APPLICATION NOTES

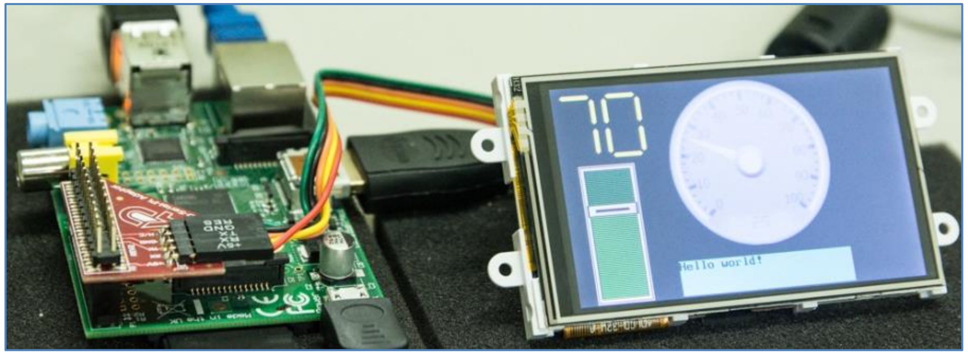
4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Genie – Connecting a 4D Display to the Raspberry Pi

DOCUMENT DATE: **1st MAY 2020**
DOCUMENT REVISION: **1.1**

Description

This Application Note explores the possibilities provided by ViSi-Genie to work with the Raspberry Pi. In the project developed in this document, a uLCD-32WPTU and a Raspberry Pi model B running on a Raspbian operating system are used. This should also work with other versions of Raspberry Pi and 4D Systems displays.



Before getting started, the following are required:

- Any of the following 4D Picaso touch display modules:

Product Name	Description
gen4-uLCD-24PT	2.4 inch resistive touch
gen4-uLCD-28PT	2.8 inch resistive touch
gen4-uLCD-32PT	3.2 inch resistive touch

and other superseded modules which support the ViSi Genie environment.

- The target module can also be a Diablo16 touch display

Product Name	Description
gen4-uLCD-24DT	2.4 inch resistive touch
gen4-uLCD-28DT	2.8 inch resistive touch
gen4-uLCD-32D(T/CT)	3.2 inch resistive or capacitive touch
gen4-uLCD-35D(T/CT)	3.5 inch resistive or capacitive touch
gen4-uLCD-43D(T/CT)	4.3 inch resistive or capacitive touch
gen4-uLCD-50D(T/CT)	5.0 inch resistive or capacitive touch
gen4-uLCD-70D(T/CT)	7.0 inch resistive or capacitive touch

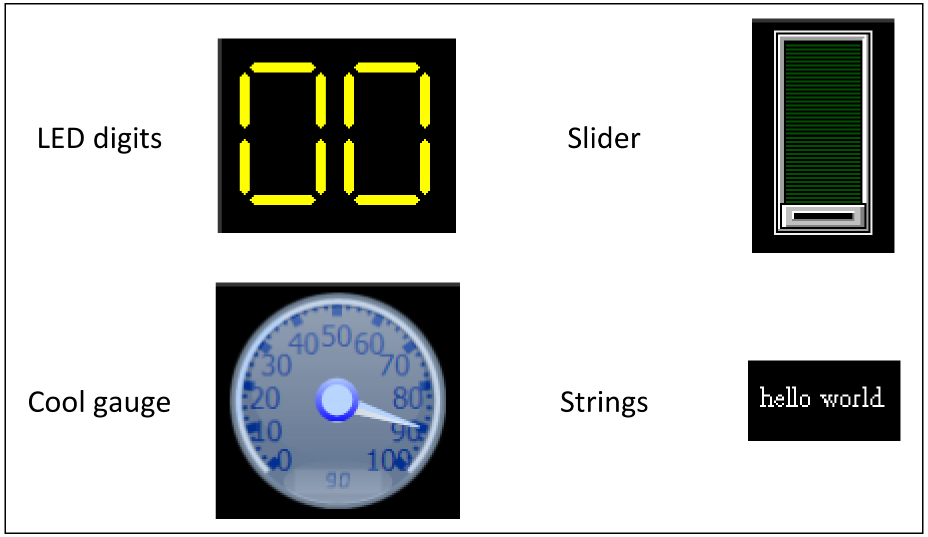
Visit www.4dsystems.com.au/products to see the latest touch display module products that use the Diablo16 processor. The display module used in this application note is the uLCD-32WPTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- 4D Programming Cable or µUSB-PA5
- micro-SD (µSD) memory card
- Workshop4 IDE (installed according to the installation document)
- A working Raspberry Pi (with Raspbian OS – this will need another µSD card)
- Ethernet (network) cable / Wi-Fi dongle – network connection is optional but recommended for easier installation of libraries
- A 4D Raspberry Pi Adaptor **or** connecting wires

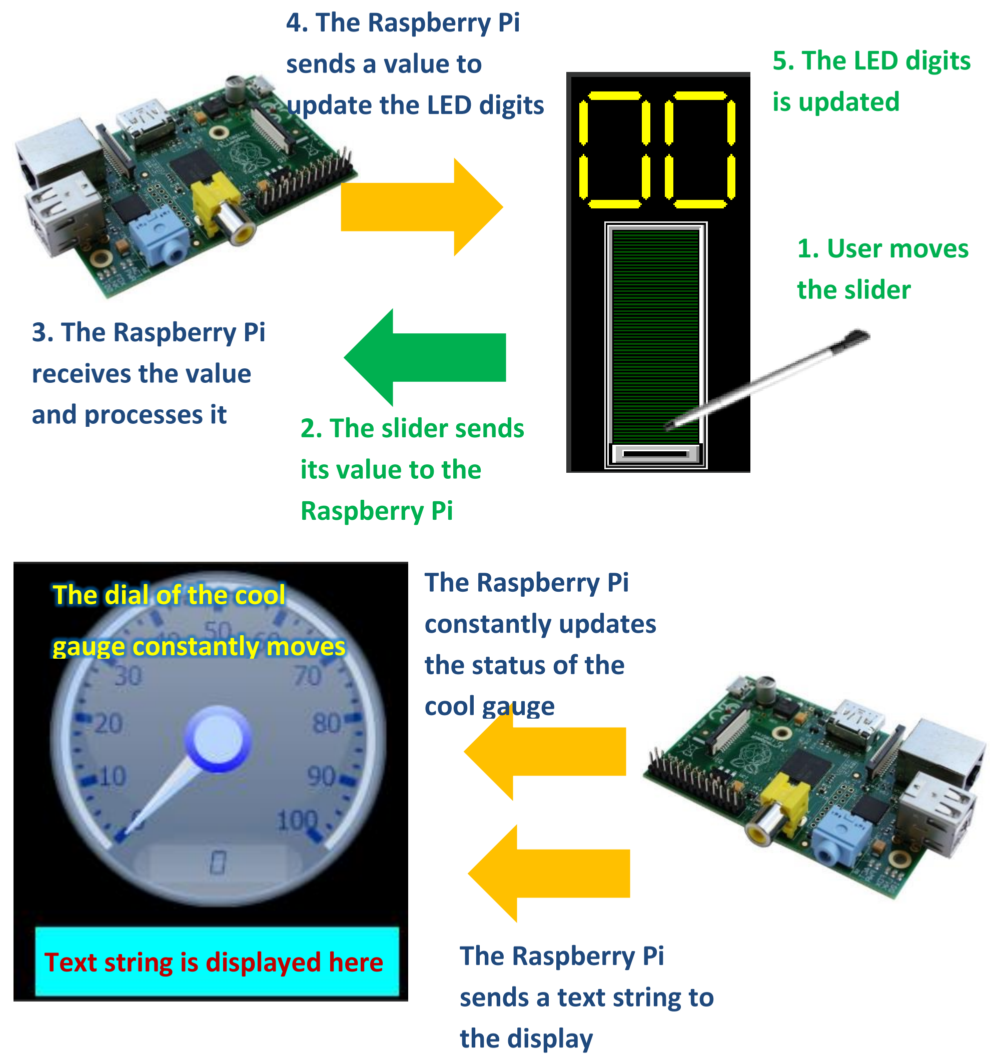
Content

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). The user can simply click on the desired widget to select it and click on the simulated display to place the widget. The following are examples of widgets or objects used in this application note.



The purpose of this application note is to introduce the use of the basic ViSi-Genie-Raspberry-Pi library functions. The application consists of a 4D Picaso display module displaying four objects – a LED digits, a slider, a cool gauge, and a text string. These objects interact with the Raspberry Pi in a manner illustrated below.



First the user creates a ViSi Genie program in the 4D Workshop IDE and downloads it to a 4D display module. On the other hand, a C program is created and is made to run on the Raspberry Pi. A library for the Raspberry Pi is provided by 4D Systems.

Setup Procedure

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

https://github.com/4dsystems/ViSi-Genie-RaspPi-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).

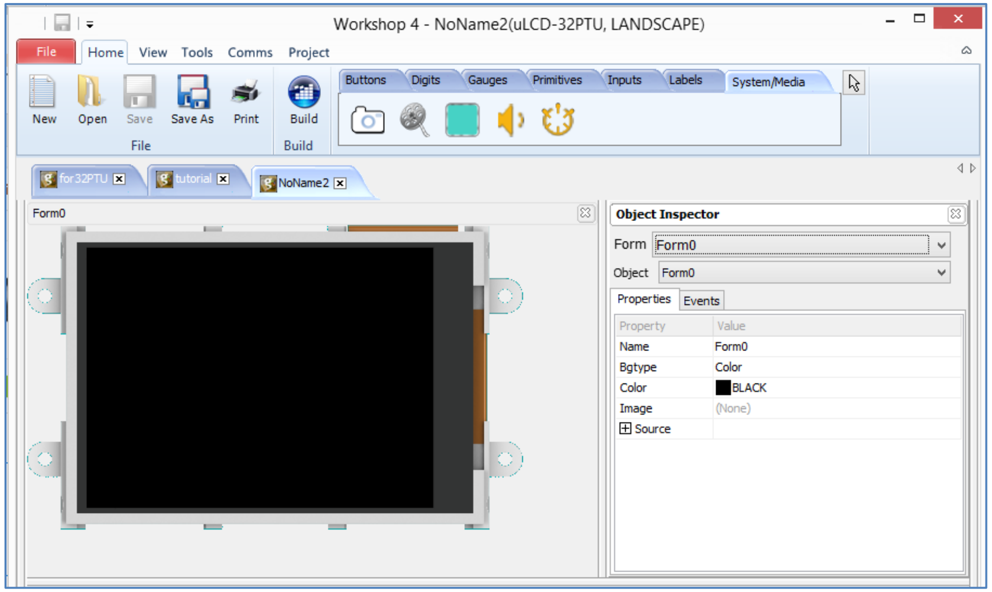
Create a New Project

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.

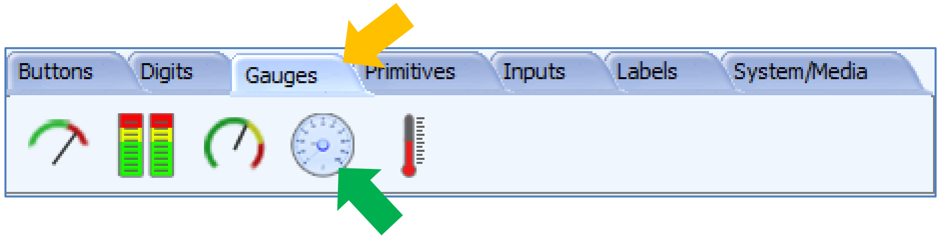


At the end of this section, the user will able to create a form with four objects. The final form will look like as shown below, with the labels excluded.

