

APRS Telemetry Toolkit

December 2015

CONTENTS

| | | |
|-----|--------------------------------------------------|----|
| 1 | Introduction | 2 |
| 1.1 | Required software..... | 2 |
| 2 | Data Formats..... | 4 |
| 2.1 | Original Telemetry Report Format..... | 4 |
| 2.2 | Base 91 compressed format | 4 |
| 3 | Metadata..... | 6 |
| 3.1 | Parameter Name Message..... | 6 |
| 3.2 | Unit/Label Message | 6 |
| 3.3 | Equation Coefficients Message..... | 7 |
| 3.4 | Bit Sense / Project Name Message | 7 |
| 4 | Balloon Example..... | 8 |
| 4.1 | Historical data | 8 |
| 4.2 | Let's try to replicate it..... | 9 |
| 4.3 | Verification..... | 9 |
| 5 | Raspberry Pi Example..... | 12 |
| 5.1 | Additional materials required:..... | 12 |
| 5.2 | Wire up the A/D converter | 12 |
| 5.3 | Install Kernel support for I ² C..... | 13 |
| 5.4 | Test the I ² C support | 13 |
| 5.5 | Install A/D software | 14 |
| 5.6 | Send Telemetry | 14 |

1 Introduction

Many people have asked about how to send telemetry with Dire Wolf.

A little APRS tracker box might be configurable to send temperature from an internal sensor and a voltage from some external source. There we have certain known fixed hardware. Just enable the option and off you go.

What we have here is much different. It is not a complete ready to go turnkey solution.

This is a collection of scripts and examples that can be used as building blocks for constructing your own customized solutions for your particular needs. It will take some effort to adapt this to your particular situation.

To enhance your learning experience, you are encouraged to work along with the examples here and experiment with your own variations. Disconnect any transmitter to avoid sending anything inappropriate over the air.

1.1 Required software

- Dire Wolf, version 1.3 or later.

Download from here: <https://github.com/wb2osz/direwolf/releases/>

For Windows, use the file with “win” in its name.

The other two are source for Linux in both zip and tar compressed tar formats.

- APRS Telemetry Toolkit

This is bundled in with Dire Wolf version 1.3 and later. It is composed of the following files:

| Name | Purpose | Linux Location |
|------------------------------------------------------------------|------------------------------------------------------------------------------|-----------------------------------------|
| telem-bits.pl telem-eqns.pl telem-parm.pl telem-unit.pl | Generic scripts for sending metadata which describes the data. | /usr/local/bin |
| telem-data.pl telem-data91.pl | Convert data values to original telemetry and more recent compressed format. | /usr/local/bin |
| telem-m0xer-3.txt | Actual historical data captured and used for balloon example. | \$HOME /usr/local/share/doc/direwolf |
| telem-balloon.conf | Configuration file used for reenactment of the historical event. | \$HOME /usr/local/share/doc/direwolf |

| | | |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| telem-balloon.pl | Script used for reenactment. This uses hard coded values for a demonstration. In practice it would obtain values from sensors. | /usr/local/bin |
| telem-volts.py | Script to read voltage from A/D converter on a Raspberry Pi. | /usr/local/bin |
| telem-volts.conf | Configuration file to read voltage from A/D converter and send as telemetry. | \$HOME /usr/local/share/doc/direwolf |
| APRS-Telemetry-Toolkit.pdf | This document. | /usr/local/share/direwolf/doc |

In the Windows version, all of the files are placed in a single directory when extracting from the zip file.

- Perl

For Microsoft Windows, I happen to be using the version from here: <http://strawberryperl.com/> Other versions will probably work as well.

For Linux, perl is probably there already. If not, install it:

Debian / Ubuntu / Raspbian: `sudo apt-get install perl`

Red Hat / Fedora / CentOS: `sudo yum install perl`

2 Data Formats

First, let's quickly review the fundamentals of APRS Telemetry. The specification is found in the **APRS Protocol Reference**, <http://www.aprs.org/doc/APRS101.PDF> , chapter 13.

2.1 Original Telemetry Report Format

The original format used fixed width columns like this:

```
T#xxx,aaa,aaa,aaa,aaa,aaa,bbbbbbbb
```

Where,

| | |
|----------|--------------------------------------------------------------|
| T | is the data type indicator for Telemetry data. |
| xxx | is a sequence number. |
| aaa | are up to 5 analog values in the range of 000 to 255. |
| bbbbbbbb | is an 8 bit binary number for single bit digital values. |
| | All analog values must be present to use the digital values. |

In reality, no one pays attention to the requirement for fixed width fields or the analog value range. Much longer numbers with decimal points are often seen and all of the applications that I looked at know how to deal with this.

