APPLICATION NOTE

# 4D SYSTEMS

*TURNING TECHNOLOGY INTO ART*

# ViSi-Genie RGB LED Control with an Arduino Host

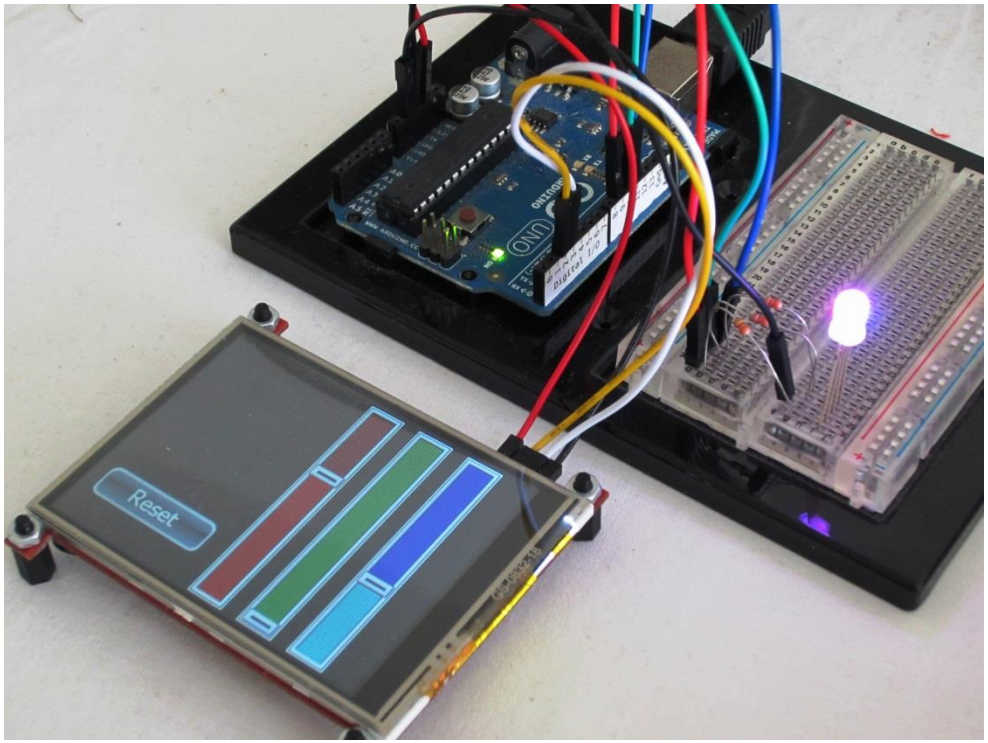
**Document Date: July 22nd, 2014**

**Document Revision: 1.0**

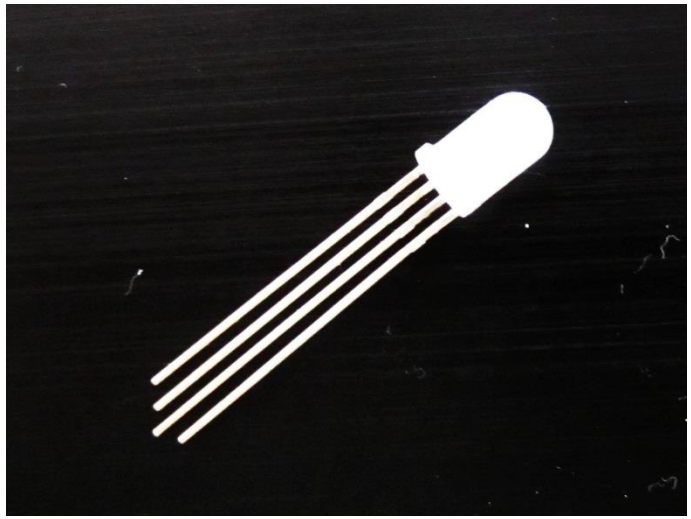
# Description

This Application Note explores the possibilities provided by ViSi-Genie to work with a host.

In this example, the host is an AVR ATmega328 micro-controller-based Arduino Uno board with a red-green-blue LED.




The RGB LED is a classic 5 mm with four pins:



This application note requires:

- Any of the following 4D Picaso display modules:

  uLCD-24PTU uLCD-32PTU uLCD-43(P/PT/PCT)
  uLCD-28PTU uLCD-32WPTU uVGA-III

  and other superseded modules which support the ViSi Genie environment

- The target module can also be a Diablo16 display

  uLCD-35DT uLCD-70DT

  Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor. The display

module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- 4D Programming Cable or µUSB-PA5
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- Any Arduino board with a UART serial port
- 4D Arduino Adaptor Shield (optional) or connecting wires
- Arduino IDE
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

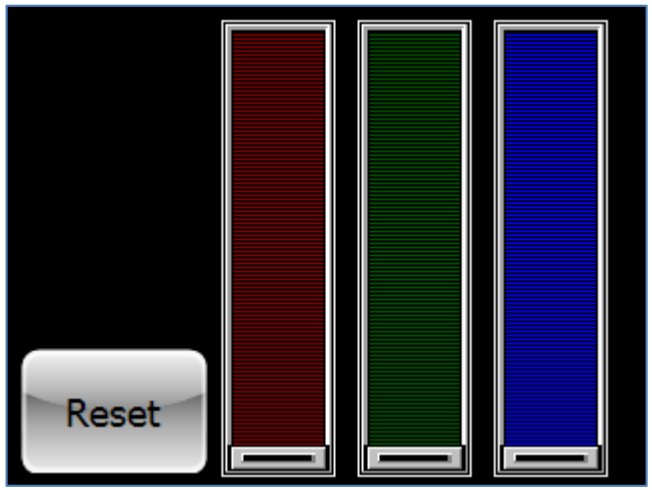
***The ViSi-Genie project and the Arduino sketch are provided as examples to help you along this application note.***

# Content

# Application Overview

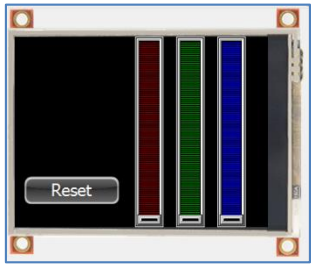
The application allows controlling a red-green-blue LED, which allows virtually any possible colour of the spectrum.

The colour is adjusted by moving each of the three sliders, one per component, red, green or blue. A **Reset** button sets the default mid-range.




The screen is connected to a host, in this example an AVR ATmega328 micro-controller-based Arduino Uno board.

The screen sends commands to the host micro-controller, when a slider is moved and when the Reset button is pushed.




