

Raspberry Pi Example: Serial Data Logger with Display

Overview

This example project describes how to create a serial data logger using a Raspberry Pi. The example provides a Python script that is automatically started at power-on to receive serial data from an external source using the 'serial0' UART input, and then timestamps and logs the data to an HDMI display and a connected USB drive. This example uses an RPi mounted to the back of a SunFounder 10" LCD display, however any HDMI display can be used. This example uses an MSP430 development board to stream serial data, however alternate serial data sources can also be used.

Hardware Used

This example used a [Raspberry Pi 3 Model B](#) mounted to the back of a [SunFounder 10" LCD display](#), as shown below. The example setup also included a 5V 2.5A power supply, and a standard wireless keyboard and mouse combo. In order to log data a USB drive is also needed. I used a [SanDisk Cruzer Fit 8GB](#) drive. Any USB drive should work, but the small form factor of this drive hides nicely behind the display and provides a clean looking setup.



SunFounder 10" LCD Display with
Raspberry Pi 3 Model B



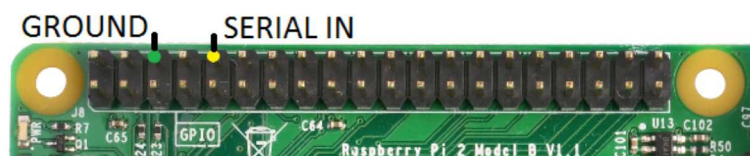
SanDisk Cruzer Fit USB Drive
or similar

IMPORTANT: Make sure the Raspberry Pi has been made ["Project Ready"](#) before proceeding.

IMPORTANT: Make sure the USB Drive has been formatted as FAT (e.g. FAT32), and that the drive is given the label "USBDRIVE", and that a textfile named "validate.txt" is created at the top level of the drive. The reason why this is necessary will be explained later.

Required Connections

The example project uses serial data coming into the UART RX (Serial Input) pin 10. When the Raspberry Pi is mounted on the back of the LCD display, this is the 5th pin from the left on the top row. It is marked by a yellow dot on the image below. This pin must be connected to the serial output of the device that will provide the data to be logged. A ground reference is also required. That connection was made using the pin 6 ground. This is the 3rd pin from the left on the top row, and is marked with a green dot in the image below.



In order to provide a stream of serial data for the example, the [MSP430 4-bit counting example](#) was modified to include the MSP430F5529LP_UART library, and to assemble and transmit a serial message in the main loop each time the counter was incremented. The code added to the example looked like this:

```
sprintf(string, "%3d, %3X, %d%d%d%d%d%d%d \r\n", s_count_u8, s_count_u8, BYTETO_BINARY(s_count_u8));
```

```
SendSerialMsg(string, strlen(string));
```

The format of the data is not important, although comma-separated data makes it easy to import into applications such as Excel. The important part is that the string is terminated by a '\n' new line character. This informs the Python script running on the Raspberry Pi that the data stream is complete which triggers the logging of the bytes received. The '\r' carriage return character is ignored by the script and is not captured or logged. It is included to make the data stream compatible with both Windows and Linux.

Example Code

The [datalogger.py](#) Python script looks like this:

```
#!/usr/bin/env python
"""
/* ##### */
/*
 * This file was created by www.DavesMotleyProjects.com
 *
 * This software is provided under the following conditions:
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * 'Software'), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
/* ##### */
"""

import io
import os
import time
import datetime
import serial

from time import sleep      # to create delays
from locale import format  # to format numbers
from shutil import rmtree   # to remove directories

PathValid = False
FileName = ""
FilePath = ""
```

Installing and Testing the Script

Before setting up the automation, try installing and running the script manually. For this initial test, it is recommended that the USB drive be removed, and the serial data stream be disconnected before starting. If everything works, this should provide a nice 'static' display to observe, and if needed troubleshoot. To download and test the datalogger, open a terminal window, and perform the following actions.

Create a folder called datalogger in the pi home directory, and then enter the new directory.

```
cd ~
mkdir datalogger
cd datalogger
```

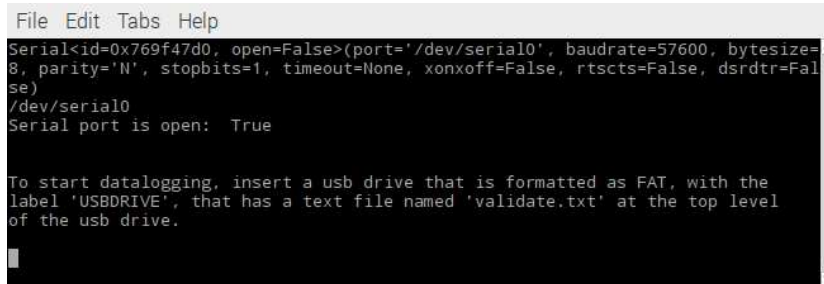
Download the [datalogger.py](#) Python script file, by entering the following in the terminal window. The wget command will download the specified file, and place it in your current directory, which is the datalogger directory.

```
wget http://www.davesmotleyprojects.com/raspi/raspi-data-logger/datalogger.py
```