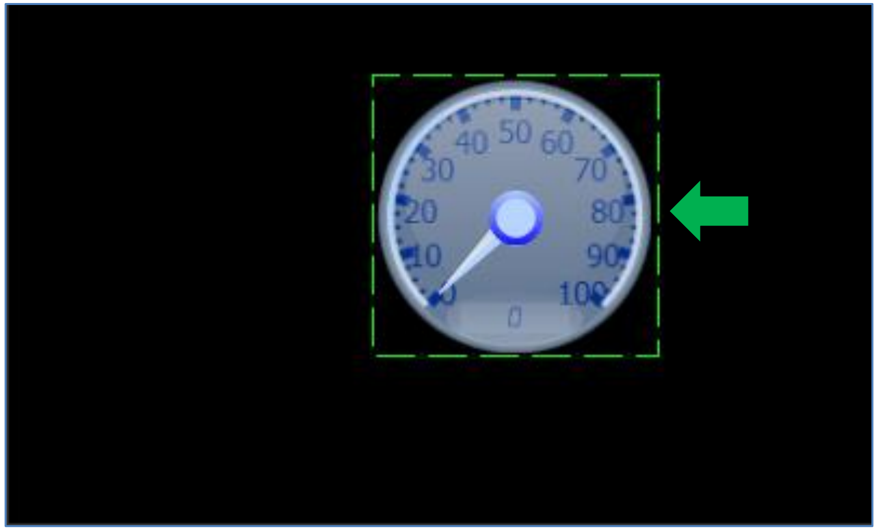
The procedure for adding each of these objects will now be discussed.

Add a Cool Gauge

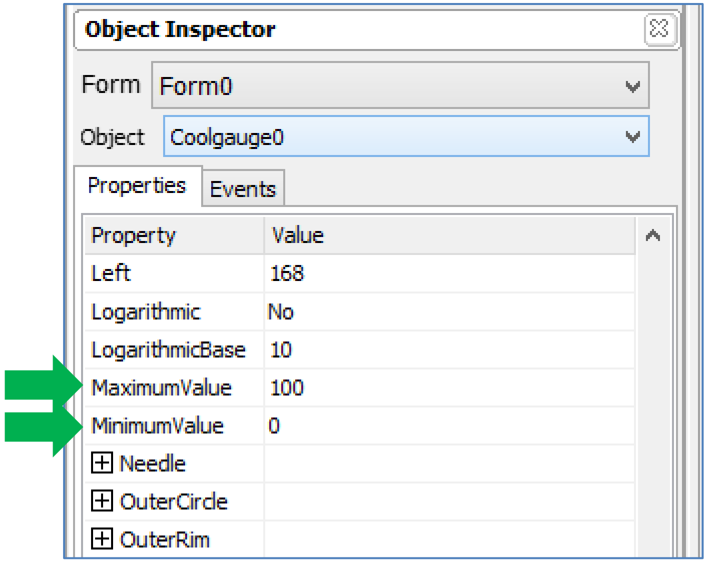
The cool gauge constantly receives values from the Raspberry Pi. The dial of the cool gauge will constantly move to correspond with the received values. To add a cool gauge, go to the **Gauges** pane then click on the **cool gauge** icon.



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



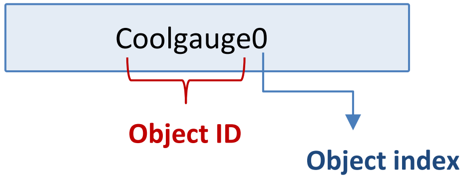
The object can be dragged to any desired location and resized to the desired dimensions. The **Object Inspector** on the right part of the screen displays all the properties of the newly created cool gauge object named **Coolgauge0**.



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 Raspberry Pi to the cool gauge. To know more about gauges, refer to ViSi-Genie Gauges.

Naming of Objects

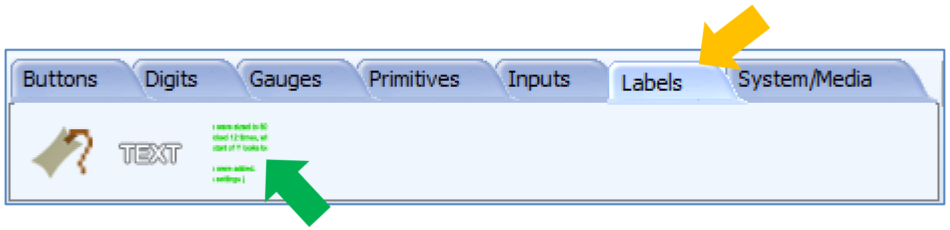
Naming is important to differentiate between objects of the same kind. For instance, suppose the user adds another cool gauge object to the WYSIWYG screen. This object will be given the name Coolgauge1 – it being the second cool gauge in the program. The third cool gauge will be given the name Coolgauge2, 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 the Raspberry Pi, 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.

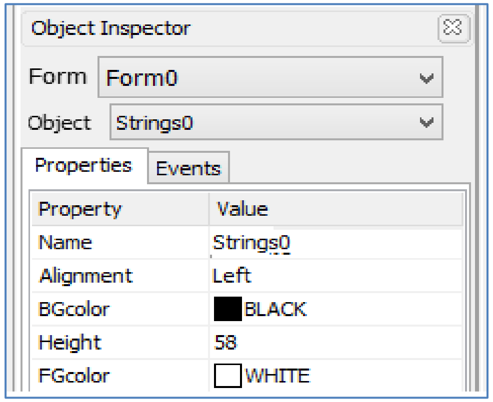
Add a Text String

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

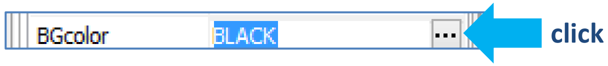


Click on the **WYSIWYG** screen to put the string in place. Again, the WYSIWYG screen simulates the actual appearance of the display module screen.

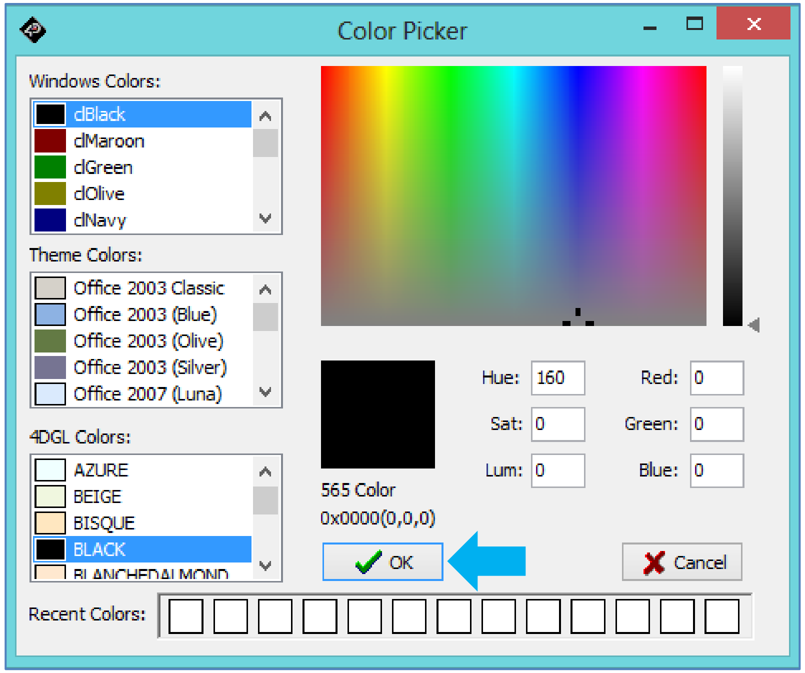
The area inside the red box will be the space occupied by the text string to be displayed. The object can be dragged to any desired location and resized to the desired dimensions. The **Object Inspector** on the right part of the screen displays all properties of the newly created strings object named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



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

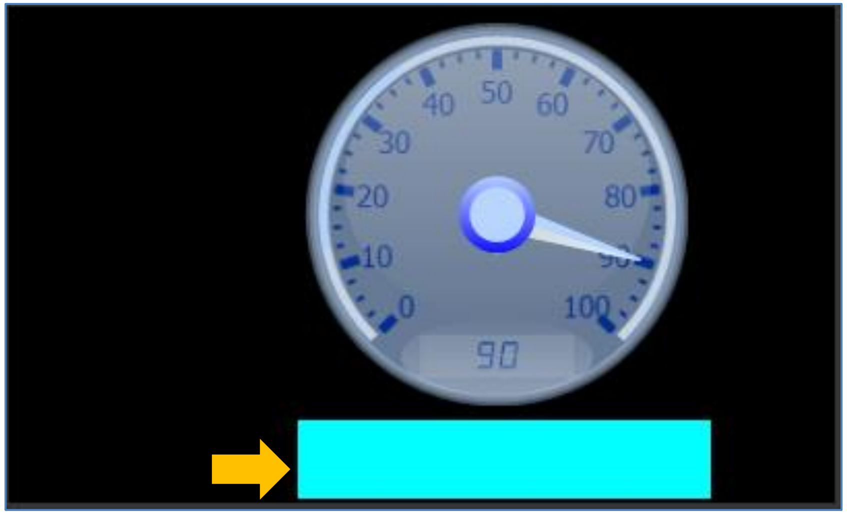


Choose the desired colour and click OK.



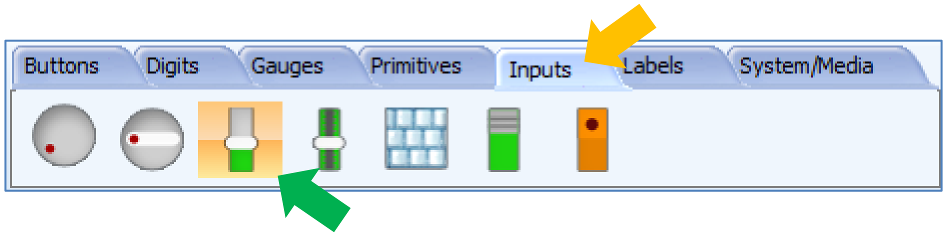
Do the same for the foreground colour.

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



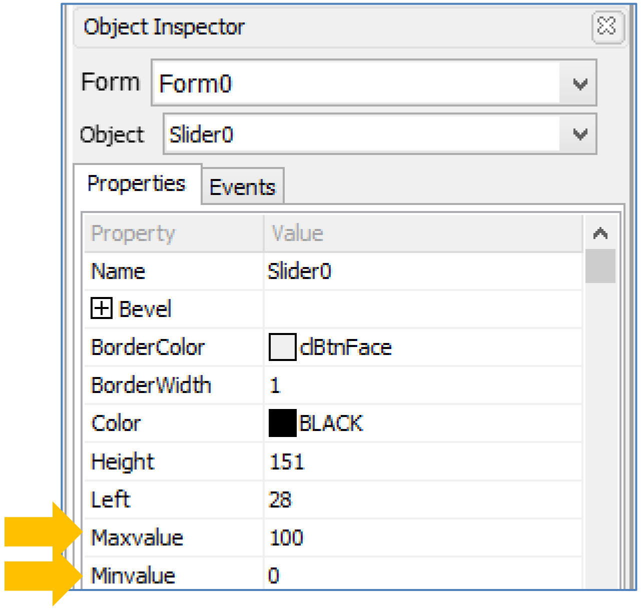
Add a Slider

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



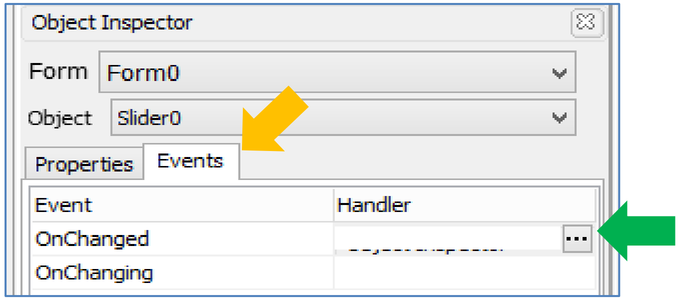
Click on the WYSIWYG screen to place the slider object. Drag the object to any desired location.