When a slider is moved or the Reset button pushed, the screen…

…sends commands to…

**HOST MCU**

…the host micro-controller.

Then the host micro-controller translates the commands and drives the RGB LED.

**HOST MCU**

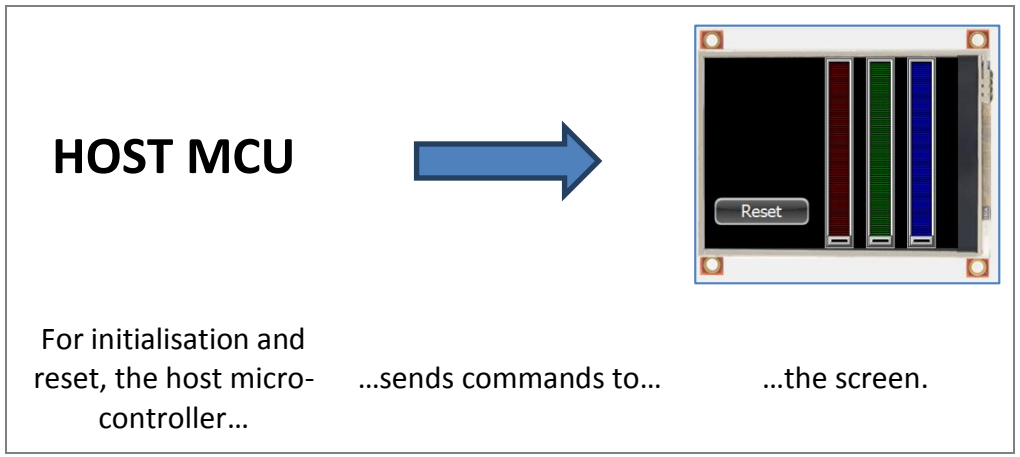
The host micro-controller translates the commands received and…

…manages…

**RGB LED**

…the red-green-blue LED.

In the specific cases of the initialisation and reset procedures, the host sends commands to the screen to place the sliders at mid-course.




Developing such an application is done in just four steps:

- Design the Screen
- Identify the Messages
- Write the Host Application
- Integrate the Screen and the Host

# Setup Procedure

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 Displays** (for Picaso)

or

**ViSi Genie Getting Started – First Project for Diablo16 Displays** (for Diablo16).

# Create a New Project

## 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

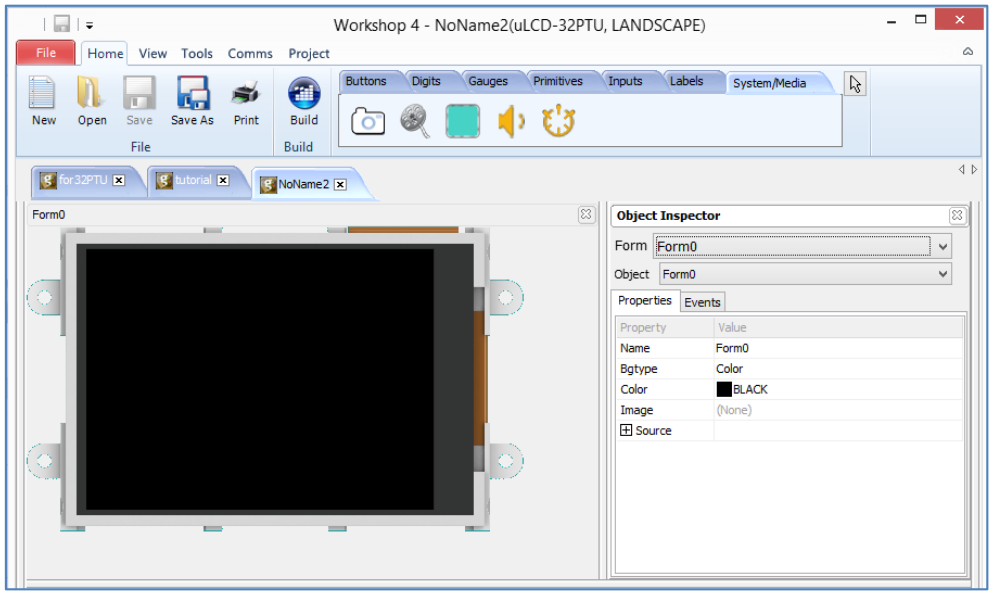
ViSi Genie Getting Started – First Project for Picaso Displays (for Picaso)

or

ViSi Genie Getting Started – First Project for Diablo16 Displays (for Diablo16).

# Design the Screen

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.

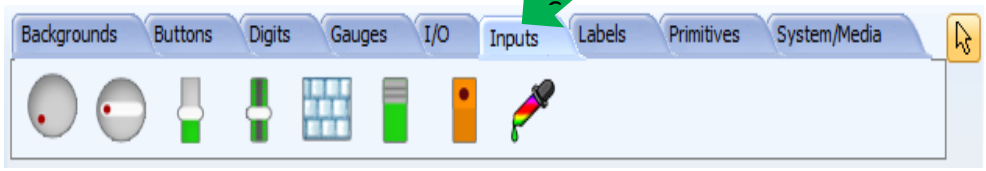



At the end of this section, the user will able to create a form with four objects. The final form will look like as shown below. The control panel includes three sliders, one per component, red, green or blue, with value included between 0 to 255, and one button Reset.


## Add the Sliders

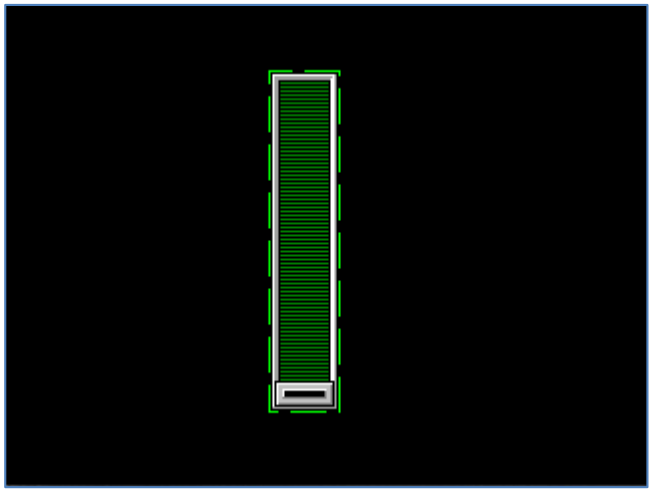
Select the **Home** menu to display the objects:


The **Slider** object is located on the Inputs pane:



