# Table of Contents

## Overview
- Parts  
  3

## System Overview

## Hardware
- Pinout  
  5
- Connector  
  6
- LED Indicators  
  7
- Antenna  
  7

## Service Setup
- Enabling Service  
  8

## Basic Comms Check
- AT Commands  
  10
- Basic Info  
  10
- Hello World  
  11
- What Are Those Status Numbers?  
  12

## CircuitPython Example
- Wiring  
  14
- CircuitPython Libraries  
  15
- Code  
  15
- Unpacking Data  
  17

## Raspberry Pi Example
- Wiring  
  19
- Enable Serial  
  19
- Install CircuitPython Library  
  21
- Run Simpletest Check  
  21
- Additional Examples  
  22

## Forwarding Messages
- Delivery Groups  
  24
- Adafruit IO  
  24

## More Info on Data Transmission
- TLDR  
  25
- Text vs. Binary  
  26
- Packing Data in Python  
  27
- Integer vs Float  
  28
Overview

Who's got the best coverage map? Anyone using the Iridium satellite constellation (https://adafruit.it/JFD), that's who. How does whole Earth coverage sound? Summit of K2 to summit of Everest? Check. South pole to North pole? Super check. In the animated image above, red is good. Check that coverage at the poles!

So how do you use the Iridium satellites? Similar to cellphones, you need to purchase both hardware and service, and there are various options out there. In this guide we will show you how to setup and use the Rock Seven (https://adafruit.it/JFE) RockBLOCK 9603 (https://adafruit.it/JFF) Iridium Satellite Modem. This hardware is tied to service that is also provided by Rock Seven (https://adafruit.it/JFE).

We also have a CircuitPython (https://adafruit.it/EFq) library that provides an easy way to transmit messages from any CircuitPython board (https://adafruit.it/Em8) with an available UART.

Parts

RockBLOCK 9603 with USB Cable - Iridium Satellite Modem Bundle
The RockBLOCK 9603 allows you to send and receive short messages from anywhere on Earth with a clear view of the sky. And when we say anywhere we mean...
https://www.adafruit.com/product/4521
System Overview

The Rock Seven RockBLOCK 9603 works with the Iridium Satellite System (https://adafruit.it/JFD) and the Rock Seven (https://adafruit.it/JFE) Servers to connect you between two points on Earth via space. That's right. That little square thing is talking to outer space!

The graphic above provides a general overview of the whole system. In the middle is the RockBLOCK 9603 modem (https://adafruit.it/JHF) itself. Data can flow in either direction.

1. The RockBLOCK 9603 communicates with the Iridium satellites to send/receive data.
2. The Iridium satellites communicate with ground based stations to send/receive data between space and Earth.
3. Data is sent between the Iridium ground stations and the Rock Seven Servers. You can access this server via a web interface.
4. Your specific application talks to the Rock Seven servers.

Hardware

There's not much to the little RockBLOCK 9603 itself. There's only one connector for both power and data. There are a couple of status LEDs. Besides some mounting holes, that's pretty much it. A nice tidy little package.

Pinout

1. RXD Serial output from RockBLOCK, so it's really TX
2. CTS Clear to Send (output from RockBLOCK)
3. RTS Ready To Send (input to RockBLOCK)
4. NET Network Available
5. RI Ring Indicator (active low)
6. TXD Serial input to RockBLOCK, so it's really RX
7. SLP Sleep control (pull to ground to switch off)
8. 5V 5V in power supply (450mA limit)
9. BAT 3.7V power supply (450mA limit)
10. GND Ground

The RockBLOCK 9603 operates at 3.3V logic level.

Serial Baud Rate is 19200.

**Connector**

The connector on the RockBLOCK modem is a Molex PicoBlade ([https://adafru.it/Jla](https://adafru.it/Jla)). The mating connector housing is Molex part number 51021-1000. Bare connectors are available in Molex product series 50079 or 50058, but need a special (very expensive) crimping tool. There are also pre-crimped cables available in Molex product series 79758.

