Makersnake

Software writing, 3D printing, hardware hacking and all-round geekery

Satellite communication with RockBLOCK

Makersnake 18/02/2015 Projects

The RockBLOCK Mk2 is one awesome module. It literally houses everything you need to send and receive data from anywhere on the planet (Iridium satellite modem, compact antenna and power circuitry) in a module smaller than the Raspberry Pi (the de facto measuring implement for these kind of things) at an equally micro price of £159/€199/$269 – available to buy online from Rock 7

This product has already been widely adopted by the Arduino community, with many happy RockBLOCK'ers thanks to **Mikal Hart's C++ IridiumSBD library**; but I couldn't find anything specifically for the many Python Raspberry Pi users… Fear not, inspired by Mikal's library, and copious amounts of Columbia's finest, coffee that is, I've created… **pyRockBLOCK.** The super-simple library that makes sending and receiving messages via satellite as simple as: sendMessage("Hello World") and requestMessageCheck().

pyRockBLOCK will work on pretty much anything that runs Python; so you'll be up and running with just a few lines of code, which will give you plenty of time to 3D print your very own **Raspberry Pi X RockBLOCK case** or create all manner of crazy **UAV's, High Altitude Balloons** and **Ocean Sensing Buoys** using your RockBLOCK!

Related Makersnake Projects/Guides:

- Remote Controlled Weather Station
- Idiots guide to Iridium
- Rock 7 Core for RockBLOCK users
- Rock7 Core – Endpoints

Part 1 – Raspberry Pi Setup (Software)

I've started off with a fresh installation of Raspbian – if you're new to Raspberry Pi (where have you been hiding?) here's a **great starter guide** to get your RPi up and running.

The first thing we'll do is install some dependencies and a couple of useful tools; needless to say, ensure that your RPi has an internet connection so that these can be downloaded!

From the command line (running as root/sudo):

```
apt-get install python-pip
pip install pyserial
apt-get install screen
apt-get install usbutils
apt-get install git
```

That's everything installed – next we'll clone the latest version of **pyRockBLOCK** from GitHub (https://github.com/MakerSnake/pyRockBlock)

```
cd /home/pi/
git clone https://github.com/MakerSnake/pyRockBlock
```

Part 2 – Raspberry Pi Setup (Wiring)

We've got two options when it comes to connecting the RPi to the RockBLOCK – via USB (using a USB to UART cable) or directly to the GPIO pins. Both are technically the same, so really just depends on your project or what cables you have laying around!

Option 1: USB (using a USB to UART cable) | Option 2: Directly to the GPIO pins

A FTDI USB to UART cable converts 5V USB signals to 3.3V UART as well as providing 5V power to the module.

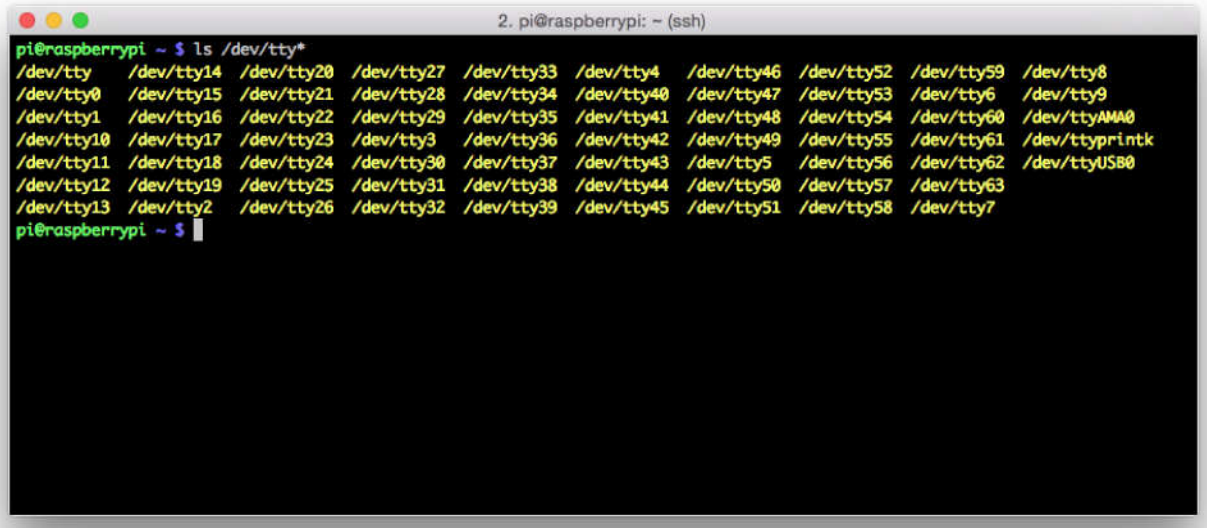
The inline 6-way connector should be connected as below (Black to GND, Green to RTS)

We ain't got the power captain!