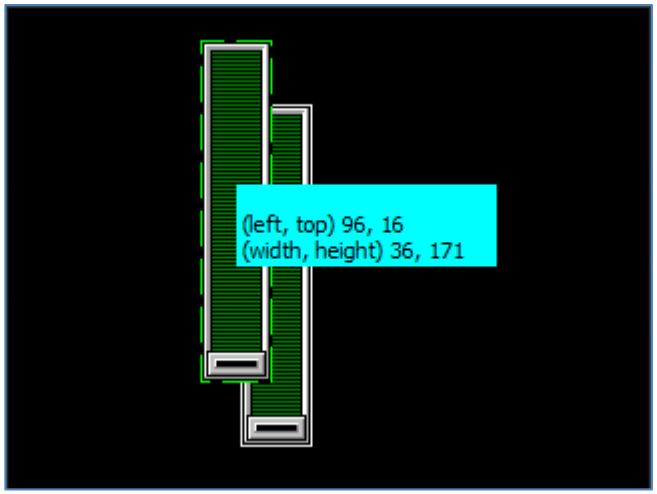

Click first on the **Slider** icon…

…and click on the WYSIWYG screen to place it.

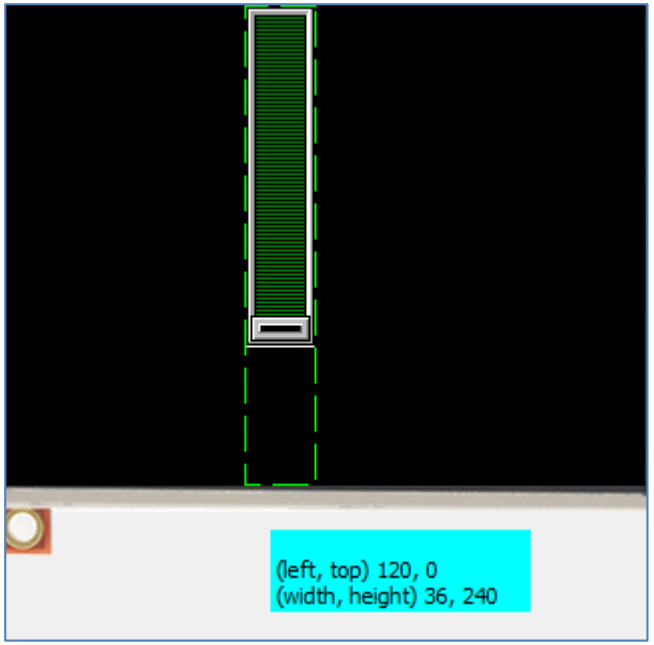


The **Slider0** is added.

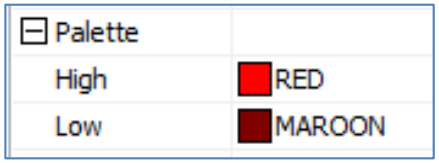
To move the slider, click and drag it to the desired position:



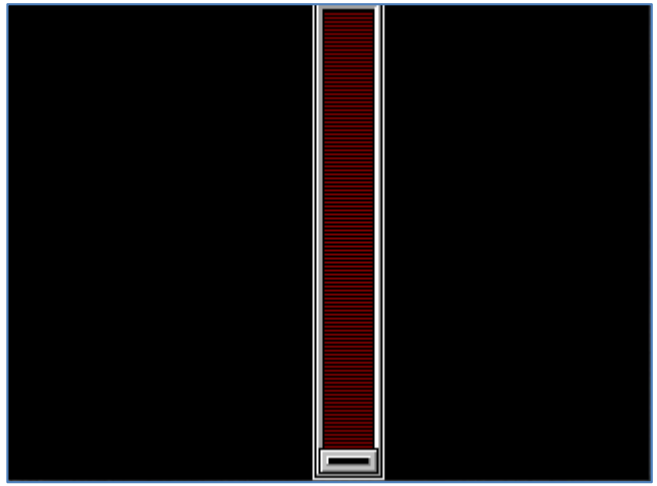

To resize the slider, click on the dotted green rectangle and move it to the desired size:




This is the slider for the red component. Change the **Palette** with **High** to Red and **Low** to Maroon:



The WISYWYG screen shows the final result:



Now, the maximum and the minimum values of the slider should be set to 255 and 0 respectively. Enter the value to **MaxValue** and **MinValue**:

| | |
|---|---|
| Maxvalue | 255 |
| Minvalue | 0 |

The slider for the red component is ready now.

Proceed the same way for the green and blue components:

- For the green component, change the **Palette** with **High** to Lime and **Low** to Green:

| Palette | |
|---|---|
| High | clLime |
| Low | clGreen |

- For the blue component, change the **Palette** with **High** to Aqua and **Low** to Blue:

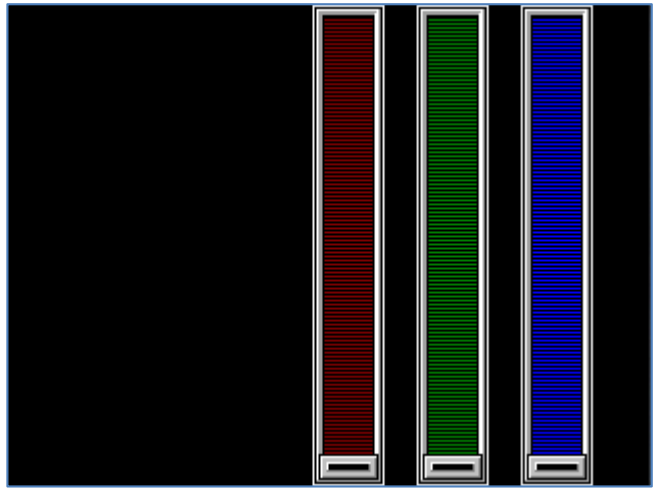
| Palette | |
|---|---|
| High | AQUA |
| Low | BLUE |

For both, enter 255 for **MaxValue** and 0 to **MinValue**:

| | |
|---|---|
| Maxvalue | 255 |
| Minvalue | 0 |

The **Slider1** and **Slider2** are added.

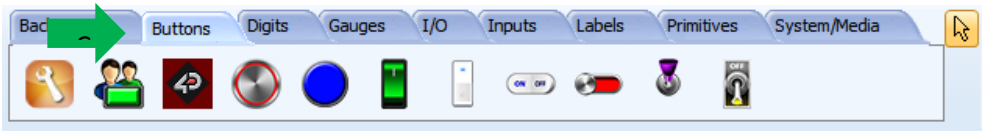
Final result looks like:



For more information on the Slider object, please refer to the application note ViSi-Genie Inputs.