You can get these on DigiKey:

- **Connector housing** ([https://adafru.it/Jlb](https://adafru.it/Jlb))
- Pre-crimped cables
  - 6 inch ([https://adafru.it/Jlc](https://adafru.it/Jlc))
  - 12 inch ([https://adafru.it/Jld](https://adafru.it/Jld))

With these parts you can make custom cable assemblies.

You do not need these items if your are using the supplied USB cable or the available accessory cable.
LED Indicators

There are two small LED indicators on the side with the big capacitors.

- Red = DC power present.
- Green = Capacitors are charged. This is needed for satellite communication.

Antenna

This is the antenna which talks to the satellites. Make sure it has a good clear view of the sky. Butter side up!

An external antenna can be attached via the coaxial connector on the side. But in this guide we show basic usage with the built in antenna.
Service Setup

Rock Seven provides a web based interface for managing your account which you can access at http://www.rock7.com/register (https://adafru.it/JIe). General information about using this system is here:

RockBLOCK Management System
https://adafru.it/JIf

Once you've created an account and logged in, you can register your specific Rock BLOCK 9603 modem and enable (purchase) service.

Enabling Service

In addition to the modem hardware, you will also need to purchase line rental AND credits to use the service.

- Line Rental - Purchased in one month increments, currently £12.00 per month. Needed to communicate with the Iridium satellites. Expires after the purchased time has elapsed. This is a fixed base cost.
- Credits - Needed to send/receive data through the Iridium satellites. These get consumed at a rate of about 1 credit / 50 bytes sent/received. They never expire. Cost varies depending on how many you buy at once, from £0.13 to £0.045 per credit. This is a variable cost.

You can purchase the above through the web based management system once logged in to your account.

More info here:
Basic Comms Check

The easiest way to get started with the RockBLOCK 9603 is to use the included USB serial cable.

This cable contains an FTDI FT232R (https://adafru.it/JIB) chip. You may need to install drivers, which are available here:

FTDI Virtual COM Port Drivers
https://adafru.it/ENP

The RockBLOCK 9603 can not communicate with the Iridium satellites indoors. So to actually try a satellite data transfer, you'll need to somehow have the antenna obtain a view of the sky. We just placed it on an open window sill and it worked OK. If you have a laptop, that could make things easier.

Once you have the COM port showing up on your PC, use a terminal program to connect. The serial parameters are:

- Baud Rate = 19200
- Data Bits = 8
- Parity = N
- Stop Bits = 1
The examples here show using `screen` on a Linux machine.

### AT Commands

The RockBLOCK 9603 uses an AT command set that is documented here:

AT Command Reference
https://adafruit.it/JIC

All commands start with `AT` followed by the specific command and end with a carriage return (`\r`). There are a lot of commands, but only a few really matter. Here's a short list of the important ones:

- `+SBDIX` - Initiate an Short Burst Data session, i.e. talk to the satellites. Make sure you have loaded message data first.
- `+SBDWT` - Write text message into outbound buffer
- `+SBDWB` - Write binary data into outbound buffer
- `+SBDRT` - Read text message from inbound buffer
- `+SBDRB` - Read binary data in from inbound buffer
- `+SBDSX` - Status

For example, to get status, you would send:

```
AT+SBDSX\r
```

If you are using a terminal program, the `\r` is typically sent when you press the Enter key. So you don't actually type it. But when you are writing custom software to talk directly over serial, then you'll need to remember to add a `\r` to your data string.

### Basic Info

Before we try a satellite data transfer, let's make sure the basics work. We'll use the commands `+CGMI` and `+CGMM` to obtain model information. This information comes directly from the RockBLOCK 9603 modem and does not require any satellite access.
Launch `screen` or whatever terminal program you are using.