The toolkit contains a script to combine the command line arguments into the proper format. Using an example from the Protocol Reference,

```
$ telem-data.pl 005 199 000 255 073 123 01101001  
T#005,199,000,255,073,123,01101001
```

The first command line argument is the sequence number. Next, up to 5 analog values. We don't restrict it to integers in the 000 to 255 range. If all 5 analog values are specified, an 8 bit binary value can be provided at the end.

2.2 Base 91 compressed format

In 2012, a new compressed format was introduced.
<http://he.fi/doc/aprs-base91-comment-telemetry.txt>

Rather than sending telemetry in its own packet, this is inserted into the comment part of a Position Report. It can be recognized by the vertical bar at the beginning and end. Inside are pairs of characters representing integers in the range of 0 to 8280.

```
|ssaaaaaaaaadd|
```

Where,

ss is the sequence number
aa are 1 to 5 analog values.
dd contains 8 bit values and some left over space.
Again, all analog values must be present before using the digital values.

Examples of script usage:

```
$ telem-data91.pl 7458 4521 587 2649 7  
|rxR_'J>+! (|
```

```
$ telem-data91.pl 7544 1472 1564 1656 1748 1840 10000000  
|ss1122334455!"|
```

Again, we specify a sequence number, up to 5 analog values, and an optional 8 bit digital value if all 5 analog values were used. The resulting encoding is more compact.

3 Metadata

Numbers alone are not very informative. Does the value represent battery voltage, river water level, or a Field Day score? Is the Jacuzzi temperature 100 degrees Fahrenheit or Celsius? In the cases where only integers, of a limited range, are available, some sort of scaling is often necessary to convey desired values.

This information is sent as APRS “Messages” in a specific format. All of them include the callsign of the station sending telemetry, and a type identifier.

```
:call-ssid:PARM. (names for up to 5 analog and 8 digital channels)
:call-ssid:UNIT. (units for analog or labels for digital channels)
:call-ssid:EQNS. (equation coefficients for scaling values)
:call-ssid:BITS. (bit polarity and project title)
```

Consult the APRS Protocol Reference for more details. Here are brief descriptions of the scripts provided to generate these messages in the proper format.

3.1 Parameter Name Message

The Parameter Name Message contains the names of the 5 analog and 8 digital channels.

Example from Protocol Reference:

```
$ telem-parm.pl N0QBF-11 Battery Btemp ATemp Pres Alt Camra Chut Sun 10m ATV
:N0QBF-11 :PARM.Battery,Btemp,ATemp,Pres,Alt,Camra,Chut,Sun,10m,ATV
```

The first command line argument is the station name. Up to 13 parameter names can be specified.

The result is what goes into the information part of an AX.25 frame. There will be another step, explained later, to generate a frame containing this. Same for the next three.

3.2 Unit/Label Message

The Unit/Label Message contains the units for the analog values and labels for digital channels.

Example from Protocol Reference:

```
$ telem-unit.pl N0QBF-11 v/100 deg.F deg.F Mbar Kft Click OPEN on on hi
:N0QBF-11 :UNIT.v/100,deg.F,deg.F,Mbar,Kft,Click,OPEN,on,on,hi
```

The first command line argument is the station name. Up to 13 units/labels can be specified.

3.3 Equation Coefficients Message

The Equation Coefficients are used to expand limited range integer values into more convenient units.

Example from Protocol Reference:

```
$ telem-eqns.pl N0QBF-11 0 5.2 0 0 .53 -32 3 4.39 49 -32 3 18 1 2 3
:N0QBF-11 :EQNS.0,5.2,0,0,.53,-32,3,4.39,49,-32,3,18,1,2,3
```

The first command line argument is the station name. Sets of 3 coefficients are specified for the analog values.

Expand on this....

3.4 Bit Sense / Project Name Message

This APRS "message" defines bit polarity and an optional title for the project.

Another example from Protocol Reference:

```
$ telem-bits.pl N0QBF-11 10110000 "N0QBF's Big Balloon"
:N0QBF-11 :BITS.10110000,N0QBF's Big Balloon
```

The definition of the Bit Sense / Project Name Message, page 70 of the APRS protocol spec, does not show a comma after the fixed length of eight binary digits. However, the example does have a comma which seems more sensible.

4 Balloon Example

For our first example, we will perform a terse reenactment of an historical event. This should work the same way on both Windows and Linux.