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

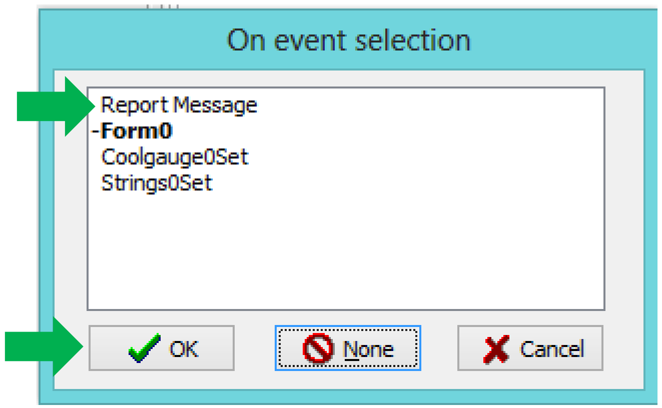


Report Event

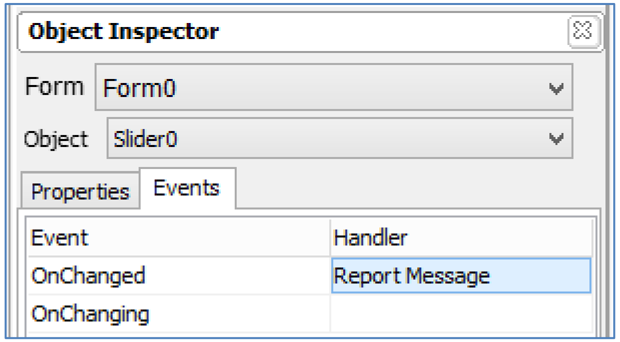
An object can report its current status independently without being polled by the Raspberry Pi. A slider, for example, can be configured to report its current status to the Raspberry Pi each time it is moved. To do this, click on the Events tab in the object inspector and click on the ... symbol in the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



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



The Events pane is now updated.

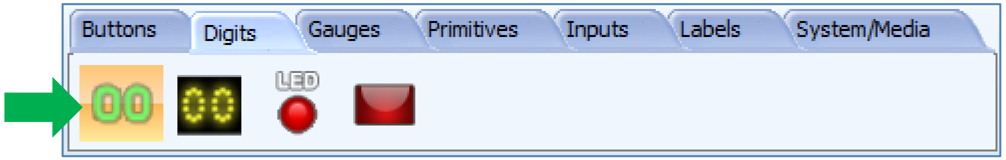


Now every time the slider is moved or its status has changed, it sends a **message** to the Raspberry Pi. To be more exact, the slider will send a report message when the stylus or finger moving it is lifted off the screen. Selecting the **OnChanging** event, on the other hand, causes the slider to send 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 Raspberry Pi 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].

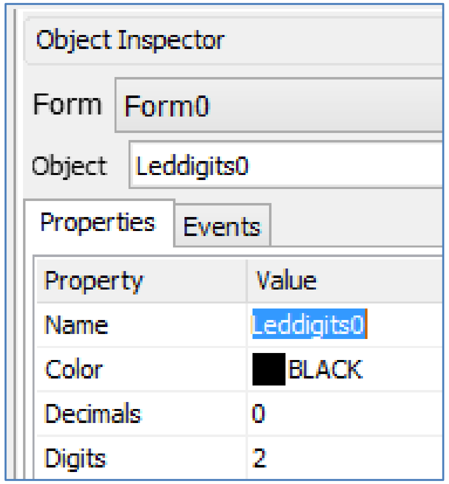
Add a LED Digits Object

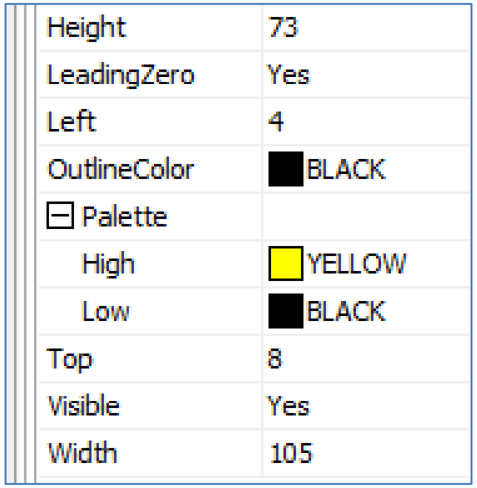
The **LED digits** object will display values received from the Raspberry Pi. To add a LED digits object, go to the **Digits** pane and select the first icon.



Click on the WYSIWYG screen to place it.

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





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

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

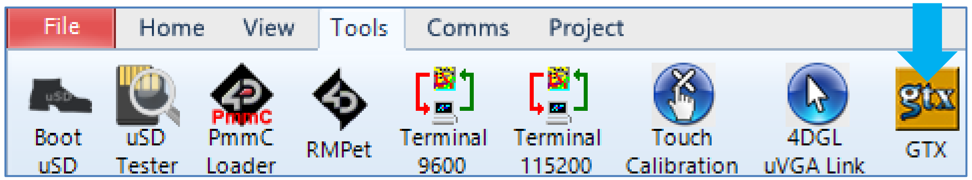
The display module is going to generate and send messages to the host or the Raspberry Pi. This section explains to the user how to interpret these messages. An understanding of this section is necessary for the user to be able to properly program the Raspberry Pi. The ViSi-Genie User Reference Manual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

Use the GTX Tool to Analyse the Messages

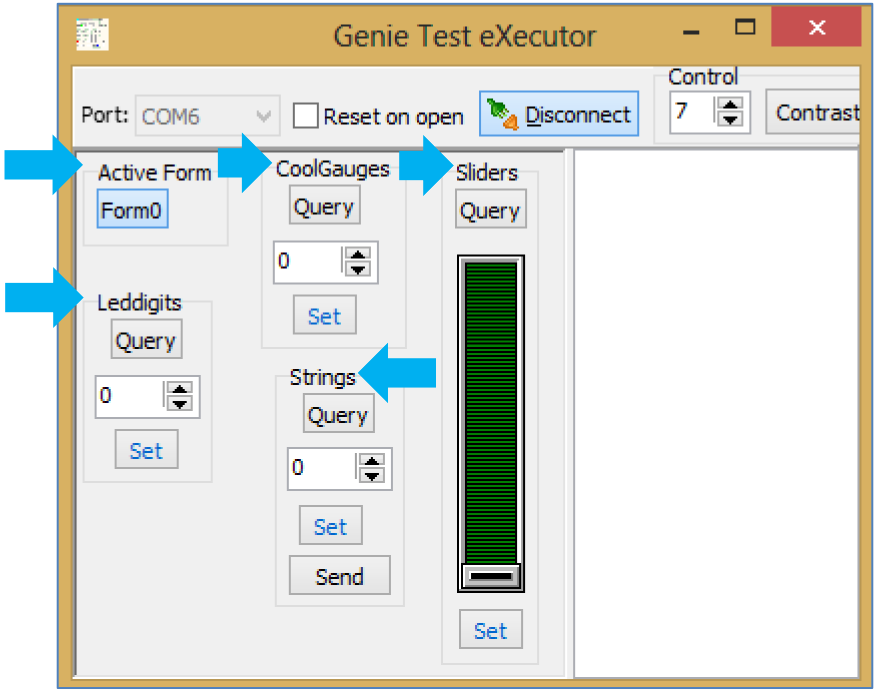
Using the GTX or **Genie Test eXecutor** tool is the first option to get the messages sent by the screen to the Raspberry Pi. Here the PC will be the host, instead of the Raspberry Pi. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to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

Launch the GTX Tool

Under tool menu click on the GTX tool button.



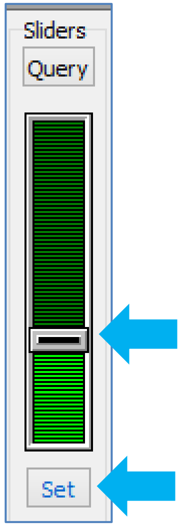
A new window appears, with the form and objects created previously.



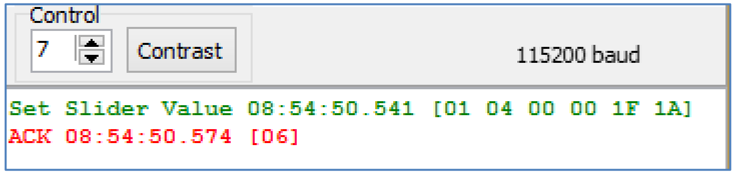
The Slider Object

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.



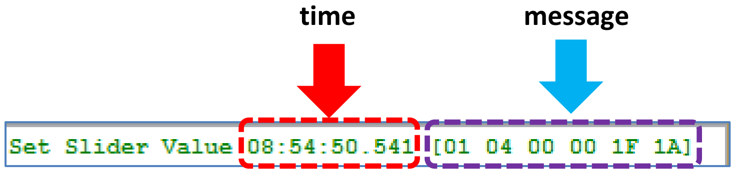
Also, messages are sent to and received from the display module.



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 (the Raspberry Pi or the PC) 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 Raspberry Pi thru the ViSi-Genie-RaspPi-Library 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.

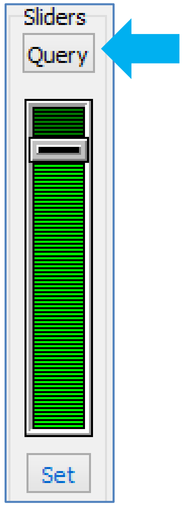
```
Slider Change 09:02:36.533 [07 04 00 00 42 41]
```

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 Raspberry Pi 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 Raspberry Pi or the PC 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. The ViSi-Genie program developed in this application note simply waits for **REPORT_EVENT** messages. The process of polling the display and receiving **REPORT_OBJ** messages will be covered in a separate application note.

Experimentation with the different objects using the GTX tool is now left to the user as an exercise. The following tables are shown below as a 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.

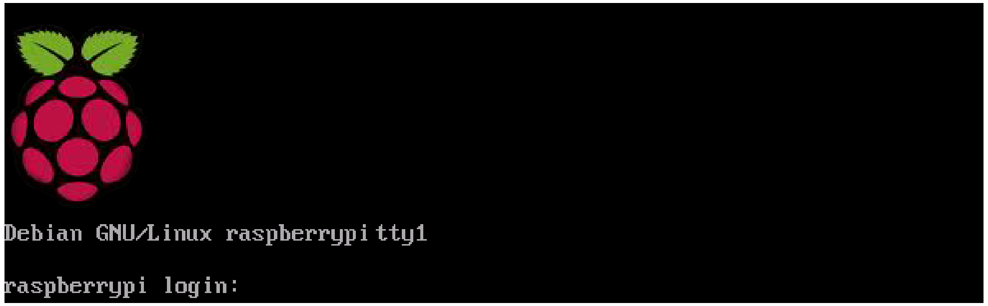
Program the Raspberry Pi

This section discusses how to open the C source code demo in the Raspberry Pi and how to compile it and make it work with the display module. It is assumed that the user has the Raspbian OS installed on an SD card or a µSD card with an SD card adaptor for the Raspberry Pi and has a basic understanding of the C programming language. Inexperienced users may need to frequently refer to C programming tutorial sites for more information. For first-time Raspberry Pi users, the following pages are recommended:

http://www.raspberrypi.org/quick-start-guide
http://www.raspberrypi.org/faqs
http://www.raspberrypi.org/

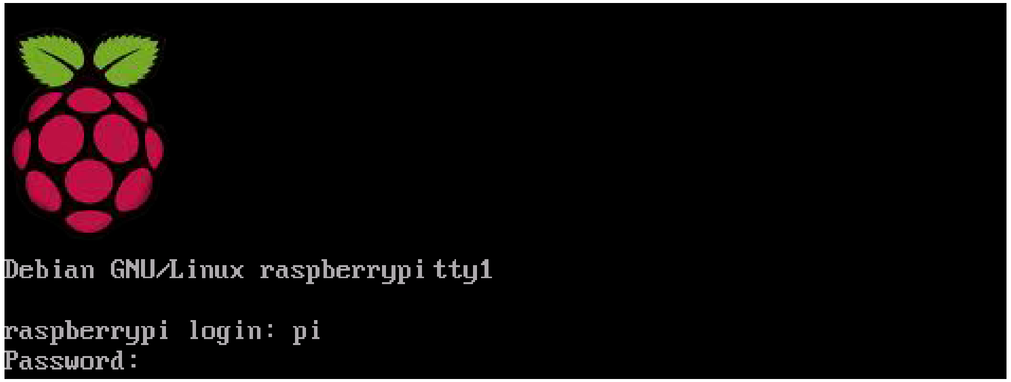
Power up the Raspberry Pi

During start up the terminal prompts for the log in details.