Set baud rate to 19200. For `screen`, that's a command line parameter.

Then type AT and press Enter. You don't need to actually enter the \r.

If you get the OK response, you're talking. Then try AT+CGMI and AT+CGMM and you should get something like the info shown.

Hello World

OK, fun part time. Let's try and send a message. If you aren't already connected to the RockBLOCK, see the previous section. You also need to have your service enabled by purchasing line rental and credits.

For this initial test we'll use text. Later we'll use binary data, which is generally the way all the communication should be done. Text is just a convenience feature.

Once you are connected, type AT+SBDWT=Hello World and then press Enter.

You should receive the response OK.
To send the message, type AT+SBDIX and press Enter.

It will attempt to talk to the satellites and after a period of time return a status as +SBDIX: 32, 8, 2, 0, 0, 0.

If the first number is not 0, then the message did NOT transmit and you'll need to try again.

Keep entering the AT+SBDIX command until the first number in the status return is 0.

You may have to try this several times depending on how good your view of the sky is, where the satellites are, etc. Wait 10s of seconds between each try. Don't spam the sats, yo.

If you now log into your account and got to Messages, you should see the arrived message with "Hello World" in the payload.

There are no credits consumed for the failed attempts, so don't worry about the number of retries. Only the one final successful attempt actually deducts from your credit balance.

What Are Those Status Numbers?

The status values, the six numbers, returned from a call to AT+SBDIX are:
MO status, MOMSN, MT status, MTMSN, MT length, MT queued

where:

- MO status = status of outgoing transmission
- MOMSN = outbound sequence number
- MT status = status of inbound transmission
- MTMSN = inbound sequence number
- MT length = bytes received
- MT queued = messages waiting to be delivered

The first one, MO status, is the most important for determining transmission success. There is a full list of possible values and their meaning in the AT Command Reference (https://adafru.it/JIC). Briefly:

- 0 - 4 = Transmit successful
- 32 = No network service

And for completeness:

- MO = Mobile Originated, i.e. from the RockBLOCK
- MT = Mobile Terminated, i.e. to the RockBLOCK

CircuitPython Example

Actual applications will most likely end up using some form of microcontroller to control the RockBLOCK. We've written a CircuitPython library (https://adafru.it/JID) to allow you to do this using an available CircuitPython board (https://adafru.it/Em8).
In this example, we'll use a Feather nRF52840 Sense (https://adafruit.it/JA1) to send data from the onboard sensors. You'll also need the accessory cable (https://adafruit.it/JIE) to attach the RockBLOCK to the Feather.

### Adafruit Feather nRF52840 Sense
The Adafruit Feather Bluefruit Sense takes our popular Feather nRF52840 Express and adds a smorgasbord of sensors...

https://www.adafruit.com/product/4516

### RockBLOCK 9603 Accessory Cable
The RockBLOCK 9603 Accessory Cable is terminated with a simple 10-pin Molex-style connector at one end and male jumper wires at the other. This accessory cable...

https://www.adafruit.com/product/4529

### Wiring

Here’s the wiring diagram. If you use the accessory cable, you should be able to match the colors. But it's also best to verify actual pin location on the RockBLOCK.

- GND to GND
- USB to 5V*
- RX to RXD
- TX to TXD
* in this example we will power everything via the Feather's micro USB connector (no battery).

**CircuitPython Libraries**

You'll need the following CircuitPython libraries (https://adafru.it/ENC) installed. Make sure all of these are in your CIRCUITPY/lib folder.