Now launch the script manually. Note: sudo is required in order to allow Python to access the Serial UART, which is a restricted resource.

```
sudo python datalogger.py
```

If everything worked, you should see the following display. The serial port being used is /dev/serial0, which is a high-performance hardware UART, and when queried the port should respond as **'Serial port is open: True'**. If the USB drive was not installed when the datalogger started, a user prompt to install a correctly configured USB drive will also be displayed.



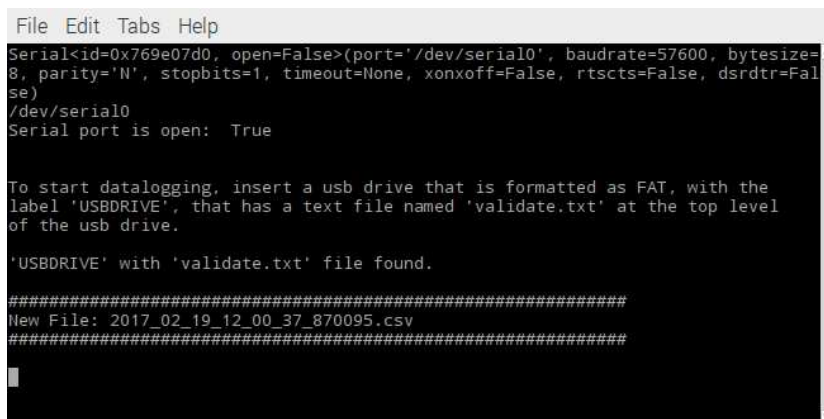
```
File Edit Tabs Help
Serial<id=0x769f47d0, open=False>(port='/dev/serial0', baudrate=57600, bytesize=
8, parity='N', stopbits=1, timeout=None, xonxoff=False, rtscts=False, dsrdtr=Fa
se)
/dev/serial0
Serial port is open: True

To start datalogging, insert a usb drive that is formatted as FAT, with the
label 'USBDRIVE', that has a text file named 'validate.txt' at the top level
of the usb drive.
```

If the Serial port isn't opening, verify that the RPi was made ["Project Ready"](#). If the Serial port is working but there appears to be something wrong with the script, you may want to consider opening and running the script inside of an editor like geany. You can do that by entering the following. Note: sudo is still required to access the serial port. The '&' at the end simply releases terminal window after executing the command. If you find an issue with the script, please provide [feedback](#), and I will correct it.

```
sudo geany &
```

If everything worked above, insert a properly configured USB drive (but don't connect the serial data source yet). The script should detect the existence of the drive, and will indicate that it was found, and a new file will be created and the file name will be displayed, as shown below.



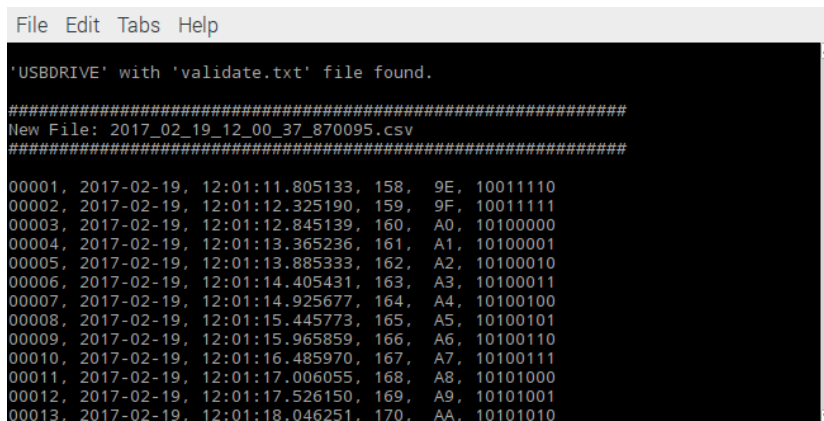
```
File Edit Tabs Help
Serial<id=0x769e07d0, open=False>(port='/dev/serial0', baudrate=57600, bytesize=
8, parity='N', stopbits=1, timeout=None, xonxoff=False, rtscts=False, dsrdtr=Fa
se)
/dev/serial0
Serial port is open: True

To start datalogging, insert a usb drive that is formatted as FAT, with the
label 'USBDRIVE', that has a text file named 'validate.txt' at the top level
of the usb drive.

'USBDRIVE' with 'validate.txt' file found.

#####
New File: 2017_02_19_12_00_37_870095.csv
#####
```

