Microstack Node Documentation

Release 0.4.0

Thomas Preston

Mar 06, 2017

Contents

Microstack Node (for Python) is the software component of Microstack which communicates with the Microstack I/O boards.

Contents:

CHAPTER 1

Installation

First, make sure you have enabled I2C and SPI by running:

```
sudo raspi-config
```

and then navigating to:

```
Advanced Options > Would you like the I2C interface to be enabled? > Yes
Would you like the I2C kernel module to be loaded by default? > Yes
```

and:

```
Advanced Options > Would you like the SPI interface to be enabled? > Yes
Would you like the SPI kernel module to be loaded by default? > Yes
```

Then reboot.

You can install *microstacknode* with either *apt-get* or *pip*.

Installing with *apt-get*

Make sure you are using the lastest version of Raspbian:

```
$ sudo apt-get update && sudo apt-get upgrade
```

Install `microstacknode` for Python 3 with the following command:

```
$ sudo apt-get install python3-microstacknode
```

Installing with *pip*

Warning: Do not install *microstacknode* with both *apt-get* and *pip* as unexpected things will happen. Consider using virtual environments.

Make sure *pip* is installed:

```
sudo apt-get install python3-pip
```

Install microstacknode using pip:

```
sudo pip3 install microstacknode
```

GPS

The GPS uses the serial port. By default it is configured to output the login shell. You must disable this before GPS will work. To do so, run:

```
sudo raspi-config
```

Navigate to `Advanced Options` > `Serial`, disable the login shell and then reboot.

Note: If you're using a Raspberry Pi 3 you will also need to fix the CPU *core_freq* at 250 otherwise the serial port baud rate will not stay constant. To do this add `core_freq=250` to `/boot/config.txt`.

Testing

Accelerometer

Dump accelerometer data with:

```
$ python3 /usr/share/doc/python3-microstacknode/examples/accelcat.py
```

GPS

Dump GPS data:

```
$ python3 /usr/share/doc/python3-microstacknode/examples/gpscat.py
```

Other GPS Software

You might also want to install standard GPS software:

```
$ sudo apt-get install gpsd gpsd-clients python-gps
```

You can dump GPS data with:

```
$ sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock
```

or:

```
$ cgps -s
```

Replace `/dev/ttyAMA0` with `/dev/ttyS0` if you're using a Raspberry Pi 3.

Automatically Starting GPS

Reconfigure the GPS daemon and choose <yes> when asked if you want to start *gpsd* automatically (use the defaults for the remaining options):

```
$ sudo dpkg-reconfigure gpsd
```

CHAPTER 2

Examples

L80 GPS

Note: It usually takes about two minutes for the GPS module to get a GPS fix (indicated by a red flashing LED on the GPS module). Until then, the L80GPS object may throw exceptions.

Note: Some commands may take one or more seconds to return. This is becasue L80GPS reading the latest data from the serial port which the GPS module is connected to. It only updates every second. It is normal for `locus_query_data` to take a long time to return.

Todo

Link to GPS command wiki describing what each of the GP*** do.

Basic GPS (GPGLL is useful for location/time):

```
>>> import microstacknode.hardware.gps.l80gps
>>> gps = microstacknode.hardware.gps.l80gps.L80GPS()  # creates a GPS object
>>> gps.get_gprmc()  # get latest GPRMC data as a dictionary
>>> gps.get_gpvtg()
>>> gps.get_gpgga()
>>> gps.get_gpgsa()
>>> gps.get_gpgsv()
>>> gps.get_gpgll()
>>> gps.get_gptxt()
```

Extra modes:

```
>>> gps.standby()
>>> gps.always_locate()
>>> gps.sleep()
>>> gps.set_periodic_normal()  # wakes from sleep or always locate off
```

LOCUS Data Logging

The GPS unit can log data to its internal memory without anything driving it using the LOCUS logging system. The following commands interface with LOCUS:

```
>>> gps.locus_query()        # Query the status of the LOCUS logger
>>> gps.locus_start()        # Start the logger
>>> gps.locus_stop()         # Stop the logger
>>> gps.locus_erase()        # Erase all data items in the log
>>> gps.locus_query_data()   # Return a list of data items in the logger
```

For example, you could issue the following command:

```
>>> gps.locus_start()
```

Shutdown/halt the device running Python, go for a walk, reboot, start Python and then run the following command and receive GPS data for the walk:

```
>>> gps.locus_query_data()
```

The logger will continue to log GPS data until it runs out of memory, at which point it will stop. Manually stop the LOCUS logger with:

```
>>> gps.locus_stop()
```

MMA8452 Accelerometer

```
>>> import microstacknode.hardware.accelerometer.mma8452q
>>> accelerometer = microstacknode.hardware.accelerometer.mma8452q.MMA8452Q()
>>> accelerometer.open()

>>> accelerometer.get_xyz()
{'x': 0.00927734375, 'y': 0.00341796875, 'z': 0.49853515625}

>>> accelerometer.get_xyz(res12=False)  # turn off 12-bit resolution
{'x': -0.0078125, 'y', -0.0078125, 'z': 0.5078125}
```

You don't have to call *open()* if you're using the *with* statement:

```
import time
from microstacknode.hardware.accelerometer.mma8452q import MMA8452Q

with MMA8452Q() as accelerometer:
    while True:
        print(accelerometer.get_xyz())
```

You can configure the range recorded by the accelerometer and change the output data rate (how fast the accelerometer is sampled):

```
import time
from microstacknode.hardware.accelerometer.mma8452q import MMA8452Q

with MMA8452Q() as accelerometer:
    # change the settings
    accelerometer.standby()  # call standby before changing settings
    accelerometer.set_g_range(2)  # Either 2G, 4G or 8G
    accelerometer.set_output_data_rate(800)
    accelerometer.activate()

    # let the accelerometer settle before reading again
    time.sleep(0.5)

    while True:
        print(accelerometer.get_xyz())
```

SHT21 Temperature and Humidity Sensor

This behaves very similar to the accelerometer:

```
>>> from microstacknode.hardware.humiditytemperature.sht21 import SHT21
>>> sht21 = SHT21()
>>> sht21.open()
>>> sht21.get_humidity()
>>> sht21.get_temperature()
```

or even using *with* (Python-magic will call *open()*):

```
with SHT21() as htsensor:
    while True:
        humidity = htsensor.get_humidity()
        temperature = htsensor.get_temperature()
        print('Humidity: {:.2f} %RH'.format(humidity))
        print('Temperature: {:.2f}°C'.format(temperature))
        print()
        time.sleep(1)
```

Reference:

CHAPTER 3

L80GPS GPS Module Reference

CHAPTER 4

MMA8452Q Accelerometer Reference

CHAPTER 5

SHT21 Humidity and Temperature Module Reference

CHAPTER 6

Indices and tables

- genindex
- modindex
- search