- adafruit_apds9960
- adafruit_bus_device
- adafruit_register
- adafruit_bmp280
- adafruit_lis3mdl
- adafruit_lsm6ds
- adafruit_rockblock
- adafruit_sht31d

![Folder showing CircuitPython libraries](image)

**Code**

The code used is included as an example in the library. Here it is:

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import struct
import board
import adafruit_lsm6ds
import adafruit_lis3mdl
import adafruit_apds9960.apds9960
import adafruit_sht31d
import adafruit_bmp280
import adafruit_rockblock
```
# RockBlock setup
uart = board.UART()
uart.baudrate = 19200
rb = adafruit_rockblock.RockBlock(uart)

# all the sensors
accelo = adafruit_lsm6ds.LSM6DS33(board.I2C())
magno = adafruit_lis3mdl.LIS3MDL(board.I2C())
prox = adafruit_apds9960.APDS9960(board.I2C())
sht = adafruit_sht31d.SHT31D(board.I2C())
bmp = adafruit_bmp280.Adafruit_BMP280_I2C(board.I2C())

# build data
# can decode on other end with struct.unpack("<6fB5f", data)
data = struct.pack("3f", *accelo.acceleration)
data += struct.pack("3f", *magno.magnetic)
data += struct.pack("B", prox.proximity())
data += struct.pack("2f", sht.relative_humidity, sht.temperature)
data += struct.pack("3f", bmp.pressure, bmp.altitude, bmp.temperature)

# send data
rb.data_out = data
print("Talking to satellite...")
retry = 0
status = rb.satellite_transfer()
while status[0] > 8:
    time.sleep(10)
    status = rb.satellite_transfer()
    print(retry, status)
    retry += 1
print("DONE.")

Save that code as code.py to your CIRCUITPY folder. If you need to do a soft-reboot, you can with <CTRL>-<D> in the REPL. It will run and display the status of the attempts to send the message:

The output will show the number of attempts and the status (the numbers in parans). Just as in the simple test done before, the first number needs to be 0 for success. So it will keep trying until that happens:
Once it is successful, it will exit and is done.

If you then go to your account on the Rock Seven server and look in your Messages, you should see something like:

<table>
<thead>
<tr>
<th>Date Time (UTC)</th>
<th>Device</th>
<th>Direction</th>
<th>Payload</th>
<th>Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17/Mar/2020 20:01:13</td>
<td>Rock7LOCK</td>
<td>MO</td>
<td>88984be9be9920fb21d41f3754c2081c3f410aed74188277a44b42a2ccbe7fdv</td>
<td>40</td>
</tr>
<tr>
<td>17/Mar/2020 20:01:13</td>
<td>Rock7LOCK</td>
<td>MO</td>
<td>Hello World</td>
<td>11</td>
</tr>
</tbody>
</table>

So what are all those numbers and letters? The next section is a brief run down. We'll go into more detail later.

**Unpacking Data**

For those familiar with Python's [struct](https://adafruit.it/BsY) module, the example data can be decoded with:

```
struct.unpack("&lt;6fB5f", data)
```

where data is a bytes or bytearray of the raw data in the message. An easy way to create that is to use `bytes.fromhex()` and give it the hex text (copy pasta it) from the message.

Using the above message as an example:

```python
import struct

data = bytes.fromhex("88984be9be9920fb21d41f3754c2081c3f410aed74188277a44b42a2ccbe7fdv"
(-0.1998061054229736, -0.24526429176330566, 9.856033325195312, -53.05464172363281, 1.344635963439944, -65.02481079101562, 0, 23.97783660888672, 26.959991455078125, 1000.61767578125, 105.8177954101562, 26.940818786621094)