Now connect the serial data source. Data logging should start, as shown in the window below.



```
File Edit Tabs Help

'USBDRIVE' with 'validate.txt' file found.

#####
New File: 2017_02_19_12_00_37_870095.csv
#####

00001, 2017-02-19, 12:01:11.805133, 158, 9E, 10011110
00002, 2017-02-19, 12:01:12.325190, 159, 9F, 10011111
00003, 2017-02-19, 12:01:12.845139, 160, A0, 10100000
00004, 2017-02-19, 12:01:13.365236, 161, A1, 10100001
00005, 2017-02-19, 12:01:13.885333, 162, A2, 10100010
00006, 2017-02-19, 12:01:14.405431, 163, A3, 10100011
00007, 2017-02-19, 12:01:14.925677, 164, A4, 10100100
00008, 2017-02-19, 12:01:15.445773, 165, A5, 10100101
00009, 2017-02-19, 12:01:15.965859, 166, A6, 10100110
00010, 2017-02-19, 12:01:16.485970, 167, A7, 10100111
00011, 2017-02-19, 12:01:17.006055, 168, A8, 10101000
00012, 2017-02-19, 12:01:17.526150, 169, A9, 10101001
00013, 2017-02-19, 12:01:18.046251, 170, AA, 10101010
```

When you open the corresponding .csv file on the USB drive, you will see the following data. (Only the first several lines are shown). The first row is added to the file each time a new file is created. This action is performed in the function `start_new_file()`. Currently everything beyond "Index, Date, Time" is listed as just "Data" because the script is currently generic. It doesn't know, or care, what data is passed to it

or what format the data is in. If the script is customized for a specific purpose, you may want to consider changing the first row to include the actual data headers.

```
Index, Date, Time, Data
00001, 2017-02-19, 12:01:11.805133, 158, 9E, 10011110
00002, 2017-02-19, 12:01:12.325190, 159, 9F, 10011111
00003, 2017-02-19, 12:01:12.845139, 160, A0, 10100000
00004, 2017-02-19, 12:01:13.365236, 161, A1, 10100001
00005, 2017-02-19, 12:01:13.885333, 162, A2, 10100010
00006, 2017-02-19, 12:01:14.405431, 163, A3, 10100011
00007, 2017-02-19, 12:01:14.925677, 164, A4, 10100100
```

Automatically Start Data Logging

Once everything is working manually, modify the startup.sh script to automatically launch the data logging script at power-on.

In the terminal window, enter the following:

```
cd ~
sudo leafpad startup.sh
```

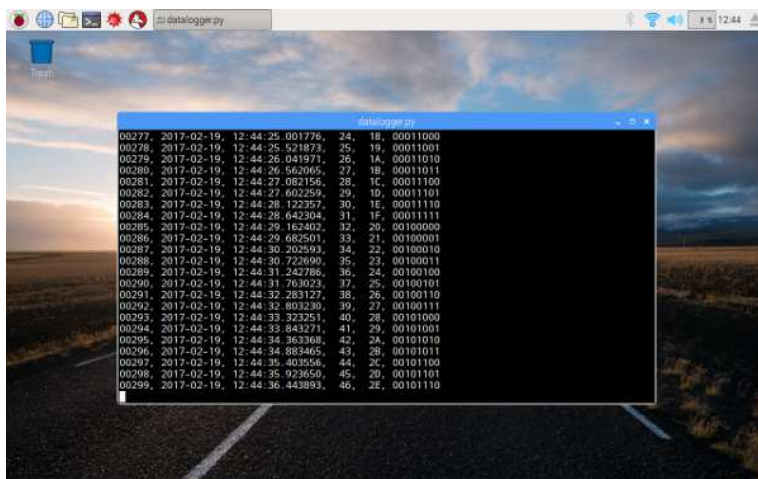
This will return to the /home/pi directory, and open the startup.sh script. In that file, enter the following items, save the file, and exit.

```
#!/bin/bash
sleep 5
echo "Starting data logger"
xterm -fg white -bg black -fa monaco -fs 12 -geometry 90x24 -e "sudo python ./datalogger
./datalogger.py"
```

If they aren't already, insert the USB drive and connect the serial data source, then reboot the RPi by entering the following in the terminal window.

```
sudo reboot
```

If everything works properly, the data logging will start automatically when the RPi restarts.



Why USBDRIVE and validate.txt File?

One of the things that may not be obvious, is *why* the requirement to label the USB drive as 'USBDRIVE', and why must it contain a file called 'validate.txt'?

The answer to the first question is that this allows us to know how the USB drive will be mounted. The file path will be '/media /pi/USBDRIVE'.