4.1 Historical data

Here is some actual data gather from a high altitude balloon flight.

```
2E0TOY>APRS::M0XER-3 :BITS.11111111,10mW research balloon
2E0TOY>APRS::M0XER-3 :PARM.Vbat,Vsolar,Temp,Sat
2E0TOY>APRS::M0XER-3 :EQNS.0,0.001,0,0,0.001,0,0,0.1,-273.2,0,1,0,0,1,0
2E0TOY>APRS::M0XER-3 :UNIT.V,V,C,,m
M0XER-3>APRS63,WIDE2-1:!/Bap'.ZGO JHAE/A=042496|E@Q0%i;5!-|
M0XER-3>APRS63,WIDE2-1:!/4\;u/)K$O J]YD/A=041216|h`RY(1>q!(|
M0XER-3>APRS63,WIDE2-1:!/23*f/R$UO Jf'x/A=041600|rxR_'J>+!(|
```

What does it all mean? One way to find out is to put it into a text file and feed it into the “decode_aprs” utility. I found it interesting that the balloon did not send its own metadata. That was provided by someone else. This required some extra effort to find the information in different places and bring it all together.

```
$ decode_aprs telem-m0xer-3.txt
```

```
2E0TOY>APRS::M0XER-3 :BITS.11111111,10mW research balloon
Telemetry Bit Sense/Project Name Message for "M0XER-3", Generic,
(obsolete. Digis should use APNxxx instead)

2E0TOY>APRS::M0XER-3 :PARM.Vbat,Vsolar,Temp,Sat
Telemetry Parameter Name Message for "M0XER-3", Generic, (obsolete.
Digis should use APNxxx instead)

2E0TOY>APRS::M0XER-3 :EQNS.0,0.001,0,0,0.001,0,0,0.1,-
273.2,0,1,0,0,1,0
Telemetry Equation Coefficient Message for "M0XER-3", Generic,
(obsolete. Digis should use APNxxx instead)

2E0TOY>APRS::M0XER-3 :UNIT.V,V,C,,m
Telemetry Unit/Label Message for "M0XER-3", Generic, (obsolete. Digis
should use APNxxx instead)

M0XER-3>APRS63,WIDE2-1:!/Bap'.ZGO JHAE/A=042496|E@Q0%i;5!-|
Position, BALLOON, Generic, (obsolete. Digis should use APNxxx instead)
N 61 34.2876, W 155 40.0931, alt 42496 ft
10mW research balloon: Seq=3307, Vbat=4.383 V, Vsolar=0.436 V, Temp=-
34.6 C, Sat=12
AE

M0XER-3>APRS63,WIDE2-1:!/4\;u/)K$O J]YD/A=041216|h`RY(1>q!(|
Position, BALLOON, Generic, (obsolete. Digis should use APNxxx instead)
N 51 07.4402, W 124 14.4472, alt 41216 ft
10mW research balloon: Seq=6524, Vbat=4.515 V, Vsolar=0.653 V, Temp=-
1.3 C, Sat=7
```


YD

```
MOXER-3>APRS63,WIDE2-1:!/23*f/R$UO Jf'x/A=041600|rxR_'J>+!(|
Position, BALLOON, Generic, (obsolete. Digis should use APNxxx instead)
N 55 58.5558, W 122 28.5933, alt 41600 ft
10mW research balloon: Seq=7458, Vbat=4.521 V, Vsolar=0.587 V, Temp=-
8.3 C, Sat=7
'x
```

If you try doing them one at a time, the metadata history won't be available and it will be displayed as a bunch of plain integers.

```
$ decode_aprs
MOXER-3>APRS63,WIDE2-1:!/23*f/R$UO Jf'x/A=041600|rxR_'J>+!(|

MOXER-3>APRS63,WIDE2-1:!/23*f/R$UO Jf'x/A=041600|rxR_'J>+!(|
Position, BALLOON, Generic, (obsolete. Digis should use APNxxx instead)
N 55 58.5558, W 122 28.5933, alt 41600 ft
Seq=7458, A1=4521, A2=587, A3=2649, A4=7
'x
```

Hmmmm. What are those two extra characters left over for the comment?