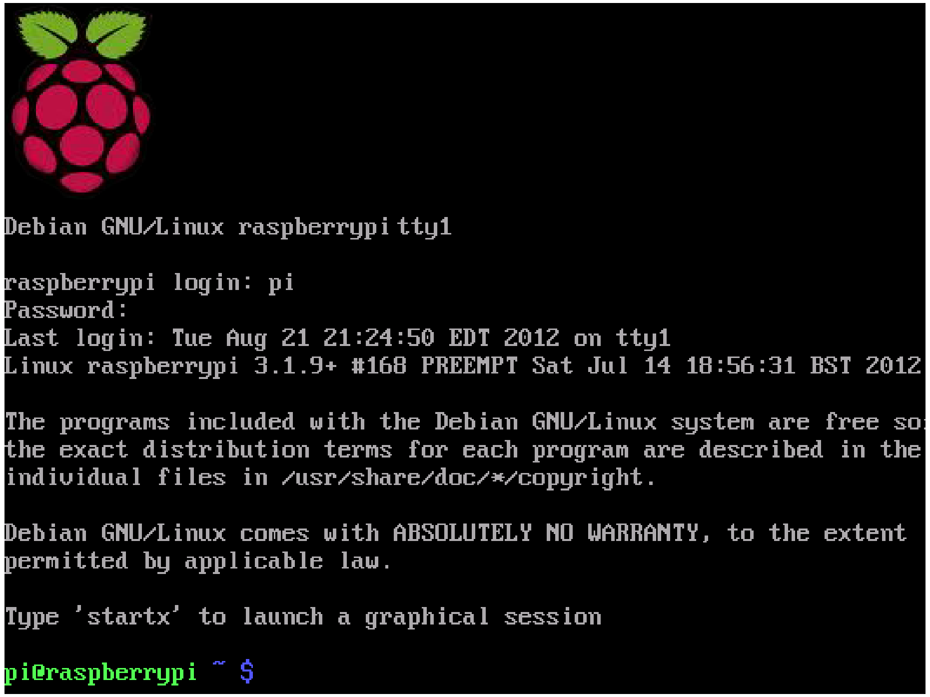


By default the username is "**pi**".

The default password is "**raspberry**". **When typing the password, the characters are not displayed, so type carefully.**



After a successful log in:



Type "**startx**" to proceed. Use a web browser (Midori for example) to download and install the ViSi-Genie-RaspPi library. The Raspberry Pi needs to be connected to the Internet for this.

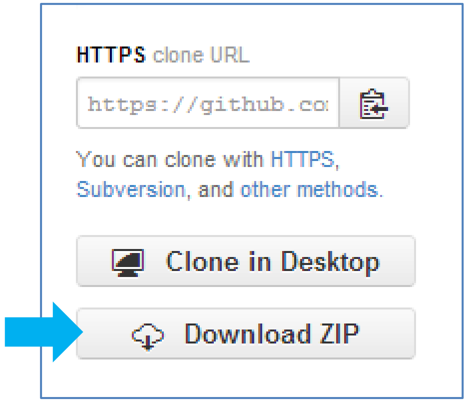
If not connected to the Internet, it is also possible to put all the necessary files into a USB flash drive and copy them to the Raspberry Pi.

Download and Install the ViSi-Genie-RaspPi Library

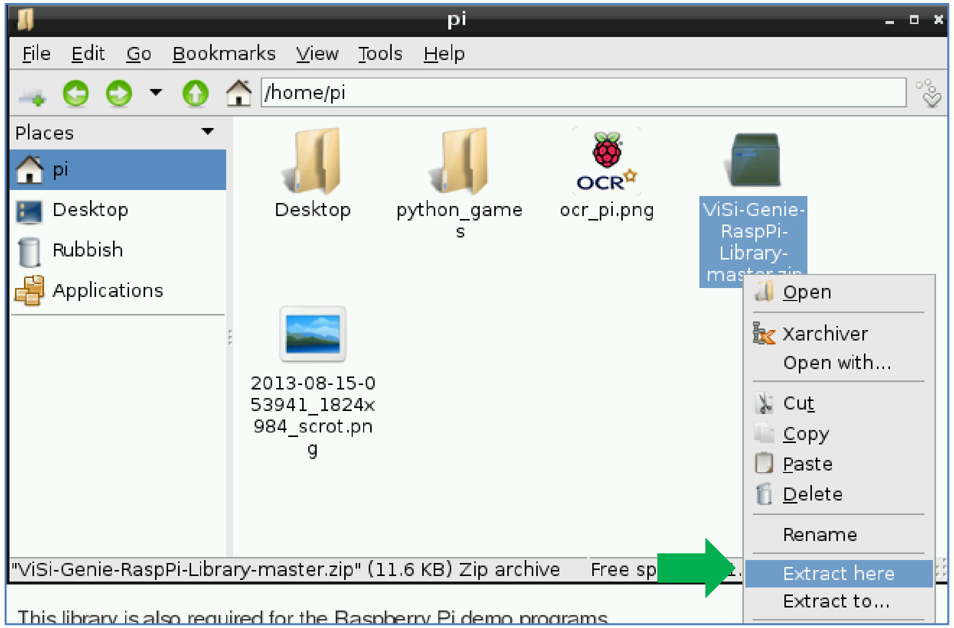
The ViSi-Genie-RaspPi-Library files are located here:

https://github.com/4dsystems/ViSi-Genie-RaspPi-Library

On the right side of the github page, click on the Download ZIP button.

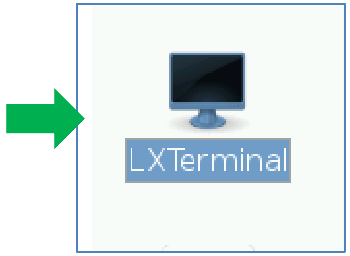


Extract the library files to a folder.

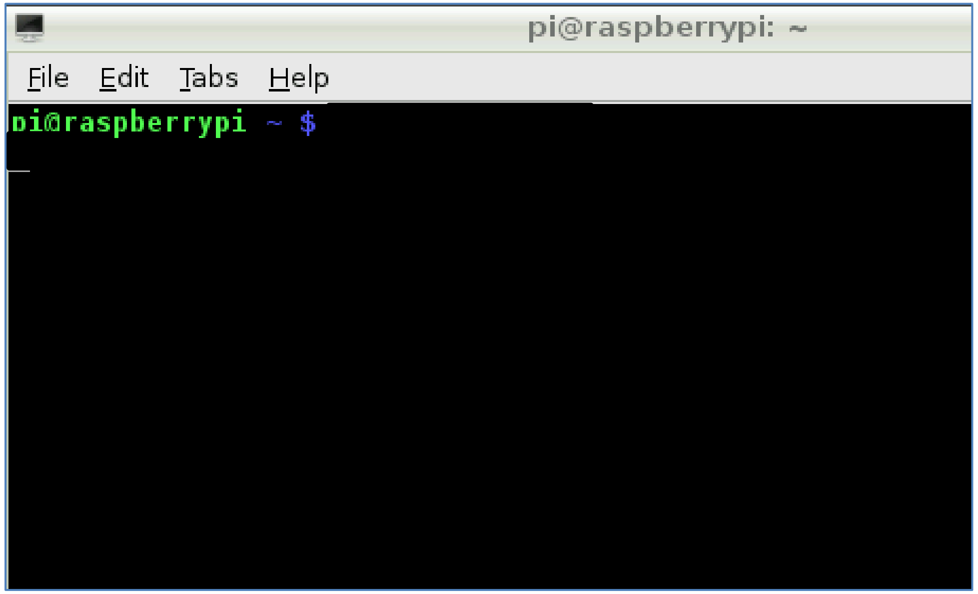


Take note of the location of the files. Here the files are located inside the folder "ViSi-Genie-RaspPi-Library-master".

On the desktop, double-click on the LXTerminal icon.



The terminal window appears.



Navigate to the folder where the library files are located by typing the command as shown below.

```
pi@raspberrypi ~ $ cd ViSi-Genie-RaspPi-Library-master
```

To compile the source code type **make**.

```
pi@raspberrypi ~/ViSi-Genie-RaspPi-Library-master $ make
```

The source code now compiles.

```
pi@raspberrypi ~/ViSi-Genie-RaspPi-Library-master $ make
[Compile] geniePi.c
[Link (Dynamic)]
```

To install, type **sudo make install**.

```
~/ViSi-Genie-RaspPi-Library-master $ sudo make install
```

The library is now installed.

```
pi@raspberrypi ~/ViSi-Genie-RaspPi-Library-master $ sudo make install
[Install]
```

Download and Install the wiringPi Library

Installation of the wiringPi library is optional but is needed to run some of the ViSi-Genie-Raspberry-Pi demo files in the 4D Systems github repository page. Installation of this library is also recommended since future application notes may require its use. Instructions for installing the library can be found on the same page where the ViSi-Genie-RaspPi-Library files are located: https://github.com/4dsystems/ViSi-Genie-RaspPi-Library. To know more about the wiringPi library and the detailed steps on how to install it, go to https://projects.drogon.net/raspberry-pi/wiringpi/.

Disable Serial Login Shell

The Raspberry Pi will use the serial port to talk to the display. However, by default it is used for a login shell. This needs to be disabled to be able to use the serial port properly.

Launch the tool from the terminal by using the command:

```
sudo raspi-config
```

```
pi@raspberrypi:~ $ sudo raspi-config
```

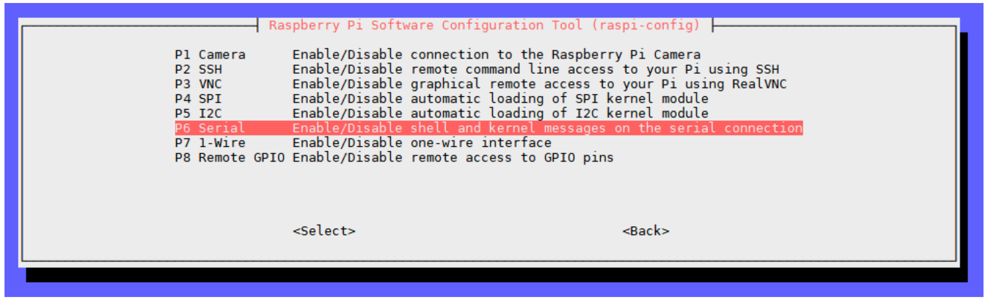
Raspberry Pi Software Configuration Tool opens.

