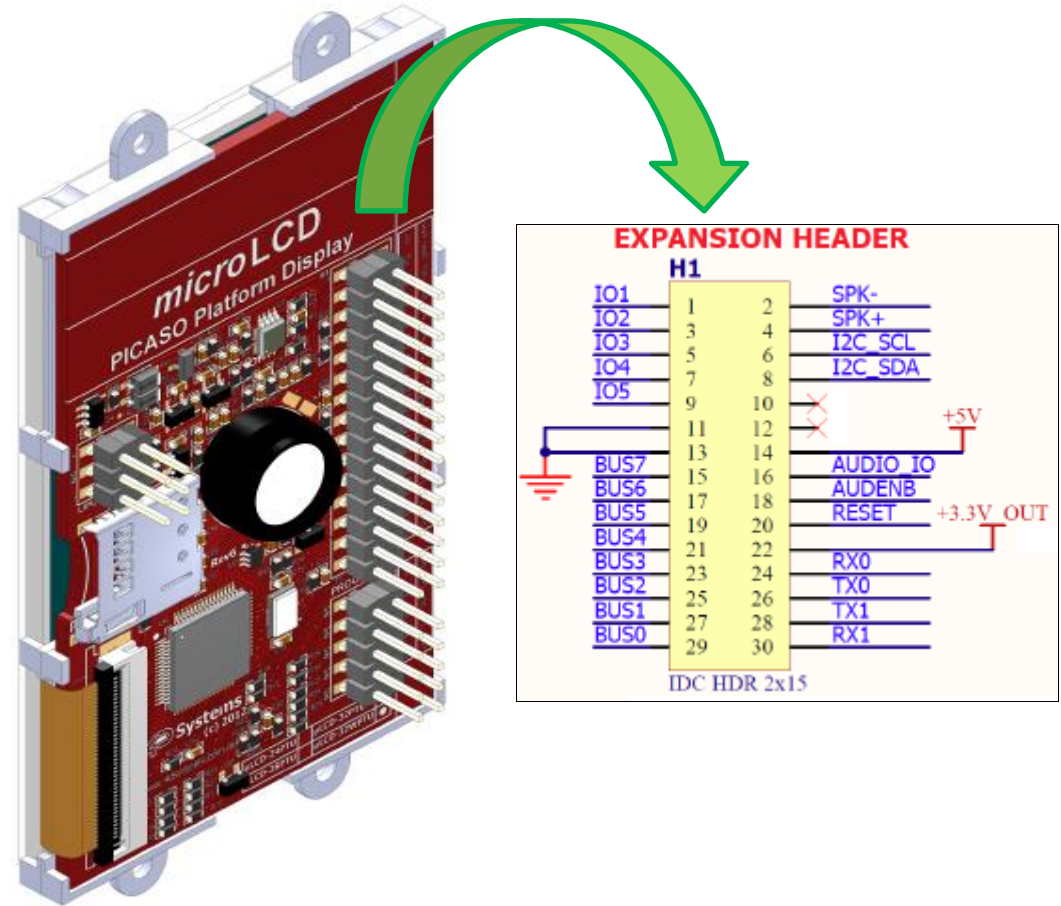
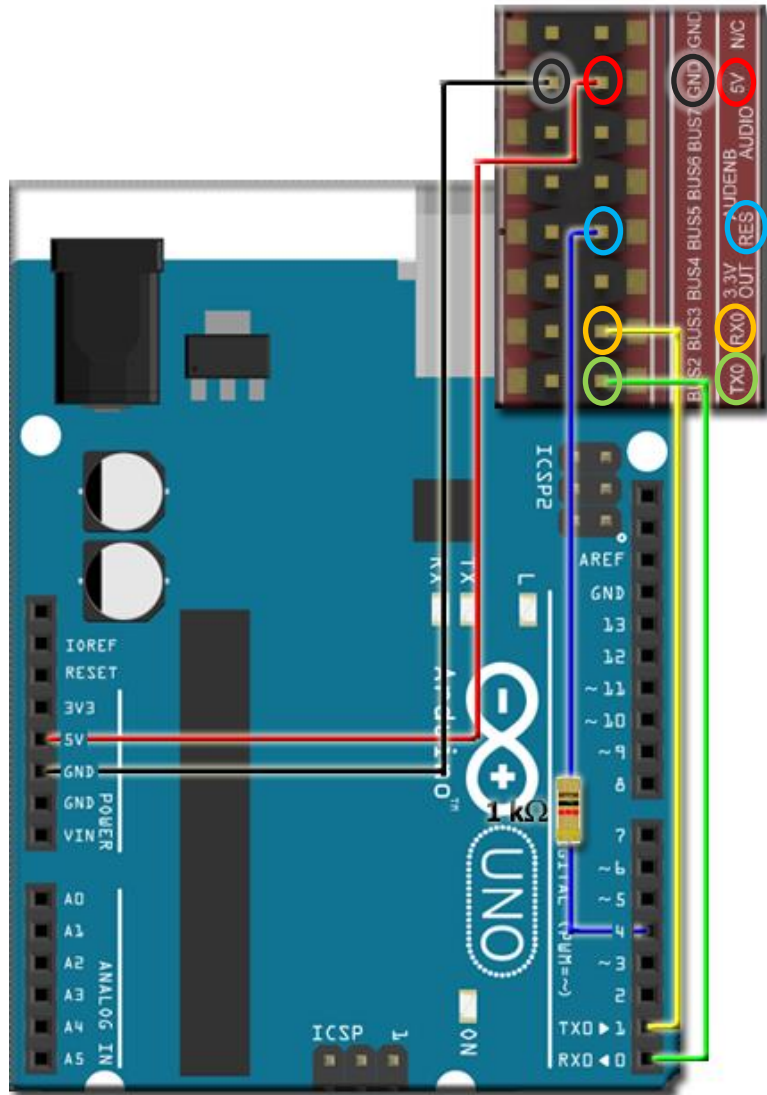
Complete setup (host and display are powered separately):