When the RockBLOCK is powered via USB at 5V it will draw a maximum of 450mA – it's therefore important that your host (RPi) is itself connected to a good quality power supply that is capable of providing a reliable source of power.

Another important point, that caught me out, is that it's essential that the FTDI USB to UART cable is *configured* to supply the required current (who knew you could configure a cable!). The FTDI cable used has an accompanying programming utility for setting the required current value.

After connecting the USB cable to the RPi it may take a few seconds for it to be detected; you can see if it has been recognised using: **ls /dev/tty***



The device handle for your cable will vary depending on your cable/system – but typically if you're using a USB/UART converter as described, it will normally include "USB" somewhere in the name (e.g. **/dev/ttyUSB0** or **/dev/tty.usbserial-FTH9I1S5**)

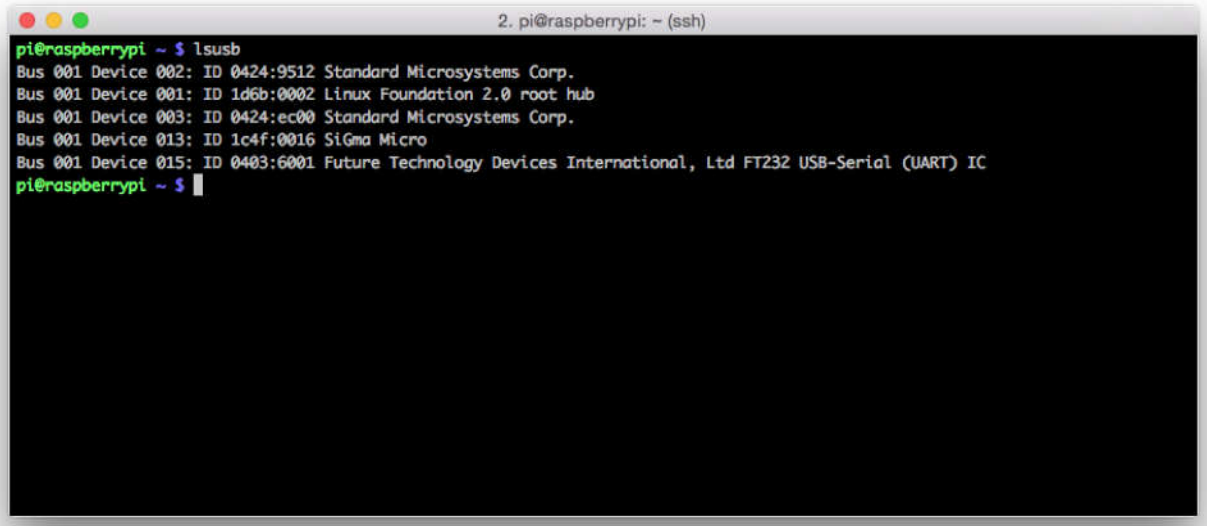
If you're not sure, or you've got multiple USB peripherals attached – try running the command with/without the peripheral connected and compare the results.

[checklist]