4.2 Let's try to replicate it

I'm not suggesting that this would be an appropriate implementation for something sent into the Stratosphere. This is just a demonstration of how the same type of packets could be generated. Start up Dire Wolf with the sample configuration file.

```
$ direwolf -c telem-ballon.conf
...
...
[0L] MOXER-3>APDW13::MOXER-3 :PARM.Vbat,Vsolar,Temp,Sat
[0L] MOXER-3>APDW13::MOXER-3 :UNIT.V,V,C,,m
[0L] MOXER-3>APDW13::MOXER-3 :EQNS.0,0.001,0,0,0.001,0,0,0.1,-
273.2,0,1,0,0,1,0
[0L] MOXER-3>APDW13::MOXER-3 :BITS.11111111,10mW research balloon

[0L] MOXER-3>APDW13,WIDE2-1:!/Bap'.ZGO !/A=042496|E@Q0%i;5!-|
[0L] MOXER-3>APDW13,WIDE2-1:!/4\;u/)K$O !/A=041217|h`RY(1>q!(|
[0L] MOXER-3>APDW13,WIDE2-1:!/23*f/R$UO !/A=041601|rxR_'J>+!(|
```

4.3 Verification

How do we know if it is correct? You could feed it into `decode_aprs` as we did before, or better yet, use some other independently developed applications and see if everyone agrees. If you wait a little longer, the sample configuration file will send the same packets again. This time, a new option is added:

```
SENDTO=R0
```

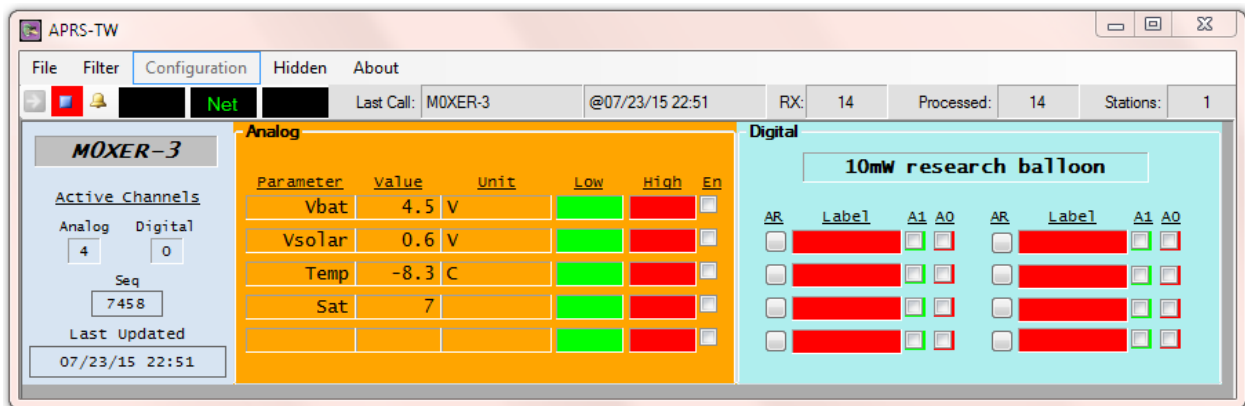
This means don't send the beacons to the transmitter. Instead process them as if they had been received over the radio. This is convenient for testing. You can see what happens if some particular packet is received without having to transmit it from some other system.

(Here is another idea that might be useful in some situations. Use the beacon "SENDTO=IG" option to send the packets through the IGate function to the APRS-IS servers. This could be used to report telemetry values without a radio link.)