## Add the Button

The **Button** object is located on the Buttons pane:

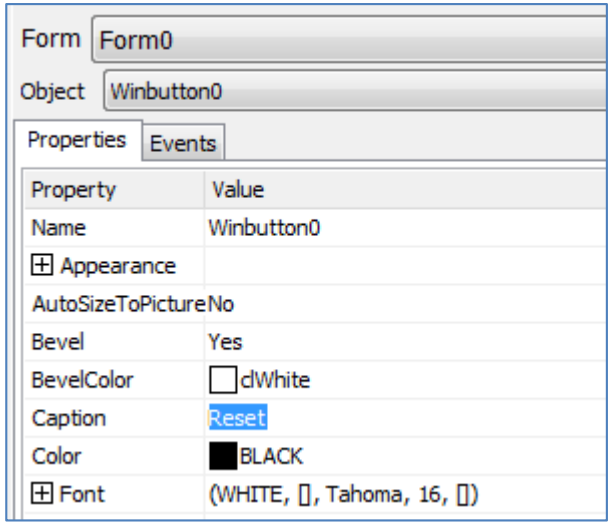



Click first on the **Button** icon…

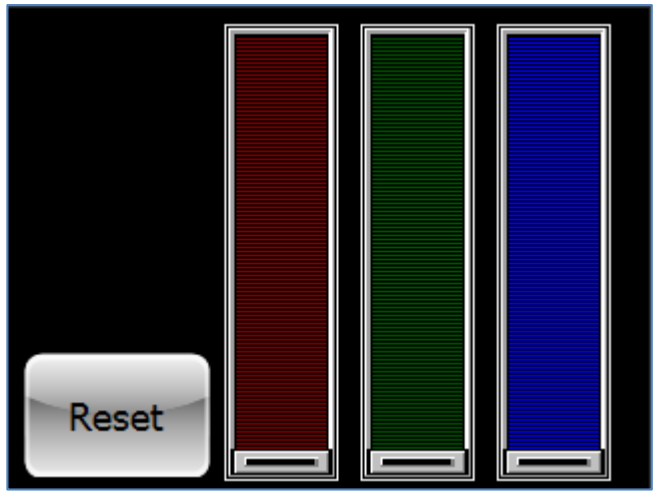
…and click on the WYSIWYG screen to place it.


The **WinButton0** is added.

Set the **Caption** to Reset and choose the appearance and the colour as you wish, here **Colour** set to black, **Font** to white and size to 16:




The control panel is ready now:




For more information on the Button object, please refer to the application note ViSi-Genie Advanced Buttons.

## Define the Actions

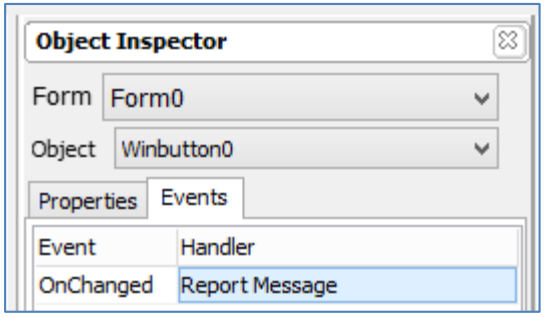
The interface is ready now but **WinButton0**, **Slider0**, **Slider1** and **Slider2** have no actions defined.

When pressed or changed, the objects raise events. All the events are going to be defined as **Message**.

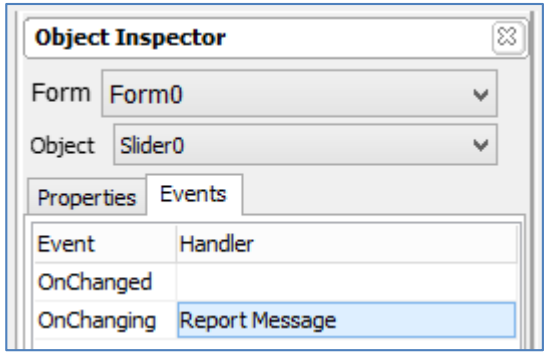
The message goes though the serial connection to the host and includes the object identification and the value.

When pressed, the **WinButton0** button…

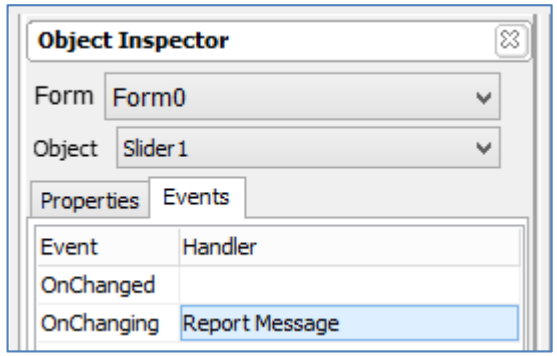

…raises the even **OnChanged** and sends a **Message**:



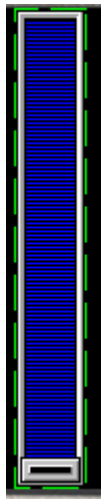
When being moved, the **Slider0** for the red component raises the event **OnChanging** and sends a **Message**:

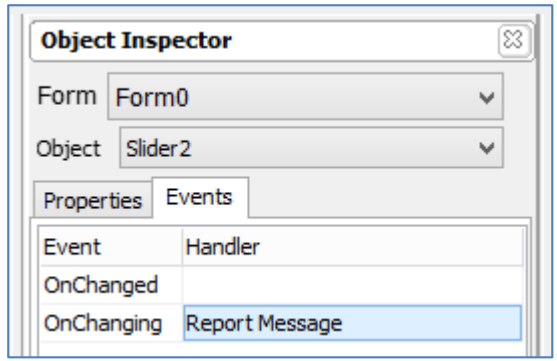



When being moved, the **Slider1** for the green component raises the event **OnChanging** and sends a **Message**:




When being moved, the **Slider2** for the blue component raises the event **OnChanging** and sends a **Message**:





# Build 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 Displays (for Picaso)

or

ViSi Genie Getting Started – First Project for Diablo16 Displays (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

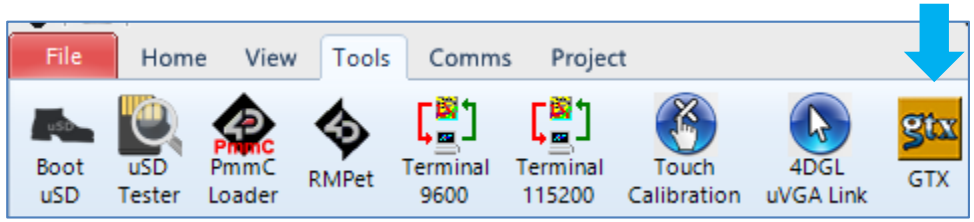
The screen is going to generate and send messages to the host. But what are those messages?