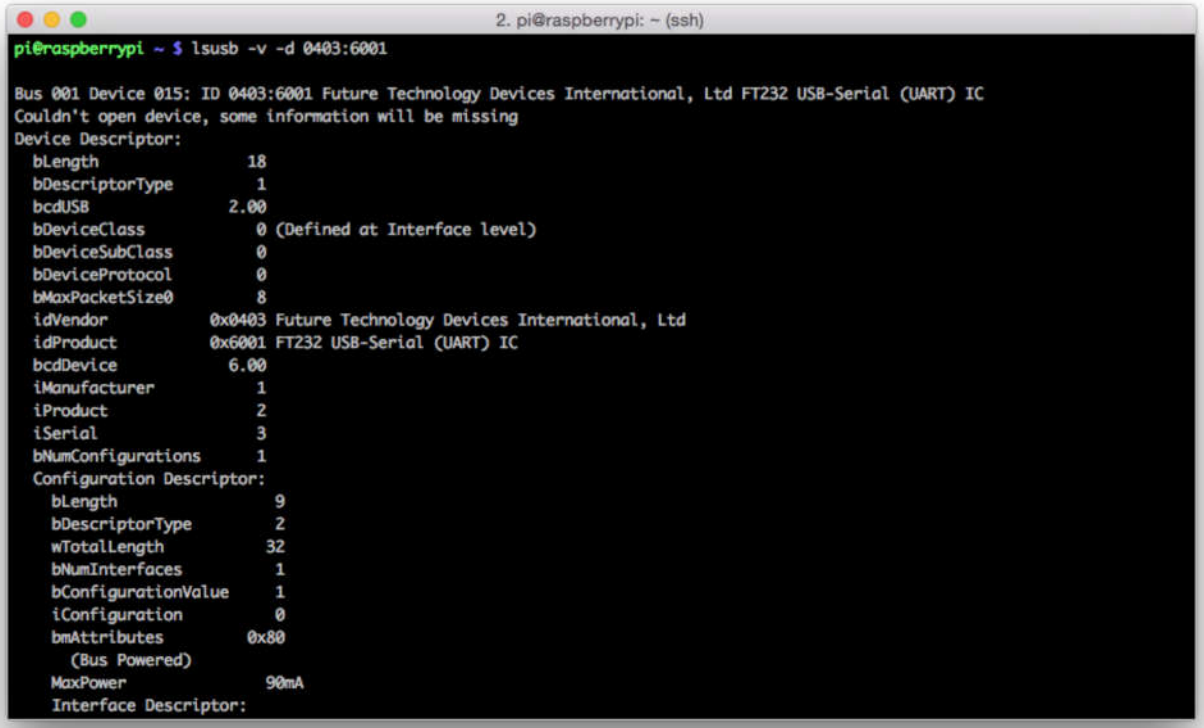
- Make sure you make a note of your device handle, this will be required later!

[/checklist]

Another useful command is *lsusb*, which lists the USB peripherals connected to your device:



You can request more information, including the MaxPower value for your cable, using the following command: *lsusb -v -d VENDOR:PRODUCT*



The Iridium modem requires a considerable amount of current whilst transmitting, that's why you'll notice the large "super capacitor" on the RockBLOCK. As such, it will take a while for the RockBLOCK to power up and begin to respond to commands (typically 60 seconds) whilst the capacitor is charging.

For more detailed information on the power usage profile of the RockBLOCK, and some pretty graphs, check out the official developer guide.

Before we start looking at the library, let's first check that we can talk to the RockBLOCK; like most modems, the **Iridium 9602 modem** on the RockBLOCK has a plethora of AT commands to control it's functionality. We'll use the **screen** command (if you're using a PC you can use PuTTY) to send commands to the modem and assess the response.

screen DEVICE HANDLE 19200 (e.g. screen /dev/ttyUSB0 19200 or /dev/ttyAMA0 19200)

NB: 19200 is the baud rate of the RockBLOCK

Now if you type **AT** and press [Enter], you should should see an **OK** response. If you don't get any response, wait 30 seconds and try again – as mentioned above it may take 60 seconds for the RockBLOCK to fully power up and respond to commands.

To get out of **screen** press [Ctrl] + a, then press k, then press y (Not totally obvious – I had to Google it!)

Part 3 – Software Setup

Now that we've connected the RockBLOCK and confirmed that we can talk to it, we can start writing some code…. Unfortunately, there's not much code to write, **pyRockBLOCK** does all the hard work for you and has been rigorously tested (it's been hanging out my window, running my Remote Controlled Weather Station for the last three weeks!).

pyRockBLOCK provides super-simple commands enabling you to send and receive messages with just a few lines of code; without having to worry about the nitty-gritty of messing around with Serial Ports, AT commands, signal strength, calculating checksums, retry strategies or making the perfect coffee (only joking, it doesn't do the latte, err doesn't do the latter!).

If you are happy to embrace this black-box, skip the next few paragraphs and jump to the good stuff – else, if you'd like to know how all this magic works, grab a coffee and read on…