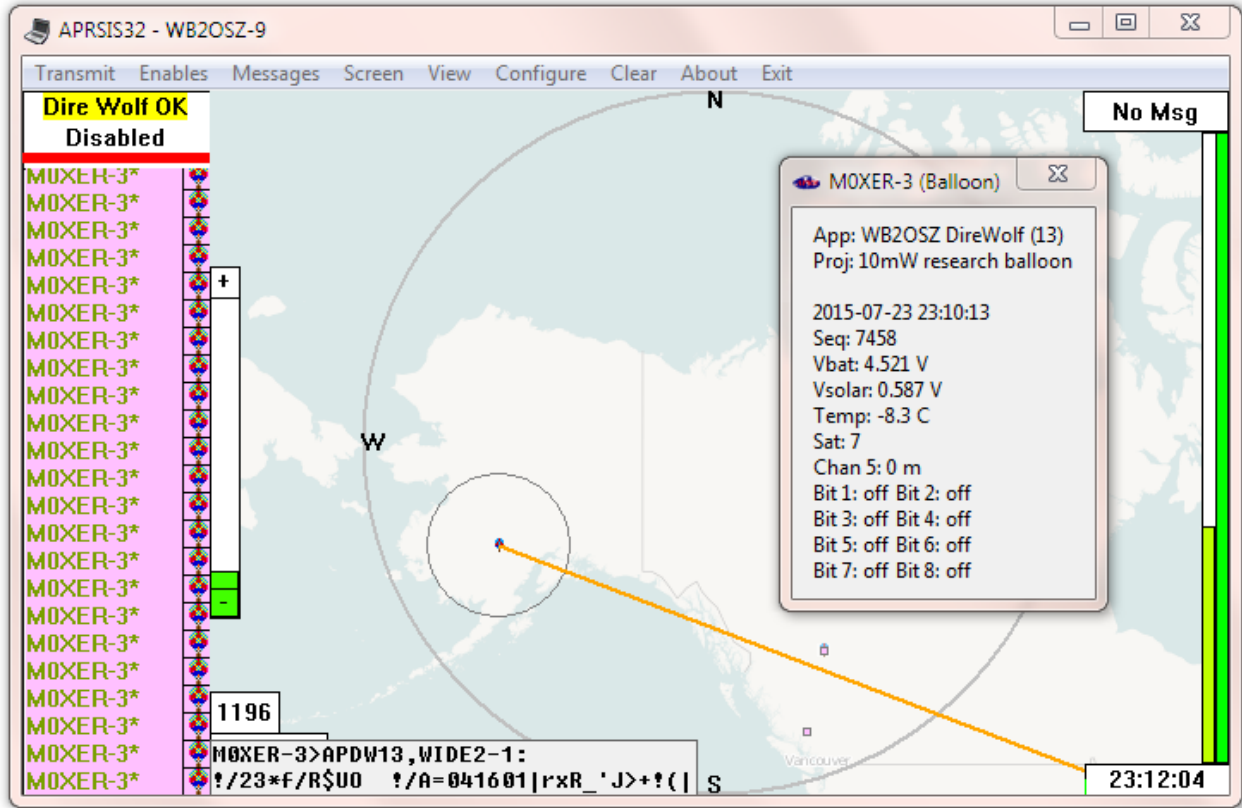
We will now run three different client applications at the same time and observe what they display when these packets are received. The first one listed has some interesting features like alarms when specified values are exceeded.

- APRS Telemetry Watcher (APRS-TW) <http://aprstw.blandranch.net/>
- APRSISCE/32 <http://aprsisce.wikidot.com/>
- Yet Another APRS Client (YAAC) <http://www.ka2ddo.org/ka2ddo/YAAC.html>

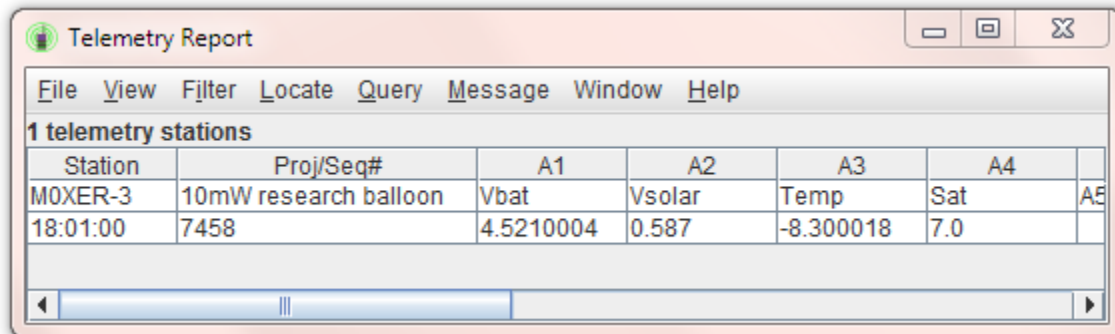
APRS Telemetry Watcher has the expected results.



APRSISCE/32 decodes it correctly.



YAAC also decodes the same values.



5 Raspberry Pi Example

In this example we will send actual data from an analog to digital converter. You can use it as a starting point for sending other data, such as temperature and humidity, from other types of sensors.

5.1 Additional materials required

- Raspberry Pi, any model

Install Dire Wolf as detailed in the included **Raspberry-Pi-APRS.pdf** document.

- Analog to digital converter

I happened to use this one: <http://www.adafruit.com/product/1085>

Software is available here: https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/tree/master/Adafruit_ADS1x15 We will download it in a later step.

Why this particular part? I picked a 16 bit version not knowing what I might want to measure in the future and how much resolution I might want in that case. The programmable gain allows measurement of very low voltages. For instance if you wanted to measure the accumulated current going in and out of a backup battery, you would need to measure a small fraction of a volt across a small resistor put in series with the battery. It would also need to be a differential measurement if one side was not at ground.