Note that the display module cannot be programmed thru the μ USB-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.

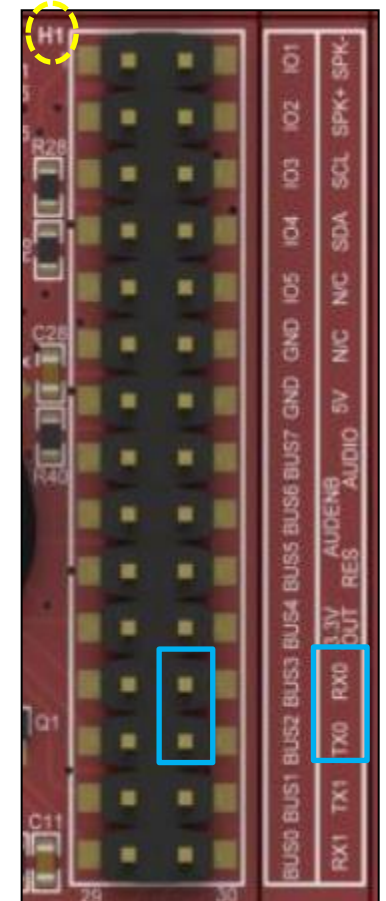
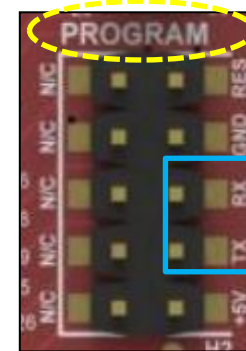
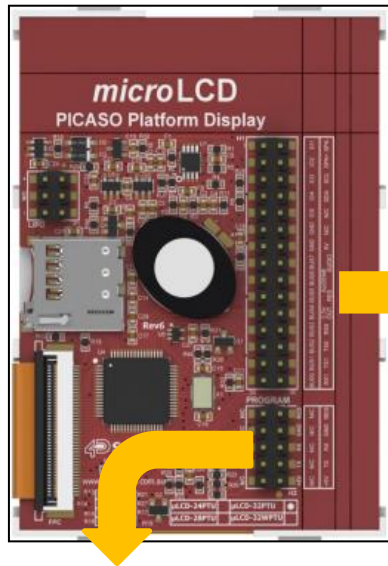
Connection Using Jumper Wires



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.