> ***Mobile Originated (MO) messages (Sent from the RockBLOCK via Satellite)***
>
> *Given that your RockBLOCK is tiny and has to communicate with a satellite up in space (781km away) that is zipping across the sky (at 27,000km/h) – it's not always possible to get a reliable connection first time, as such the library faciilitates an element retry.*
>
> *When you try to and send a message the library will first query the signal strength from the modem (AT+CSQ). The signal strength is expressed on a 0 to 5 scale, where 0 is no signal, and 5 is perfect signal – just like the signal bars on your mobile/cell phone.*
>
> *The signal strength is only updated every few seconds and is approximate, so even with a signal strength of 5 transmissions can still fail, conversely a signal strength of 1 can be acceptable for transmission. So really its just an approximate guide, that may, or may not be correct (insert pinch of salt here)!*
>
> *For the library, i'll consider that a signal strength of 2 to be acceptable to attempt to establish a satellite session – if the signal strength is iess than this, we'll go to sleep for a bit and then retry, this will occur a maximum of 10 times before permanently failing.*
>
> *After verifying the signal strength, we'll write the message to the modem's output buffer (AT+SBDWB) and attempt then to initiate the satellite session (AT+SBDIX). If this fails, it often does, it will try again a maximum of 3 times automatically before permanently failing.*
>
> *It will typically take 20-30 seconds to establish a satellite connection, but given the various levels of retry, I would only start to panic if this process takes more than a couple of minutes!*
>
> ***Mobile Terminated (MT) messages (Sent to the RockBLOCK via Satellite)***
>
> *To receive a message on your RockBLOCK you'll firstly need to request it from the satellite – messages will not be automatically "pushed" you need to "pull" it (like a cracker).*
>
> *When you request the library to check for messages, it will undertake the signal strength check (AT+CSQ) and attempt to establish a satellite session (AT+SBDIX) – if successful, your message will be downloaded (only one message can be downloaded in each satellite session).*
>
> *The response from the satellite will indicate if there are additional messages to download; if there are additional messages "in the queue", you'll need to check for messages again; this will establish another satellite session, this should be repeated until there are no further messages in the queue.*

pyRockBLOCK is built around a single module called **rockBlock** which is instantiated by providing a device handle string (e.g. /dev/ttyUS0) and a reference to your *callback object*. The callback object should extend the **rockBlockProtocol** class and implement the applicable methods (e.g. ***rockBlockConnected***, ***rockBlockDisconnected***, ***rockBlockMessageReceived*** etc).

Messages from the Raspberry Pi to RockBLOCK (MO)

This simple example connects to the RockBLOCK and attempts to send a message – *rockBlockTxStarted* is called to mark the start of the process, and when the successfully sent *rockBlockTxSuccess* will be called along with the "MOMSN". This is a unique message identifier for the transmission, you can search for this in the Rock 7 Core system. Alternatively, if the message fails to transmit ***rockBlockTxFailed*** will be called. Make sure that you call ***close*** to disconnect from the RockBLOCK.

```
import rockBlock

from rockBlock import rockBlockProtocol

class MoExample (rockBlockProtocol):

    def main(self):

        rb = rockBlock.rockBlock("/dev/ttyUSB0", self)

        rb.sendMessage("Hello World RockBLOCK!")

    rb.close()

    def rockBlockTxStarted(self):
        print "rockBlockTxStarted"

    def rockBlockTxFailed(self):
        print "rockBlockTxFailed"

    def rockBlockTxSuccess(self,momsn):
        print "rockBlockTxSuccess " + str(momsn)

if __name__ == '__main__':
    MoExample().main()
```

Messages to the Raspberry Pi from RockBLOCK (via Space) (MT)

Receiving messages on your RockBLOCK is equally as simple – after you've queued a message with the satellites (using the Rock 7 Core, or the accompanying web-service) your message can be requested by the RockBLOCK by calling **requestMessageCheck**. This will attempt to establish a satellite session – if you've got a message, you'll see a callback to **rockBlockRxReceived** with the unique message identifier (mtmsn) and the message payload (data) as a byte array.

You'll also receive a callback to **rockBlockRxMessageQueue**, this will tell you how many (if any) additional messages are available to download. Remember, that you can only download one message per satellite session. Therefore, to get additional messages, you'll need to call **requestMessageCheck** until *count* in **rockBlockRxMessageQueue** is zero. Make sure that you call ***close*** to disconnect from the RockBLOCK when you're done.

```
import rockBlock

from rockBlock import rockBlockProtocol

class mtExample (rockBlockProtocol):

    def main(self):

        rb = rockBlock.rockBlock("/dev/ttyUSB0", self)

        rb.requestMessageCheck()

        rb.close()

    def rockBlockRxStarted(self):
        print "rockBlockRxStarted"

    def rockBlockRxFailed(self):
        print "rockBlockRxFailed"

    def rockBlockRxReceived(self,mtmsn,data):
        print "rockBlockRxReceived " + str(mtmsn) + " " + data

    def rockBlockRxMessageQueue(self,count):
        print "rockBlockRxMessageQueue " + str(count)


if __name__ == '__main__':
    mtExample().main()
```

That's it – you can now send and receive messages from anywhere in the world!

To see **pyRockBLOCK** in action have a look at the Remote Controlled Weather Station project.

← PREVIOUS
Remote Controlled Weather Station
NEXT –
Sneak Peak at the new Apple Watch

HELLO MAKERS

To get in touch with feedback, complaints or suggestions - hello@makersnake.com