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the default user (pi)
2 Hostname             Set the visible name for this Pi on a network
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

          <Select>                         <Finish>
```

Select **Interfacing Options** and press **Enter**

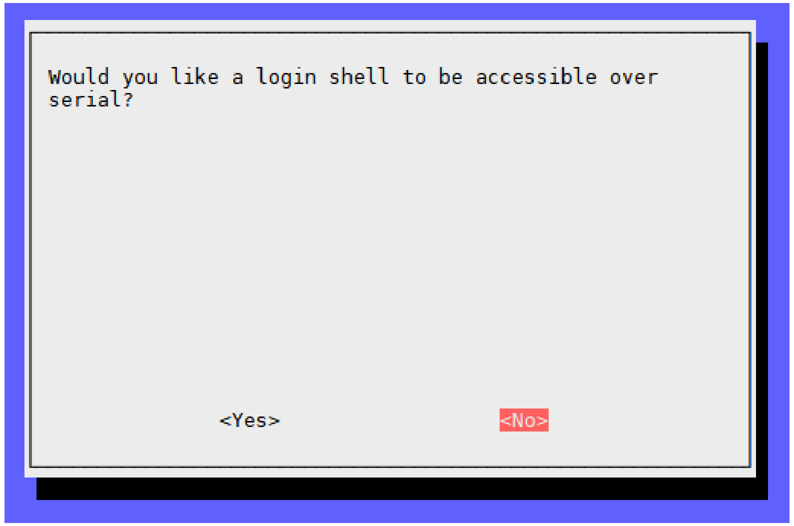
```
5 Interfacing Options  Configure connections to peripherals
```

Another set of options will appear.



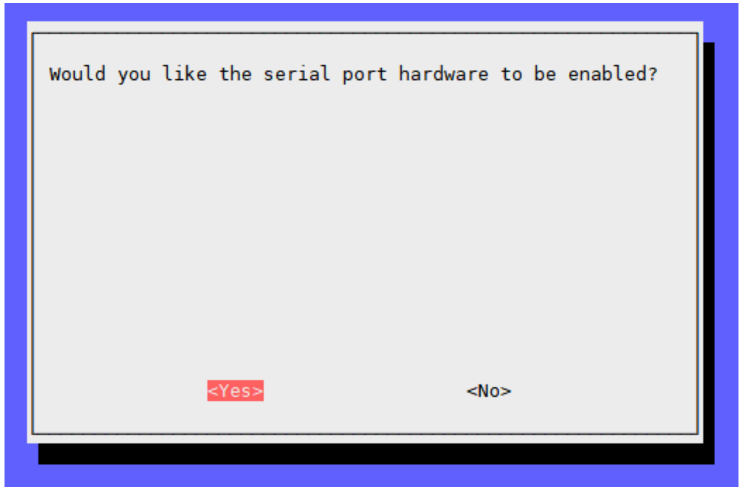
Select **Serial** and press **Enter**

You will be asked if you would like a login shell to be accessible over serial.



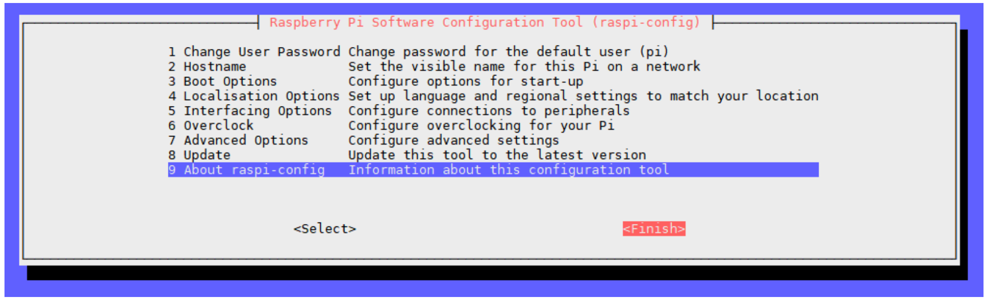
Select **No** and press **Enter**.

Then you will be asked if you would like the serial port hardware to be enabled.

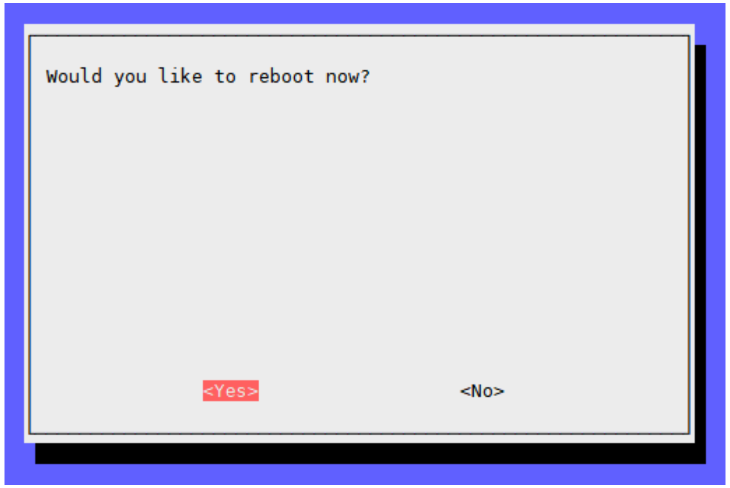


Select **Yes** and press **Enter**.

Press **Enter** again to return to the first set of menus.



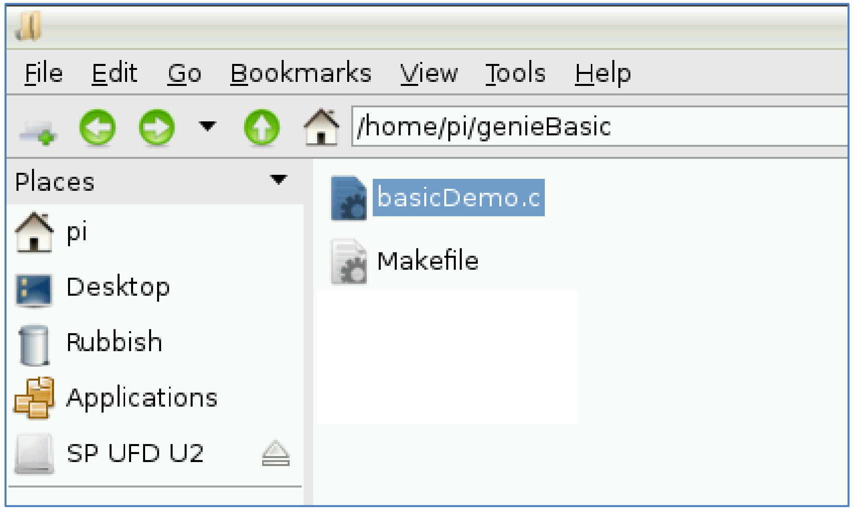
Select **Finish** and press **Enter**.



When prompted to reboot, select **Yes** and press **Enter**.

Download the Demo Files

This document comes with a demo source code written in C and an accompanying make file. Save these files inside a folder in the Raspberry Pi (either transfer them using a USB flash drive or download them from the Internet if the Raspberry Pi has a network connection). In this tutorial, the files are saved in the location shown below.



Understanding the Raspberry Pi Demo Source Code

Open the file **basicDemo.c**. Note that comments have been added to the code to help the user. Additional explanations are now given below. Discussion starts with the main function.

Use a Thread to Drive the Cool Gauge

There are two tasks to be carried out by the program. One of these is that it waits for messages from the slider and then it writes the values contained in these messages back to the LED digits. The other task is that it constantly writes to the cool gauge. The first task is assigned to the main program. The second task is assigned to a thread. One can think of a thread as another program running in parallel with the main program. The virtual effect is that the program is writing to the cool gauge while simultaneously receiving messages from the slider and writing to the LED digits. The first statement inside the main function declares a thread.

```
pthread_t myThread;                    //declare a thread
```

About ten lines below the declaration, the thread is created and started.

```
(void)pthread_create (&myThread,
                        NULL, handleCoolGauge, NULL);
```

The third argument, handleCoolGauge, is the routine that the thread will execute, once it is created. Note that this routine is at the beginning part of the source code.

```
static void *handleCoolGauge(void *data)
{
  int gaugeVal = 0;   //holds the value of the cool gaug
  int step = 1;       //increment or decrement value, in

  for(;;)             //infinite loop
  {
    //write to Coolgauge0
    genieWriteObj(GENIE_OBJ_COOL_GAUGE, 0x00, gaugeVal);
    usleep(10000);    //10-millisecond delay
    gaugeVal += step; //increment or decrement

    if(gaugeVal == 99)  step = -1;
    if(gaugeVal == 0)   step = 1;
  }
```

Observe the correct syntax when creating a thread routine.

Change the Cool Gauge's Status

Of particular importance in thread routine is the function

```
genieWriteObj(GENIE_OBJ_COOL_GAUGE, 0x00, gaugeVal);
```

This function is used to change the status of an object by writing values to it. **`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 **`step`**. 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-RaspPi library. All of the objects can be written to except GENIE_OBJ_KEYBOARD and GENIE_OBJ_STATIC_TEXT.

```
#define GENIE_OBJ_DIPSW          0
#define GENIE_OBJ_KNOB           1
#define GENIE_OBJ_ROCKERSW       2
#define GENIE_OBJ_ROTARYSW       3
#define GENIE_OBJ_SLIDER         4
#define GENIE_OBJ_TRACKBAR       5
#define GENIE_OBJ_WINBUTTON      6
#define GENIE_OBJ_ANGULAR_METER  7
#define GENIE_OBJ_COOL_GAUGE     8
#define GENIE_OBJ_CUSTOM_DIGITS  9
#define GENIE_OBJ_FORM           1(
#define GENIE_OBJ_GAUGE          1:
#define GENIE_OBJ_IMAGE          1:
#define GENIE_OBJ_KEYBOARD       1:
#define GENIE_OBJ_LED            1‹
#define GENIE_OBJ_LED_DIGITS     1!
#define GENIE_OBJ_METER          1(
#define GENIE_OBJ_STRINGS        1'
```

```
GENIE_OBJ_THERMOMETER    18
GENIE_OBJ_USER_LED       19
GENIE_OBJ_VIDEO          20
GENIE_OBJ_STATIC_TEXT    21
GENIE_OBJ_SOUND          22
GENIE_OBJ_TIMER          23
GENIE_OBJ_SPECTRUM       24
GENIE_OBJ_SCOPE          25
GENIE_OBJ_TANK           26
GENIE_OBJ_USERIMAGES     27
GENIE_OBJ_PINOUTPUT      28
GENIE_OBJ_PININPUT       29
GENIE_OBJ_4DBUTTON       30
GENIE_OBJ_ANIBUTTON      31
GENIE_OBJ_COLORPICKER    32
GENIE_OBJ_USERBUTTON     33
```

Declare a genieReplyStruct Type Structure

Following the thread declaration is the statement

```
struct genieReplyStruct reply ;  //declare a genieRe
```

A structure, **reply**, of the **genieReplyStruct** type is now created. It contains four variables, each of which will hold a corresponding byte to be received from the display module. To illustrate:

```
struct genieReplyStruct
{
  int cmd ;
  int object ;
  int index ;
  unsigned int data ;
} ;
```

The last variable, **data**, will hold the MSB and LSB data bytes of a message from the display.

Print Text on the Terminal

The following lines will print some informative text on the Raspberry Pi's terminal.

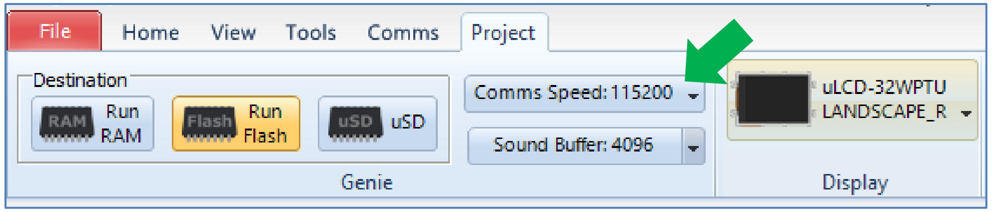
```
printf("\n\n");
printf("Visi-Genie-Raspberry-Pi basic demo\n");
printf("==================================\n");
printf("Program is running. Press Ctrl + C to close.
```

Open the Serial Port and Set the Baud Rate

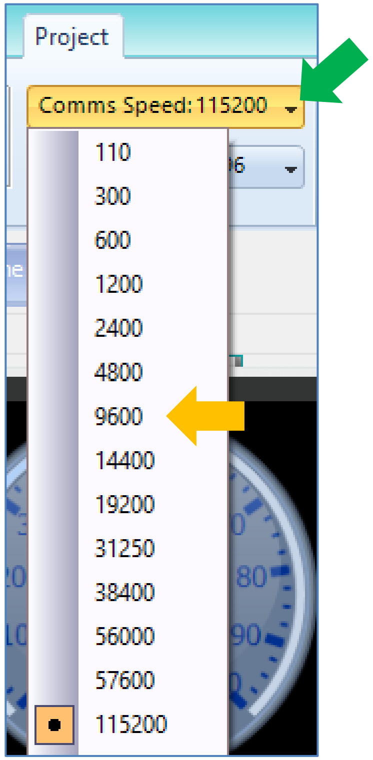
The command

```
genieSetup("/dev/serial0", 115200);
```

opens the Raspberry Pi's on-board serial port for communication with the 4D display module at 115200 bps. Logically, the display should also communicate with the Raspberry Pi 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 Raspberry Pi as well.