We need first to build and upload the project, then use either the debugger Codeless Executive Test Instrument or the terminal.

## Launch the Debugger

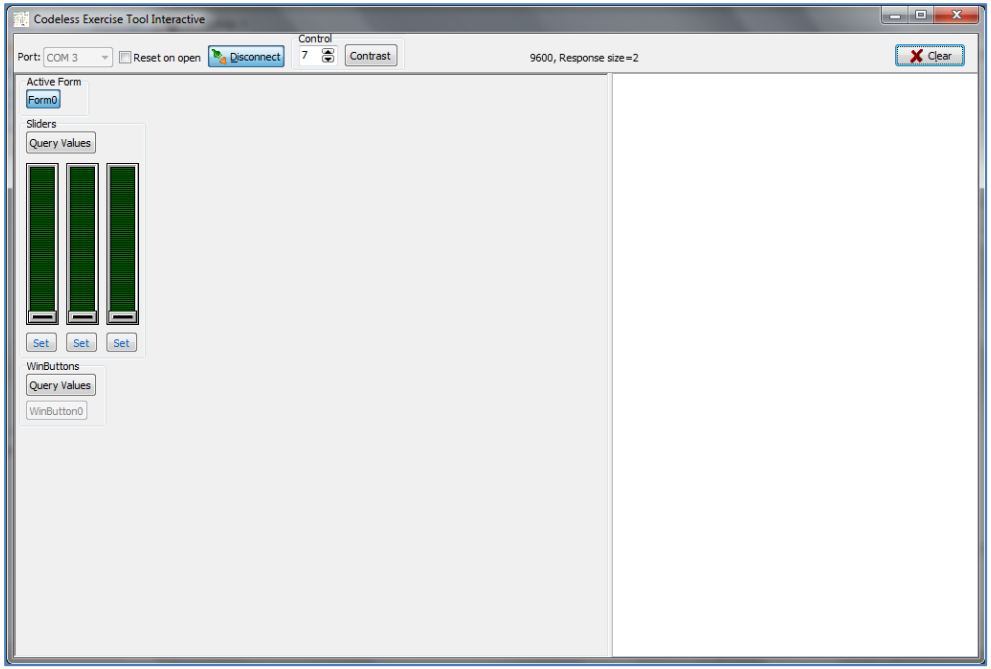
To debug the project, launch the **Genie Test Executor** or GTX.

Select the **Tools** menu…

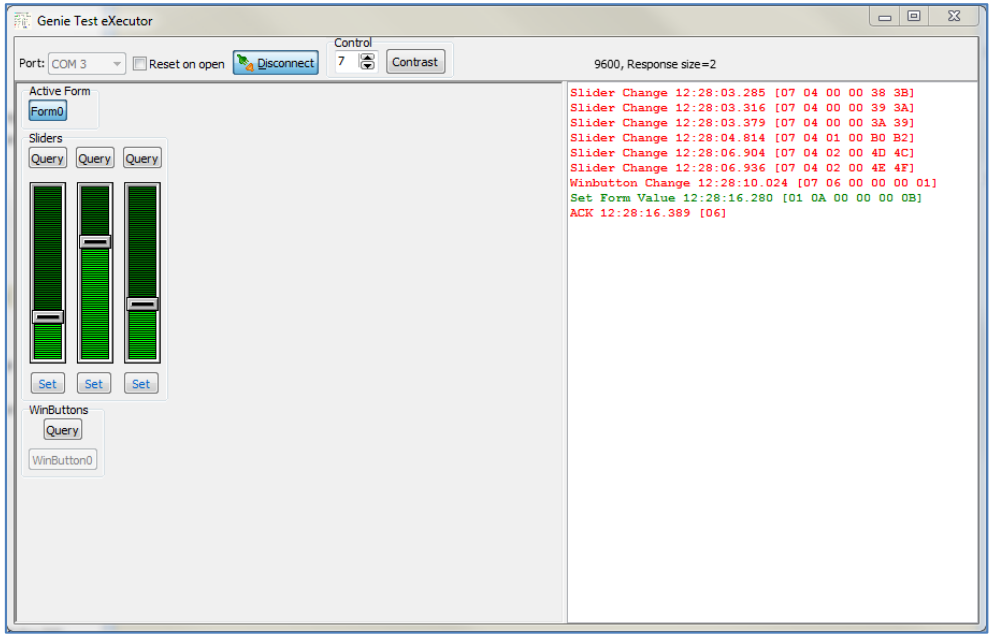



…and then click on the **GTX** button.

A new screen appears, with the form and objects we have defined previously:




Just try to move one of the sliders and press **Reset**. Messages are sent to the debugger:




The white area on the right displays

- in **green** the messages sent to the screen
- and in **red** the messages received from the screen:

```
Slider Change 12:28:03.285 [07 04 00 00 38 3B]
Slider Change 12:28:03.316 [07 04 00 00 39 3A]
Slider Change 12:28:03.379 [07 04 00 00 3A 39]
Slider Change 12:28:04.814 [07 04 01 00 B0 B2]
Slider Change 12:28:06.904 [07 04 02 00 4D 4C]
Slider Change 12:28:06.936 [07 04 02 00 4E 4F]
Winbutton Change 12:28:10.024 [07 06 00 00 00 01]
Set Form Value 12:28:16.280 [01 0A 00 00 00 0B]
ACK 12:28:16.389 [06]
```

All values are in hexadecimal.

The messages are formatted according to the following pattern:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 07 | 04 | 00 | 00 | 38 | 3B |
| REPORT_EVENT | Slider | Number 1 | 0x0038 | | |

## Message from a Slider

The messages from **Slider** objects are formatted according to the following pattern:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 07 | 04 | 00 | 00 | 38 | 3B |
| REPORT_EVENT | Slider | Number 1 | 0x0038 | | |

Each slider has its specific number:

- The red slider is number **00**, so the message from the red slider starts with **070400**
- The green slider is number **01**, so the message from the red slider starts with **070401**
- The blue slider is number **02**, so the message from the red slider starts with **070402**

The values allowed for the slider are between 0 and 255. So:

- minimum value sent on a message is 0 = **0000**
- maximum value sent on a message is 255 = **00FF**

The messages are sent when the slider is being moved, because **Message** has been defined for the event **OnChanging**, which is raised as many times and as long the slider is being moved.