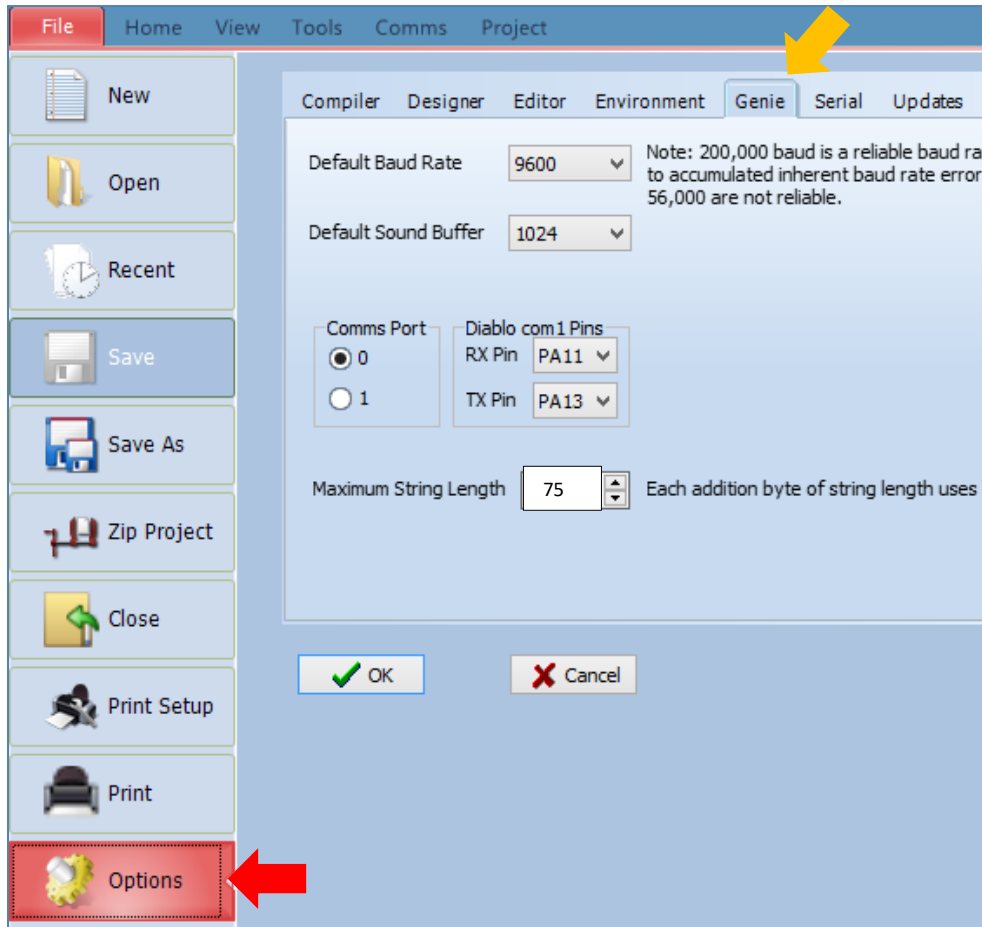
Changing the Serial Port of the Genie Program

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/O Expansion header and the programming header.

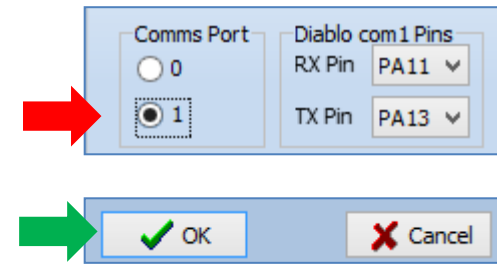


As the reader may have already perceived, the TX and RX pins on the programming header are the same pins as TX0 and RX0 on the H1 I/O expansion header. In Workshop it is possible to change the serial port being used by a ViSi-Genie program. Instructions for doing this are as follows.

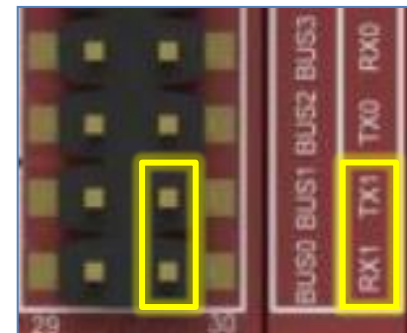
Under the **File** menu, select **Options** then select the **Genie** tab.



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.



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.

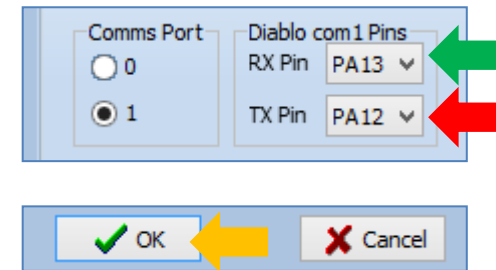


The **Diablo16 processor** has four serial ports – COM0, COM1, COM2, and COM3. The TX and RX pins of COM0 are fixed and are used for programming the processor. Again, COM0 is also the default serial port used by a ViSi-Genie program. The TX and RX pins for COM1, COM2, and COM3, on the other hand, are ‘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