```
genieSetup("/dev/serial0", 9600);
```

N.B.: Again, remember that the baud rate of the display module should match that of the Raspberry Pi.

Send a Text String

To send a string to the display module, use the function

```
genieWriteStr(0x00, "Hello world!"); //write to Strin
```

Note that two arguments are required – the first being the index of the strings object to be written to and the second being the text to be displayed.

Receiving Data from the Display

The following block is the core part of the main program.

```
for(;;)                          //infinite loop
{
  while(genieReplyAvail())       //check if a mes
  {
    genieGetReply(&reply);       //take out a mes
    handleGenieEvent(&reply);    //call the event
  }
  usleep(10000);                 //10-milliseconc
}                                //CPU in case ar
```

There are three tasks being performed in this block. First, it is checked if there are pending messages from the display.

```
  while(genieReplyAvail())       //check if a mes
```

A message represents a reply or an event from any of the objects in the display module. Messages are actually queued in the background, starting from the moment that the serial port is successfully opened. If there are pending messages, one is taken out from the queue and copied to the previously created **genieReplyStruct** structure, **reply**.

```
    genieGetReply(&reply);       //take out a mes
```

After having copied the contents of the actual message to **reply**, a user-defined function is now called to process the **reply**. This user-defined function is the event handler.

```
    handleGenieEvent(&reply);    //call the event
```

The User's Event Handler

This function is responsible for evaluating the message received from the display.

```
void handleGenieEvent(struct genieReplyStruct * reply)
{
  if(reply->cmd == GENIE_REPORT_EVENT)     //check if the cmd byte
  {
    if(reply->object == GENIE_OBJ_SLIDER) //check if the object b
    {
      if(reply->index == 0)               //check if the index by
        //write to the LED digits object
        genieWriteObj(GENIE_OBJ_LED_DIGITS, 0x00, reply->data);
    }
  }

  //if the received message is not a report event, print a messag
  else
    printf("Unhandled event: command: %2d, object: %2d, index: %d
}
```

The message is broken down into its components using nested IF statements. First, the **cmd** or **command** byte is checked if it is a **GENIE_REPORT_EVENT**.

```
  if(reply->cmd == GENIE_REPORT_EVENT)     //che
```

Objects in ViSi-Genie can be set to report a message when their status has changed. This message will have the command byte GENIE_REPORT_EVENT. Here is a list of Genie commands taken from the **.h** file of the ViSi-Genie-RaspPi 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
```

Then the object byte is tested if it is that of a slider.

```
if(reply->object == GENIE_OBJ_SLIDER) //check i
```

Lastly, the index byte is checked if it is equal to 0, which means that the message is from Slider0.

```
 if(reply->index == 0)              //check if th
```

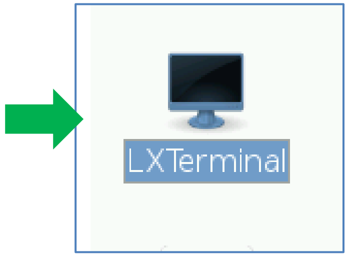
After having performed this series of confirmations, the program now writes the value contained by the data bytes of the message to Leddigits0.

```
genieWriteObj(GENIE_OBJ_LED_DIGITS, 0x00, reply->data);
```

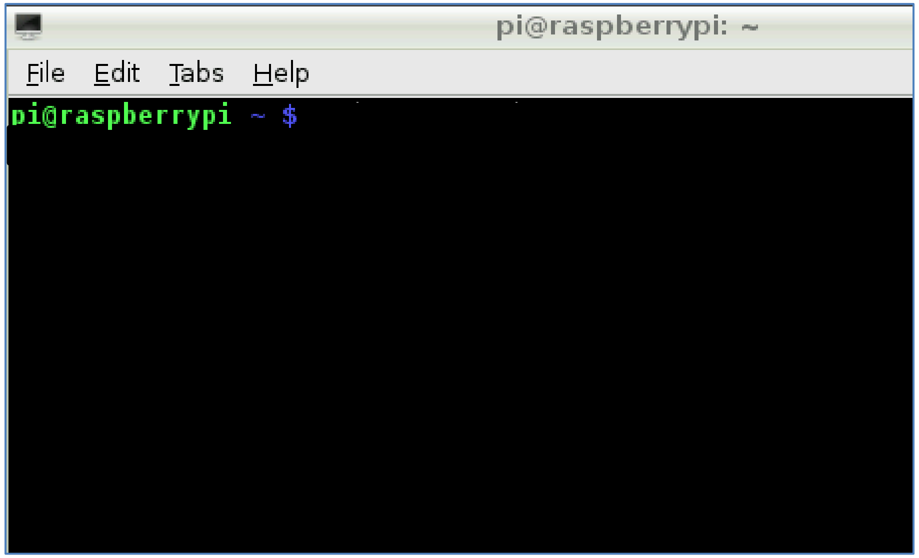
In conclusion the program evaluates each byte of a message (command, object, object index, and value), and then makes a decision according to the result of this evaluation.

Compile the Source Code

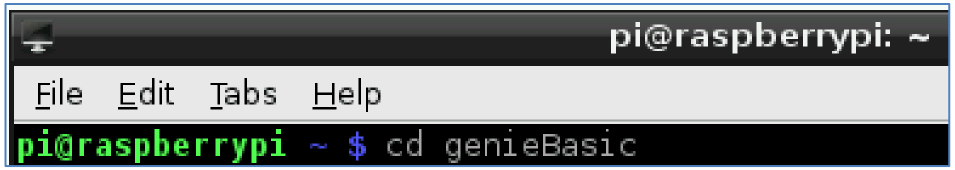
Compile the demo source code first before running it. On the desktop, double-click on the LXTerminal icon.



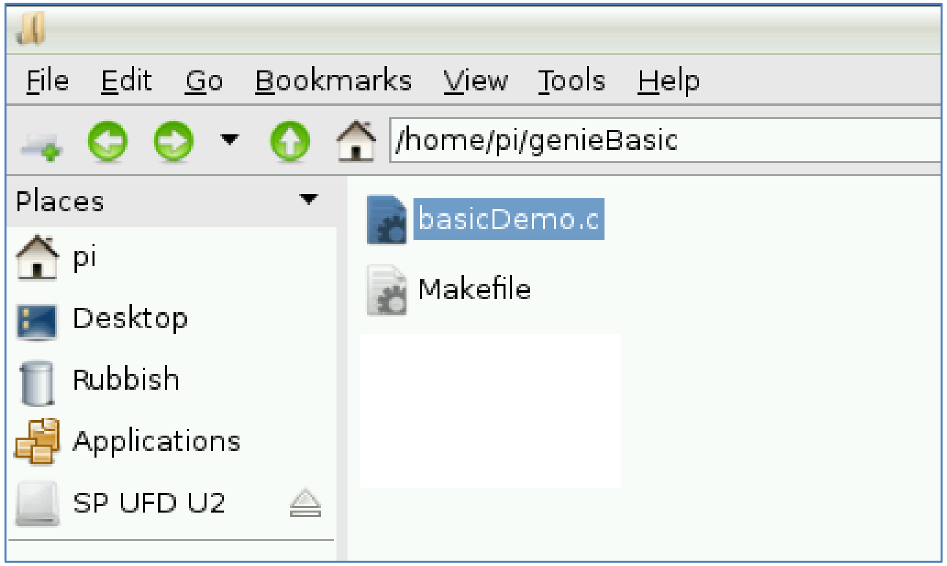
The terminal window appears.



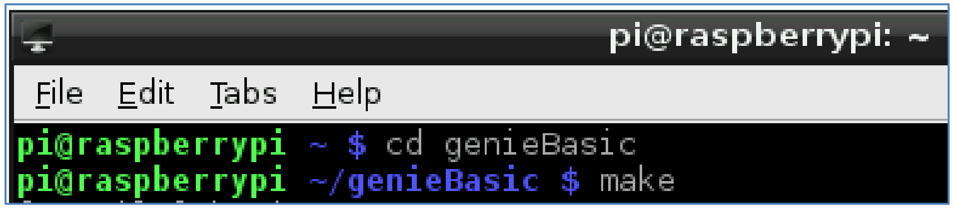
Navigate to the folder where the demo files are located by typing the command as shown below.



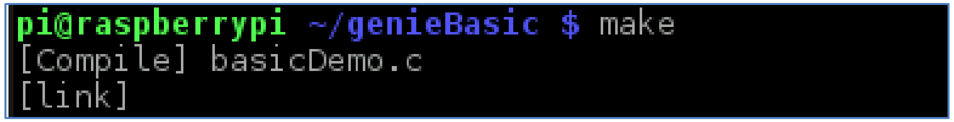
In this tutorial, the demo files are located here.



To compile the source code type **make**.

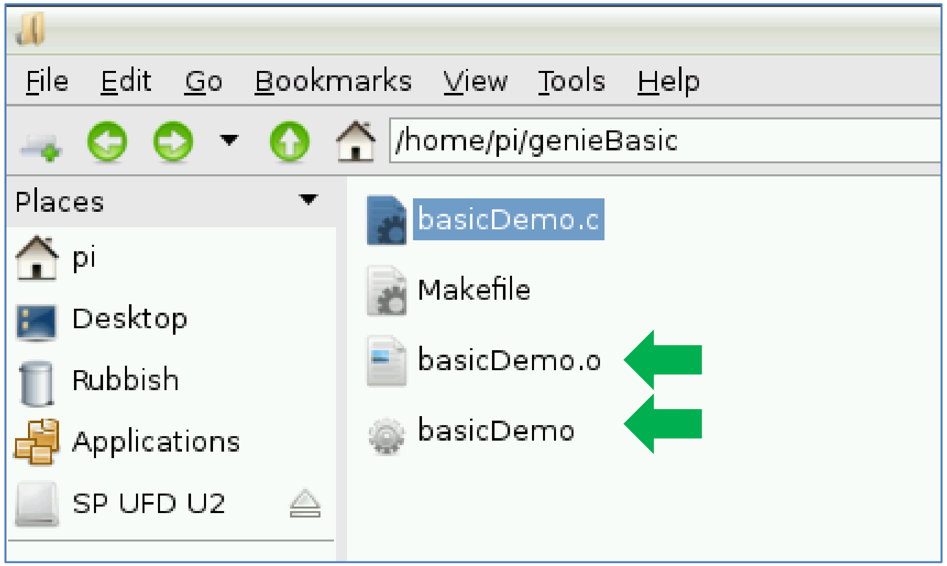


The source code now compiles.



Run the Program

Notice that two additional files are created in the folder. One of these is an executable file.



Before running the program, connect the display module to the Raspberry Pi first. See the following section "Connect the 4D Display Module to the Raspberry Pi" then return here. To run the program, type the command shown below.

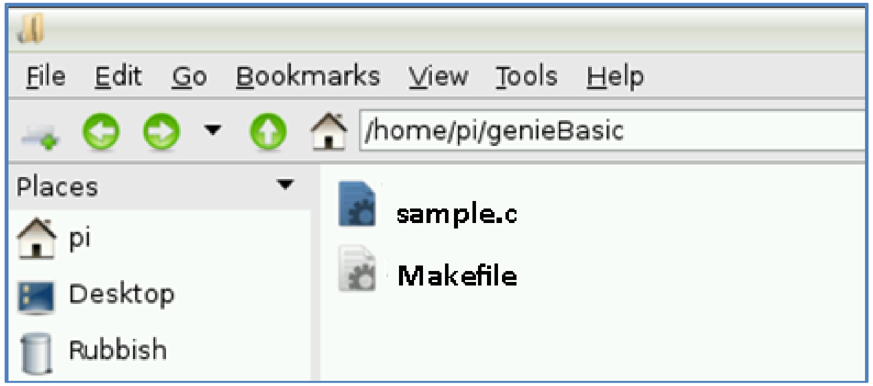
The program now runs.

```
pi@raspberrypi ~/genieBasic $ sudo ./basicDemo

Visi-Genie-Raspberry-Pi basic demo
==================================
Program is running. Press Ctrl + C to close.
```

Editing the Make File

The make file is used to compile the source code. After having written a new C source code, the user has the option of using the make file that comes with this application note. The make file needs to be edited though. This section shows how this is done.