## Message from the Button

The message is formatted according to the following pattern:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 07 | 06 | 00 | 00 | 38 | 01 |
| REPORT_EVENT | Button | Number 1 | 0x0038 | | |

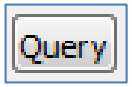
Each button has its specific number:

- the Reset button is number **00**, so the message from the Reset button starts with **070600**

Value is always **0000** and sent when the button is pressed.

## Message from Query Values

The Codeless Executive Test Instrument debugger features four Query buttons, one per object:



### Query for Sliders

Pressing **Query** on top of the leftmost slider…


…adds two lines to the message area on the right:

```
Request Slider Value 12:47:45.068 [00 04 00 04]
Slider Value 12:47:45.084 [05 04 00 00 3A 3B]
```

The first line in green is a command sent by the debugger to the screen, asking the values for the first slider:

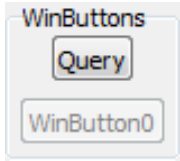
| Command | Object Type | Object Index | Checksum |
|---|---|---|---|
| 00 | 04 | 00 | 04 |
| READ_OBJ | Slider | Number 1 | |

The second line in red is the message sent back by the screen, with **0400** for **Slider0** or slider number 0 for the red component:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 05 | 04 | 00 | 00 | 3A | 3B |
| WRITE_ CONTRAST | Slider | Number 1 | 58 | | |

### Query for Button

Pressing **Query** on top of the button…




…adds two lines to the message area on the right:

```
Request Winbutton Value 12:52:00.115 [00 06 00 06]
Winbutton Value 12:52:00.146 [05 06 00 00 00 03]
```

The first line in green is a command sent by the debugger to the screen, asking the values for the button:

| Command | Object Type | Object Index | Checksum |
|---|---|---|---|
| 00 | 06 | 00 | 06 |
| READ_OBJ | Button | Number 1 | |

The second line in red is the message sent back by the screen:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 05 | 06 | 00 | 00 | 00 | 03 |
| WRITE_ CONTRAST | Button | Number 1 | Button released | | |

If the button is pressed…

```
Request Winbutton Value 12:58:52.006 [00 06 00 06]
Winbutton Value 12:58:52.021 [05 06 00 00 01 02]
```

…the value is

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 05 | 06 | 00 | 00 | 01 | 03 |
| WRITE_ CONTRAST | Button | Number 1 | Button pressed | | |

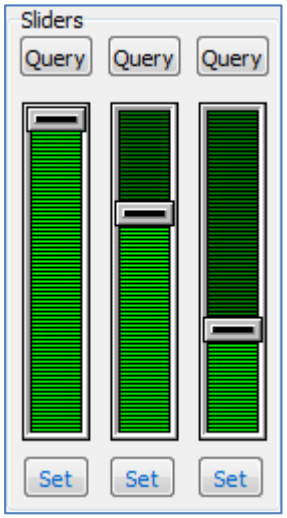
## Send Commands to the Screen

The screen sends messages to the host but the host can also send commands to the screen.

A first example is the **Query Values** seen before.

Another example is sending a value for a specific slider, for example for **Slider0**, the slider for the red component.

First set the value on the debugger, here close to the maximum…




…press **Set** below…


…and two lines are added to the message area on the right:

```
Set Slider Value 13:02:44.681 [01 04 00 00 FE FB]
ACK 13:02:44.712 [06]
```

The first line in green is a command:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| 01 | 04 | 00 | 00 | FE | FB |
| WRITE_OBJ | Slider | Number 1 | 254 | | |

The handler for the command is **0400**, or the object **Slider0**, with **04** for slider and **00** for the first slider number 1, and the value is **FF**.

In case the value is greater than 0xFF, for example 0x0102, the value is coded on the MSB and LSB:

| Command | Object Type | Object Index | Value MSB | Value LSB | Checksum |
|---|---|---|---|---|---|
| **01** | **04** | **00** | **01** | **02** | **06** |
| WRITE_OBJ | Slider | Number 1 | 257 | | |

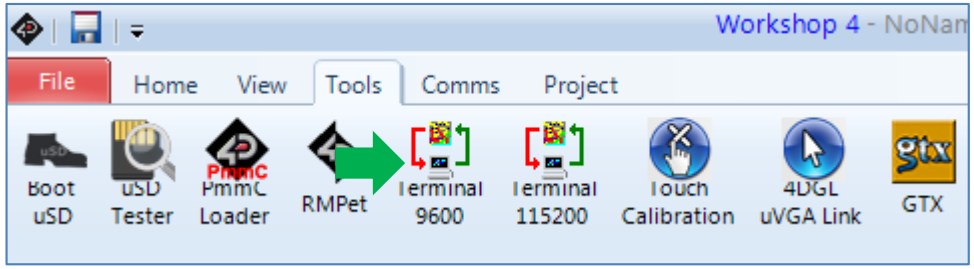
When a command is sent to the screen, the screen always answers by an acknowledgement:

- **06** for success,
- **15** for failure.

## Use the Terminal

As an alternative to GTX, the terminal is the second option to get the messages sent by the screen to the host.

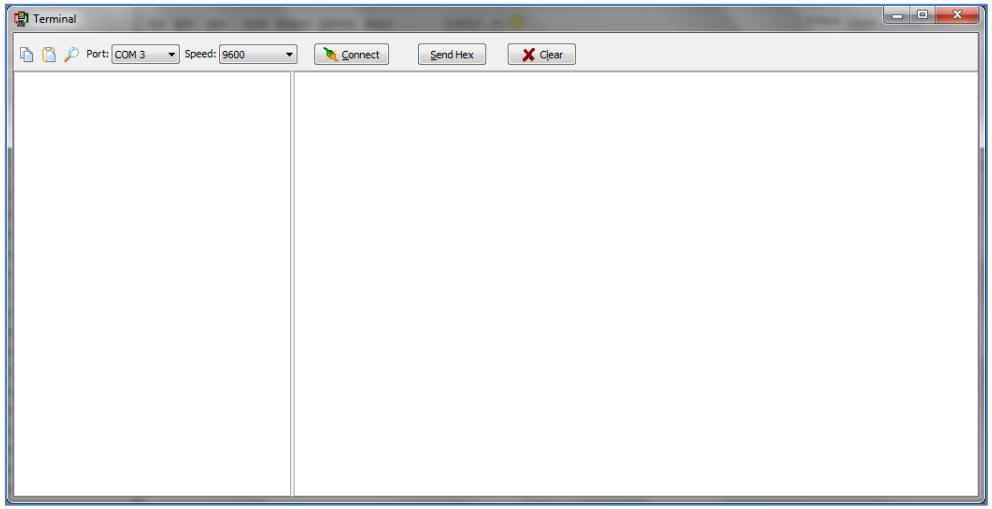
To launch the Terminal, select the **Tools** menu…



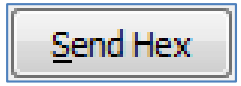

…and

- Click '**Terminal connect 9600**' to open the currently selected com port at 9600 baud in the Terminal program.