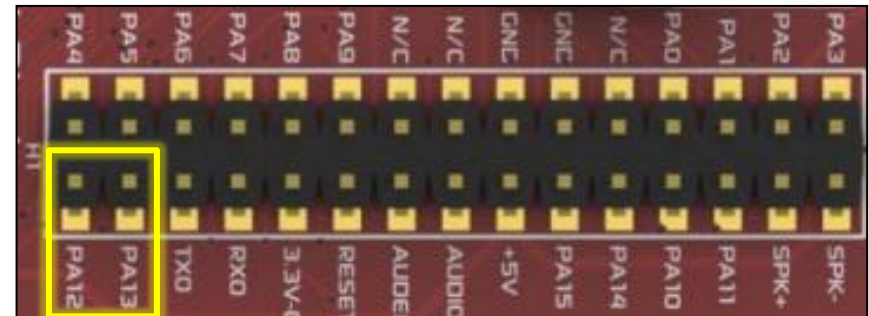
pins that can be used as TX and RX pins for COM1, COM2, and COM3. This table is taken from the [Diablo16 datasheet](#).

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.



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,

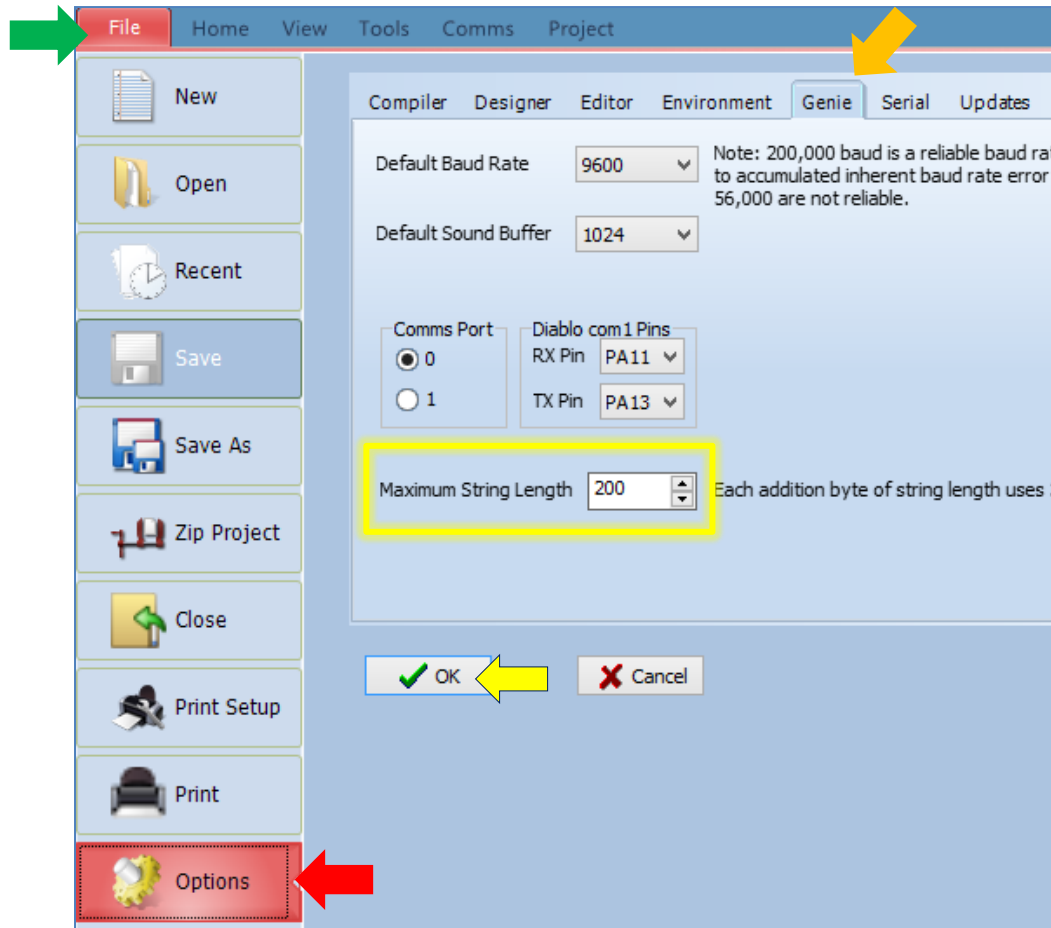


Consult the datasheet of your display for more information.

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.



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

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.