First, save the new source code and a copy of the make file inside a folder. In this example, the source code, named **sample.c**, and the make file are saved inside the folder **/genieBasic**.



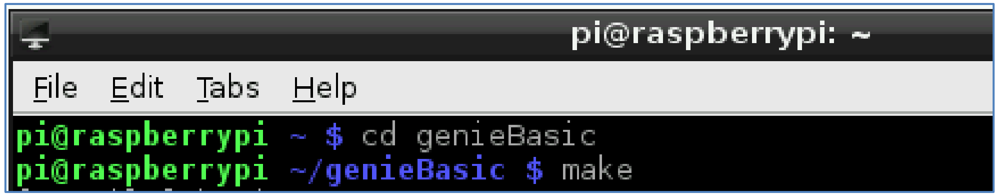
Next, open the make file and replace the text highlighted below with the filename of the new source code.

```
# Makefile:
#
#     Make Visi-Genie Demos on the Raspberry Pi
#
#     4D Systems August 2013
#     Based on makefile by Gordon Henderson
##################################################

#DEBUG    = -g -O0
DEBUG= -O2
CC   = gcc
INCLUDE   = -I/usr/local/include
CFLAGS    = $(DEBUG) -Wall $(INCLUDE) -Winline -pipe

LDFLAGS   = -L/usr/local/lib
LIBS    = -lm -lpthread -lwiringPi -lgeniePi

SRC   = basicDemo.c

# May not need to  alter anything below this line
##################################################

OBJ   =    $(SRC:.c=.o)
BINS  =    $(SRC:.c=)

basicDemo: basicDemo.o
      @echo [link]
      @$(CC) -o $@ basicDemo.o $(LDFLAGS) $(LIBS)
```

Save the file.

```
# Makefile:
#
#     Make Visi-Genie Demos on the Raspberry Pi
#
#     4D Systems August 2013
#     Based on makefile by Gordon Henderson
#####################################################

#DEBUG     = -g -O0
DEBUG= -O2
CC    = gcc
INCLUDE    = -I/usr/local/include
CFLAGS     = $(DEBUG) -Wall $(INCLUDE) -Winline -pipe

LDFLAGS    = -L/usr/local/lib
LIBS    = -lm -lpthread -lwiringPi -lgeniePi

SRC    = sample.c

# May not need to  alter anything below this line
#####################################################

OBJ  =     $(SRC:.c=.o)
BINS =     $(SRC:.c=)

sample: sample.o
     @echo [link]
     @$(CC) -o $@ sample.o $(LDFLAGS) $(LIBS)
```

To compile the source code type **make**.

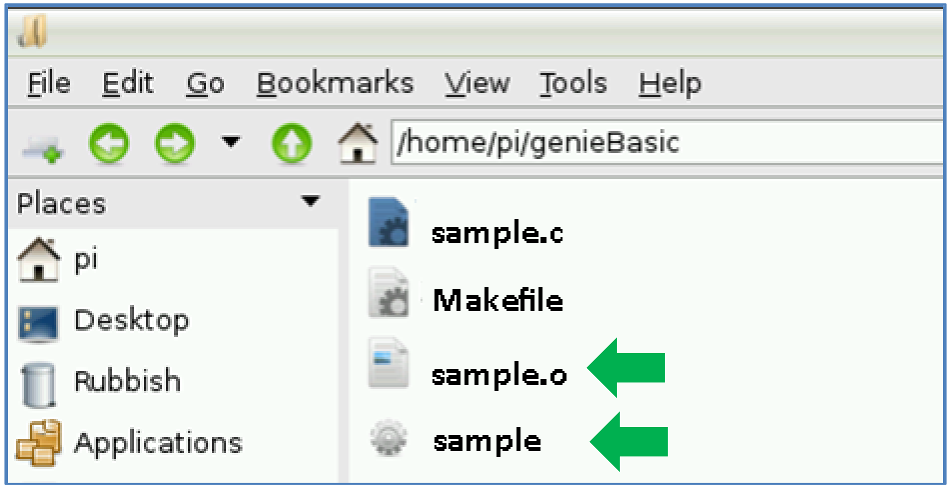


If there are no errors, the source code will now compile.

```
^Cpi@raspberrypi ~/genieBasic $ make
[Compile] sample.c
[link]
```

Run the Program

Notice that two additional files are created in the folder. One of these is the executable file.



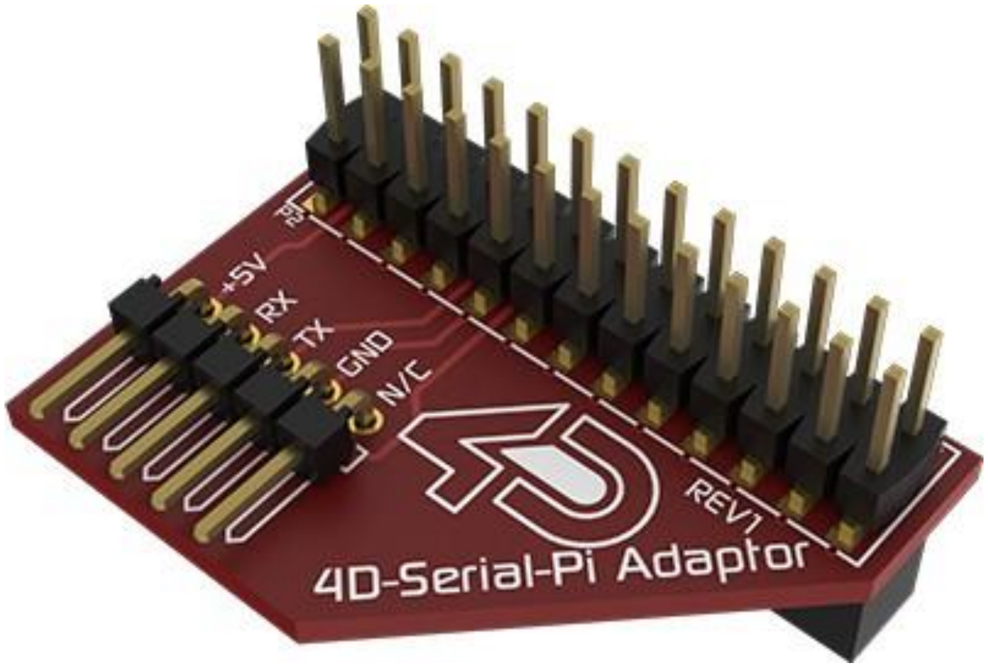
To run the program, type the command shown below.

```
pi@raspberrypi ~/genieBasic $ sudo ./sample
```

Connect the 4D Display Module to the Raspberry Pi

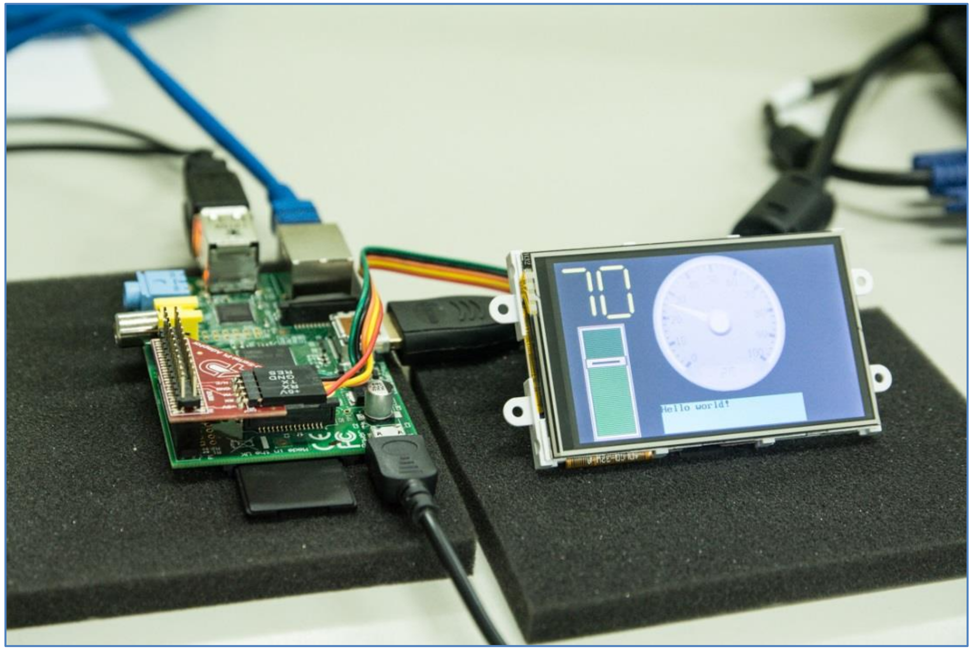
This section discusses how to connect the display module to the Raspberry Pi. The user has the option of using a 4D Serial Pi Adaptor or simply connecting the Tx0, Rx0, 5V, and GND pins of the display module to the corresponding pins of the Raspberry Pi.

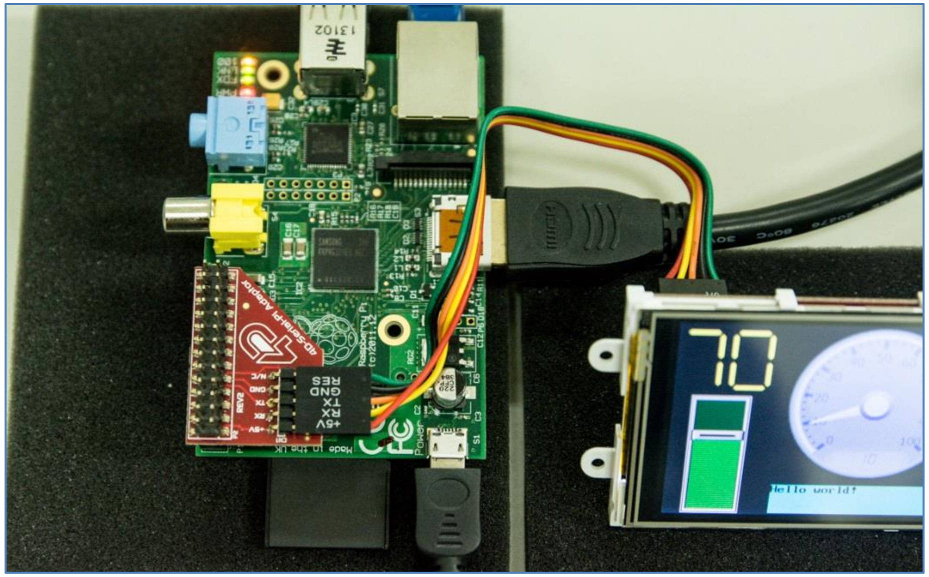
Using the 4D Serial Pi Adaptor



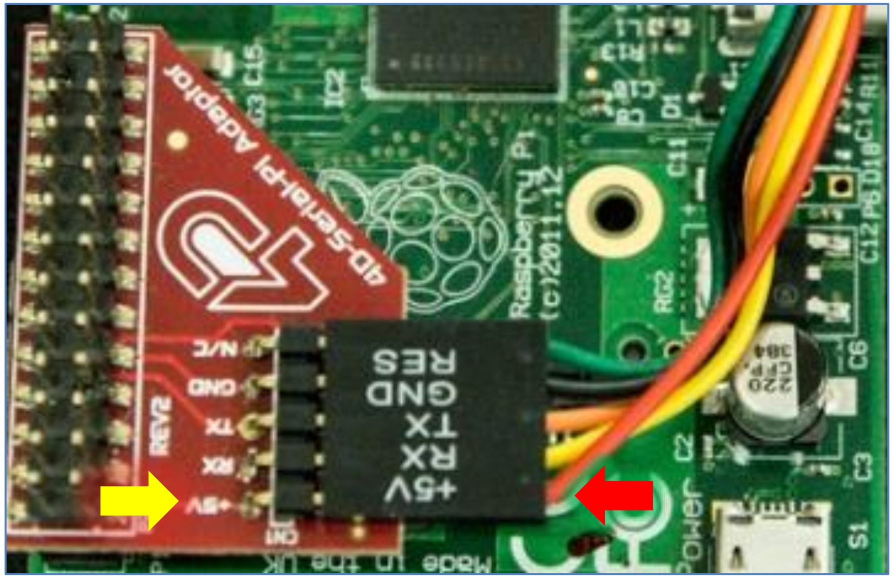
When using the 4D Serial Pi Adaptor, the display module is powered from the Raspberry Pi's 5V bus. The power supply therefore must be able to provide enough current to both the Raspberry Pi and the display module.