A new screen appears:

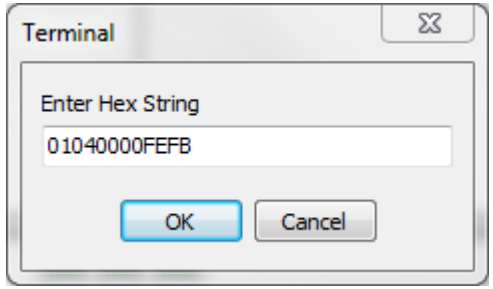



To send the commands on hexadecimal format, press

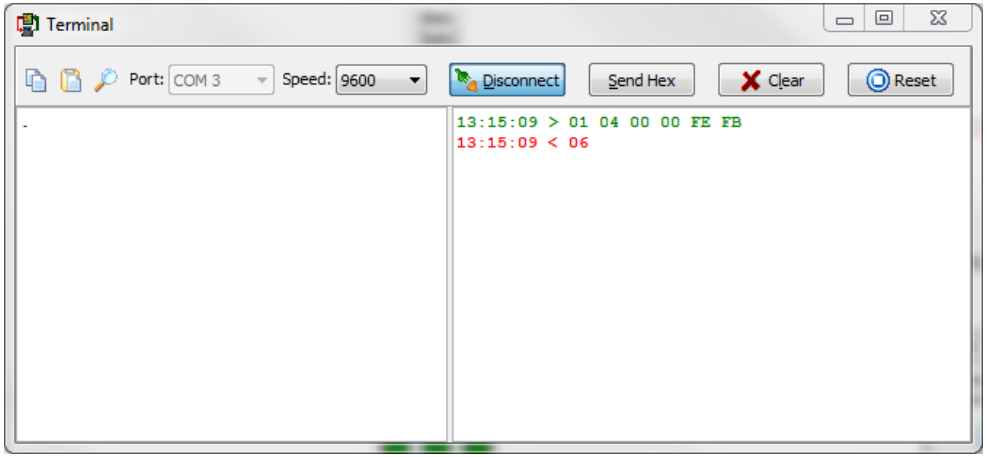



The commands sent by the host and the messages sent by the screen are the same as with the Codeless Executive Test Instrument debugger.

Enter the hexadecimal code in the **Terminal** window, including the final checksum:




Here, the command *Set Slider0 to value 0xFE* is sent to the screen module, or **01 04 00 00 FE FB** displayed in green on the terminal window.




And the screen answers with the **06** for successful acknowledgement, displayed on red on the terminal window.

# Program the Arduino Host

A thorough understanding of the application note **ViSi-Genie Connecting a 4D Display to an Arduino Host** is required before attempting to proceed further beyond this point. **ViSi-Genie Connecting a 4D Display to an Arduino Host** provides all the basic information that a user needs to be able to get started with ViSi-Genie and Arduino. The following is a list of the topics discussed in **ViSi-Genie Connecting a 4D Display to an Arduino Host**.

- How to download and install the ViSi-Genie-Arduino library
- How to open a serial port for communicating with the display and how to set the baud rate
- The genieAttachEventHandler() function
- How to reset the host and the display
- How to set the screen contrast
- How to send a text string
- The main loop
- Receiving data from the display
- The use of a non-blocking delay in the main loop
- How to change the status of an object
- How to know the status of an object
- The user's event handler

Discussion of any of these topics is avoided in other ViSi-Genie-Arduino application notes unless necessary. Users are encouraged to read **ViSi-Genie Connecting a 4D Display to an Arduino Host** first.

## Shorthand Version for Evaluating a Message

There is an alternative way of evaluating a message besides breaking it down successively. It can be thought of as the shorthand version since it is compact. Instead of using nested if statements, write

```
//if the message received is a REPORT EVENT form Slider 0 (shorthand)
if(genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_WINBUTTON, 0x00))
{
  genie.WriteObject(GENIE_OBJ_SLIDER, 0x00, 128);     // Set Slider0 val
  genie.WriteObject(GENIE_OBJ_SLIDER, 0x01, 128);     // Set Slider1 val
  genie.WriteObject(GENIE_OBJ_SLIDER, 0x02, 128);     // Set Slider2 val
  analogWrite(RED, 128);                             // Set red LED brig
  analogWrite(GREEN, 128);                           // Set green LED br
  analogWrite(BLUE, 128);                            // Set blue LED bri
}
```

The function

```
(genie.EventIs(&Event, GENIE_REPORT_EVENT,
                       GENIE_OBJ_WINBUTTON, 0x00)
```

compares the **cmd**, **object**, and **index** fields of **Event** to the parameters specified by the user. It returns TRUE if the all the fields match the caller's parameters, FALSE if any of the fields does not match the corresponding parameter. Hence, the statement

```
if(genie.EventIs(&Event, GENIE_REPORT_EVENT,
                       GENIE_OBJ_WINBUTTON, 0x00))
```

evaluates if **Event** is a REPORT EVENT message from Winbutton0.

## Change the Status of a Slider

To change the status of a slider, use the function indicated below.

```
//If the cmd received is from a Reported Event
if(Event.reportObject.cmd == GENIE_REPORT_EVENT)
{
  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
  {
    if (Event.reportObject.index == 0x00)
    genie.WriteObject(GENIE_OBJ_SLIDER, 0x00, 128);
    genie.WriteObject(GENIE_OBJ_SLIDER, 0x01, 128);
    genie.WriteObject(GENIE_OBJ_SLIDER, 0x02, 128);
    analogWrite(RED, 128);
    analogWrite(GREEN, 128);
    analogWrite(BLUE, 128);
  }
```

## Extract the Data of a Message

Since the values of the sliders are used to control the brightness levels of the LEDs, the MSB and LSB data bytes of a message are extracted and stored into a variable (**slider_val** in this example).

```
slider_val = (Event.reportObject.data_msb << 8)
+ Event.reportObject.data_lsb;
```

The value of **slider_val** is then passed to one of the LEDs, depending on which slider the message is from.

```
if (Event.reportObject.index == 0x00)
  analogWrite(RED, slider_val);

else if (Event.reportObject.index == 0x01)
  analogWrite(GREEN, slider_val);

else if (Event.reportObject.index == 0x02)
  analogWrite(BLUE, slider_val);
```

The statement

```
slider_val = (Event.reportObject.data_msb << 8)
+ Event.reportObject.data_lsb;
```

can also be written as

```
slider_val = genie.GetEventData(&Event);
```

The function returns the LSB and MSB of the event's data combined into a single variable.