12 or 10 bits would be fine for measuring a battery voltage.

5.2 Wire up the A/D converter

To be continued.... Draw wiring diagram.

| Name | Break out board pin | RPi GPIO pin |
|-----------------|---------------------|---------------------|
| VDD (3.3 volts) | VDD | 1 or 17 |
| GND | GND | 6, 9, 14, 20, or 25 |
| SCL | SCL | 5 |
| SDA | SDA | 3 |
| | | |

5.3 Install Kernel support for I²C

This is a condensed version of <https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c>

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
sudo raspi-config
    (Select Advanced Options then I2C. Answer yes and yes.)
```

Next, we need to manually edit some files. This needs to be done as root so put “sudo” in front of your favorite editor command.

/etc/modules - Add these two lines to the end:

```
i2c-bcm2708
i2c-dev
```

/etc/modprobe.d/raspi-blacklist.conf - If the following line is present, **put # in front of it.**

```
blacklist i2c-bcm2708
```

/boot/config.txt – Verify that “raspi-config” added the following to the end of the file.

```
dtparam=i2c=on
dtparam=i2c_arm=on
```

Finally, we reboot.

```
sudo reboot
```

5.4 Test the I²C support

After it finishes rebooting, see if we can communicate with the I²C device.

```
sudo i2cdetect -y 1
```

You should see something like this:

```
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  48  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --
```

This indicates that an I²C device is present at address 0x48.

If you don't see this, go back and fix the problem rather than continuing with the next step.

If you run into difficulties, this information might be helpful:

<http://raspberrypi.stackexchange.com/questions/27073/firmware-3-18-x-breaks-i%C2%B2c-spi-audio-lirc-1-wire-e-g-dev-i2c-1-no-such-f>

5.5 Install A/D software

If the previous step was successful, we now have the ability to talk to an I²C device. Next, install software for talking to this specific analog to digital converter chip.

```
cd ~
git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code
cd Adafruit-Raspberry-Pi-Python-Code/Adafruit_ADS1x15
chmod +x *.py
```

Connect the analog input to the 5 volt supply or some other convenient voltage available then type:

```
cd ~/Adafruit-Raspberry-Pi-Python-Code/Adafruit_ADS1x15
./ads1x15_ex_singleended.py
```

In my case it responded with "0.449750." Is this correct? We used a voltage divider of 1M and 100k resistors so the A/D input should see $4.98 \text{ V} \times 100\text{k} / (1\text{M} + 100\text{k}) = 0.453 \text{ volts}$. We're very close.

We will make our own version of that script with a couple modifications. Change the full scale from 4096 to 2048 mV because we are using a 3.3 volt supply. Add scaling for the voltage divider on the input. The result is /usr/local/bin/telem-volts.py. Try running it:

```
cd ~
export PYTHONPATH=~/Adafruit-Raspberry-Pi-Python-Code/Adafruit_ADS1x15
telem-volts.py
```

In my case it responded with 4.889, about 2% low. If you need better precision, use higher precision resistors or put a calibration correction factor in your copy of the script.

5.6 Send Telemetry

In our previous example, we inserted compressed telemetry in a position report. In this case, we will use the traditional format. We will ignore the 000 – 255 value range restriction like everyone else does. Connect some sort of USB Audio adapter and run:

```
$ direwolf -c telem-volts.conf
```

You should see something like this:

```
[0L] MYCALL-9>APDW13,WIDE2-1::MYCALL-9 :PARM.Supply
[0L] MYCALL-9>APDW13,WIDE2-1::MYCALL-9 :UNIT.Volts
[0L] MYCALL-9>APDW13,WIDE2-1:T#1,4.808
[0L] MYCALL-9>APDW13,WIDE2-1:T#2,4.797
[0L] MYCALL-9>APDW13,WIDE2-1:T#3,4.808
[0L] MYCALL-9>APDW13,WIDE2-1:T#4,4.802
[0L] MYCALL-9>APDW13,WIDE2-1:T#5,4.807
...
```

6 Additional Reading

Raspberry Pi SPI and I²C Tutorial

<https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial>

Tradeoffs when considering SPI or I²C?

<http://electronics.stackexchange.com/questions/29037/tradeoffs-when-considering-spi-or-i2c>

I²C vs GPIO vs UART vs SPI - Super Noob Question

<https://www.raspberrypi.org/forums/viewtopic.php?t=51450&p=478418>