The complete setup:

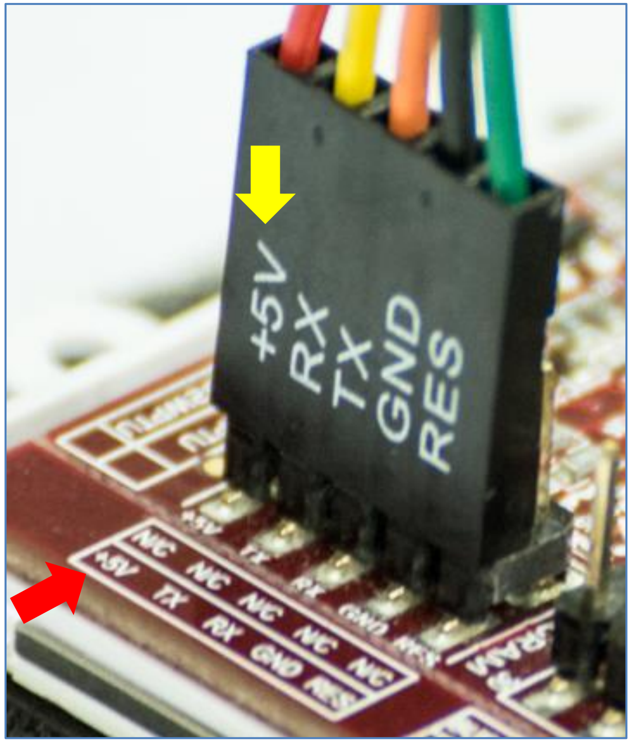




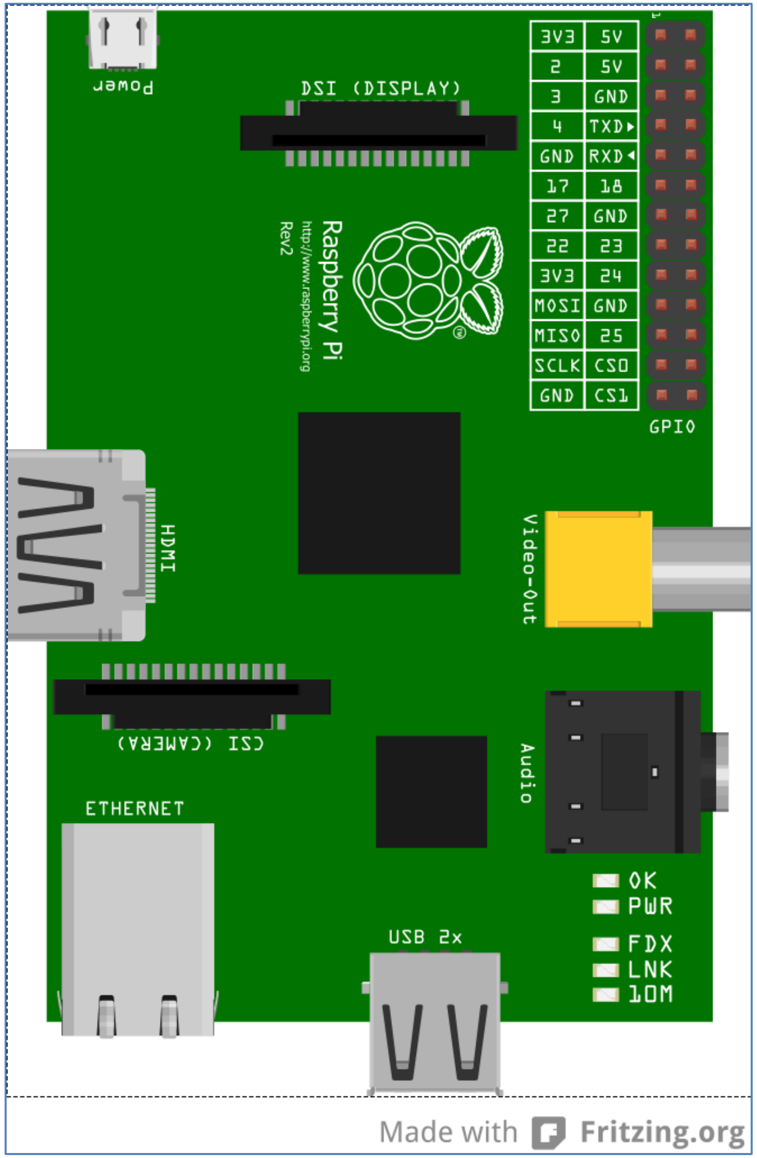
Check the orientation of the 5-way-cable-to-adaptor connection.



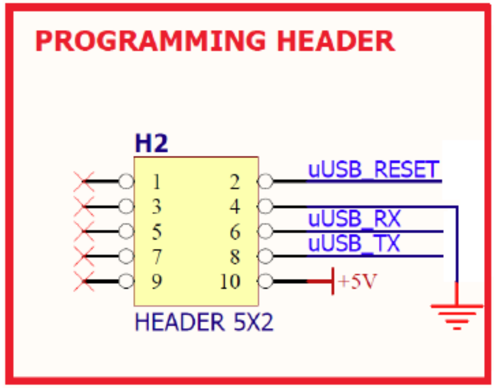
Check the orientation of the 5-way-cable-to-display-module connection.



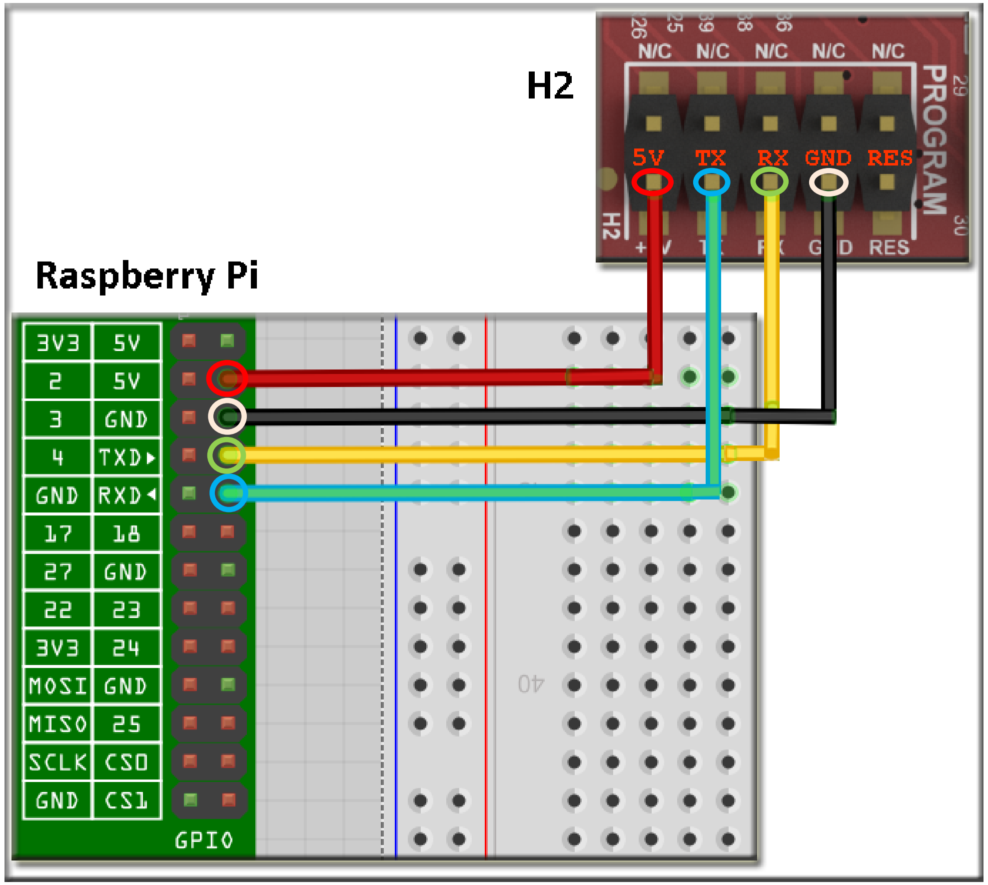
The Raspberry Pi



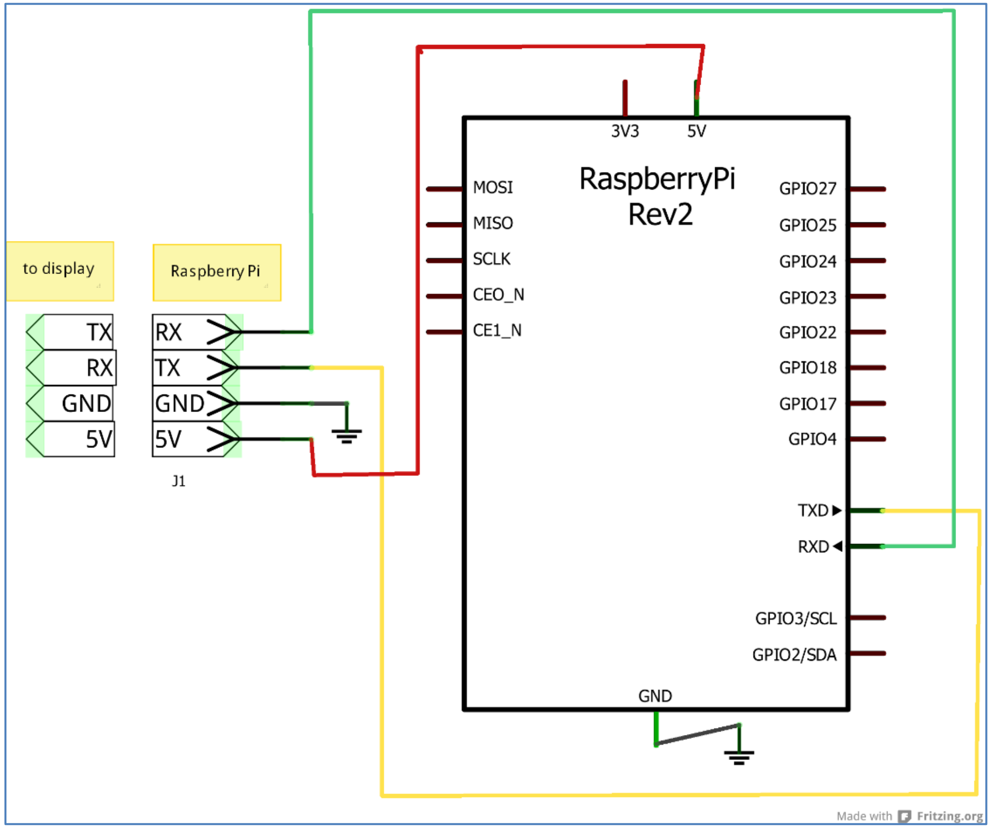
The 4D Display Module



Connection Using Jumper Wires



Schematic diagram:



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.