# Connect the 4D Display Module to the Arduino Host

Refer to the section "**Connect the Display Module to the Arduino Host"** of the application note "**ViSi-Genie Connecting a 4D Display to an Arduino Host**" for the following topics:

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

## Connection of RGB LED

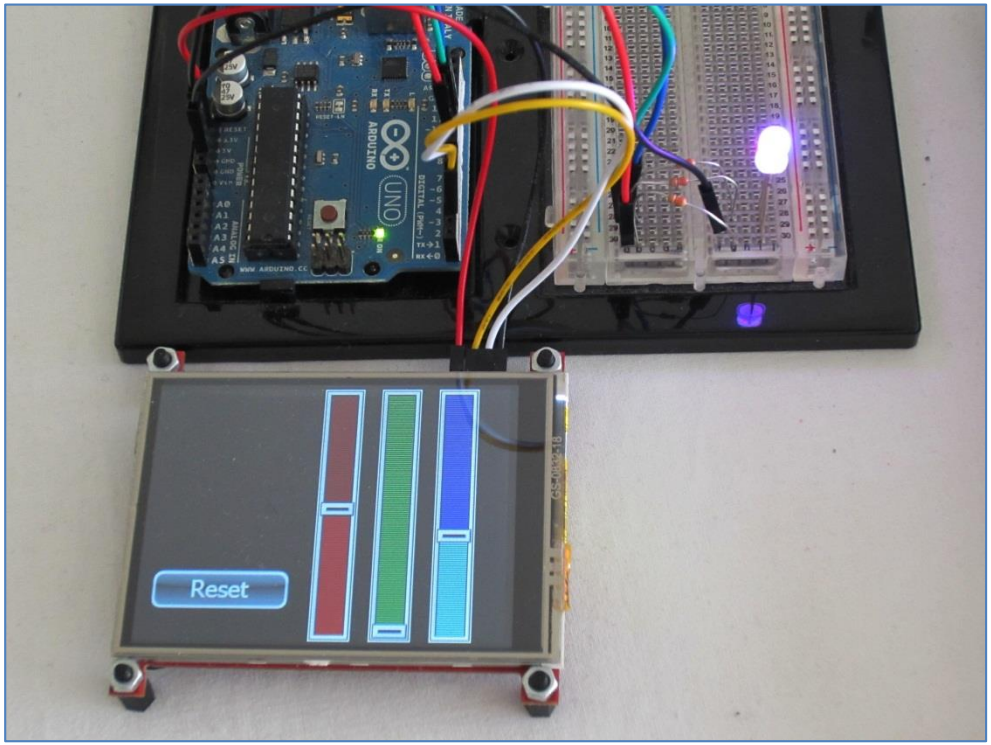
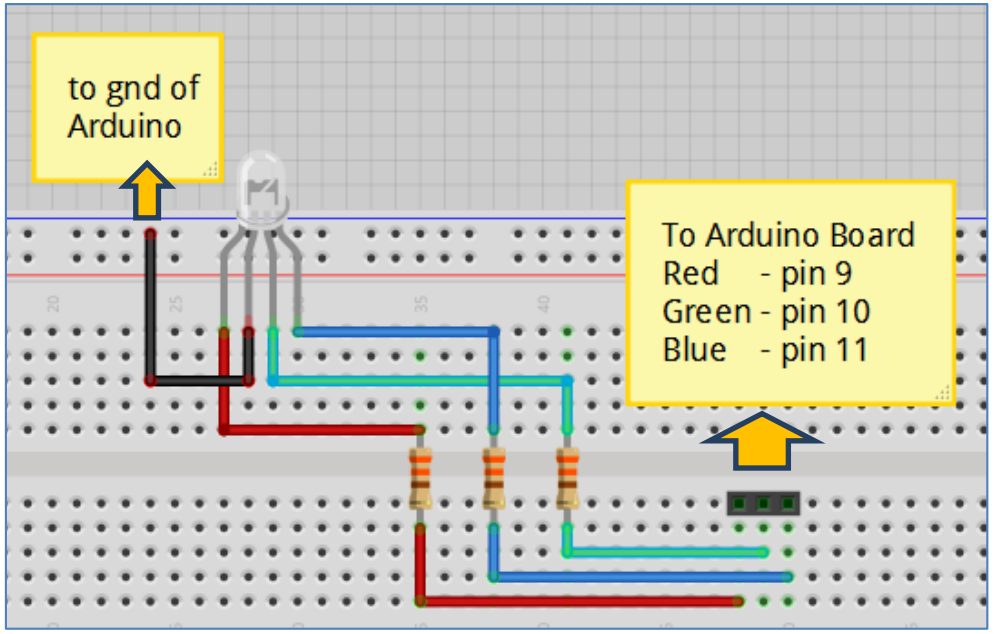
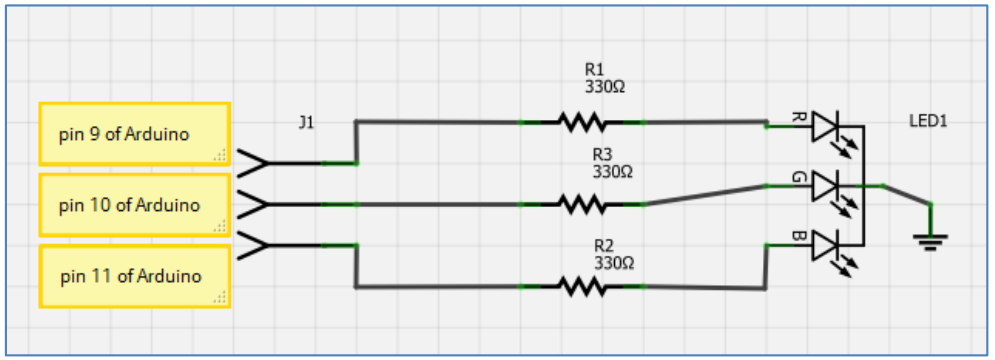



Photo of the actual project using jumper wires




Breadboard layout for the RGB LED (made with Fritzing)




Schematic diagram for the RGB LED (made with Fritzing)

# 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.