Answering the second question is a little more complicated. This has to do with the operating system response to executing the Python `open(filepath, "a")` command when the USB filepath doesn't exist. This can happen when the operating system un-mounts the drive *just*

before the execution of the open command. In response, Raspbian will create a 'fake' USBDRIVE directory under /media/pi. When this occurs the Python script would normally continue data logging because the directory *appears* to exist, however the 'fake' USBDRIVE directory is not accessible. When the real USB drive is re-mounted, Raspbian will mount it as USBDRIVE1, and the data logging will be permanently broken.

To resolve this, the 'validate.txt' file requirement was added. The reason is that the 'validate.txt' file will *not* exist in the 'fake' USBDRIVE directory, and the condition can be detected and resolved by deleting the 'fake' path. This allows the real USB drive to be re-mounted correctly as USBDRIVE and data logging continues properly when re-inserted. The screenshot below shows the response when the 'fake' path is created by continuing to remove and insert the USB drive.

```
File Edit Tabs Help
00031, 2017-02-19, 12:54:17.910986, 140, 8C, 10001100
00032, 2017-02-19, 12:54:18.431089, 141, 8D, 10001101
00033, 2017-02-19, 12:54:18.951187, 142, 8E, 10001110
00034, 2017-02-19, 12:54:19.471427, 143, 8F, 10001111
00035, 2017-02-19, 12:54:19.991521, 144, 90, 10010000
00036, 2017-02-19, 12:54:20.511610, 145, 91, 10010001

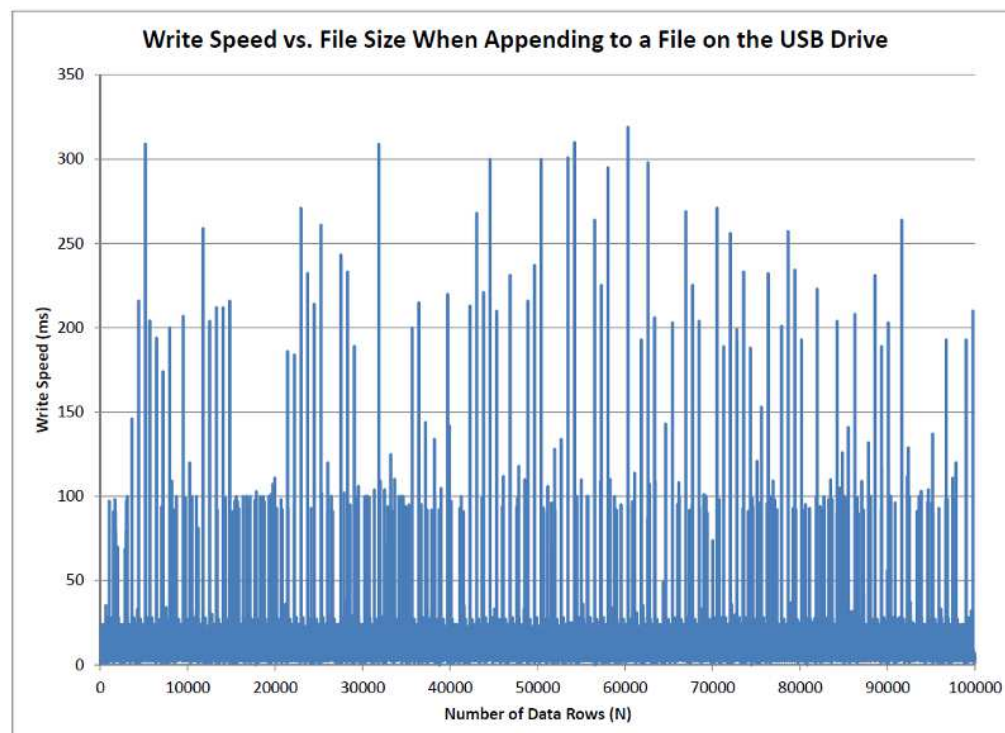
#####
New File: 2017_02_19_12_54_21_032140.csv
#####

00001, 2017-02-19, 12:54:21.551807, 147, 93, 10010011
path corruption suspected
path corruption confirmed
validate.txt file was not found
USB path is not valid, corrupted, missing, or ejected
path corruption corrected
```

Data Logging File Write Speed

In order to keep the file size manageable, a new file is created each time the number of rows in the file reaches 65535. The file size is controlled by the value of MAX_ROWS_IN_FILE. An interesting question is, does the write speed slow down when appending to files that are very large? In order to test this, a small Python script was used to write 100000 lines to a text file on the USB drive, tracking the time before and immediately after creating the time stamp and data string, printing to the display, and opening, appending, and closing the USB text file. A plot of the resulting write speed is shown below. Although, it can be seen that occasionally the operating system will execute other tasks, interrupting the Python script, and extending the time, there is no apparent trend to suggest that write speed is dependent on file size.

The average write speed is about 4.5 ms.



Copyright © 2017 by David Hunt. All rights reserved.