struct.unpack("&lt;6fB5f", data)
```
And there's the data. First 3 are x/y/z acceleration from the LSM6DS33, next 3 are x/y/z magnetic from the LIS3MDL, next one is proximity from the APDS9960, then humidity and temperature from the SHT31D, and finally pressure, altitude, and temperature from the BMP280.

Much sensors. So data. Wow!

Raspberry Pi Example

A Raspberry Pi or similar Single Board Computer (SBC) can also be used, as long it has a serial port for talking to the RockBLOCK. If you want to use the CircuitPython library, you'll also want to use a SBC that has Blinka (https://adafruit.it/EQa) support.

You can use any Raspberry Pi. However, since the remote location will likely not have WiFi and power will probably be a premium, the Pi Zero is a good option.

Raspberry Pi Zero - Version 1.3
At first glance, the Pi Zero isn't much. It just looks like a slimmed down version of the Raspberry Pi we know and love. But when we started to think of the...
https://www.adafruit.com/product/2885

RockBLOCK 9603 Accessory Cable
The RockBLOCK 9603 Accessory Cable is terminated with a simple 10-pin Molex-style connector at one end and male jumper wires at the other. This accessory cable...
https://www.adafruit.com/product/4529
Wiring

Here's the wiring diagram. If you use the accessory cable, you should be able to match the colors. But it's also best to verify actual pin location on the RockBLOCK.

- 5V to VIN
- GND to GND
- TX to TXD*
- RX to RXD*

* Yep, TX to TX and RX to RX. The RockBLOCK uses backwards nomenclature for its TX/RX pins.

Enable Serial

Follow these steps to enable serial on the Pi's GPIO header. The general process is to enable the serial hardware AND remove the default login shell that would otherwise use it.

You can do all this via raspi-config.

```
sudo raspi-config
```

- Select 5 Interfacing Options
• Select P6 Serial

• Answer NO to "Would you like a login shell to be accessible over serial?"

• Answer YES to "Would you like the serial port hardware to be enabled?"
Install CircuitPython Library

See here for information on how to install Blinka to allow using CircuitPython libraries with Python on the Raspberry Pi:

[Install CircuitPython Libraries on Raspberry Pi](https://adafruit.it/Deo)

and make sure you have passed the [Blinka Test](https://adafruit.it/Deo).

Then, install the RockBLOCK library with:

```bash
pip3 install adafruit-circuitpython-rockblock
```

Run Simplestest Check

As a quick test to make sure you are connected and talking to the RockBLOCK, use the simplestest example from the library. Here's the code.

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
# pylint: disable=wrong-import-position
# CircuitPython / Blinka
import board
uart = board.UART()
uart.baudrate = 19200

# via USB cable
# import serial
# uart = serial.Serial("/dev/ttyUSB0", 19200)
```
from adafruit_rockblock import RockBlock

rb = RockBlock(uart)
print(rb.model)

We need to make some changes first.

Comment out these lines:

```python
uart = board.UART()
uart.baudrate = 19200
```

so they look like this:

```python
# uart = board.UART()
# uart.baudrate = 19200
```

and add these new lines:

```python
import serial
uart = serial.Serial("/dev/serial0", 19200)
```

and now your UART is talking on the Pi's GPIO header.

Try running the example:

![Example output](image)

and you should see the model information text printed out as shown above.

Additional Examples

The only change that should be needed for running other examples is to make the changes for configuring the UART. Once you've done that and passed that UART to the CircuitPython RockBLOCK library, the rest should just work.

Check out the other examples in the library repo.
Forwarding Messages

OK, we've covered how to use the RockBLOCK to send messages from anywhere on Earth. But how do you get those messages to your final destination? That is, how do you forward them from the Rock Sever server to your specific end application, like so:

Rock Seven only provides a very basic functionality here. Once a message is received on their server, it can be forwarded to a location you specify. There are only two options:

- Email - You'll receive an email at the supplied address(es).
- HTTP - An HTTP POST request will be made to the supplied URL(s)

Rock Seven has documentation on this here:

Integration with your application
https://adafru.it/JIF

The HTTP API is documented here:

Receiving Messages via HTTP
https://adafru.it/JJa
Delivery Groups

You configure message forwarding using the web based Management System. Once logged in, click Delivery Groups in the left hand navigation.

It's a fairly simple idea. Each Delivery Group is associated with one or more RockBLOCK devices. Then, for each group, you can specify one or more delivery destinations in the Delivery Addresses. It's a simple forwarding scheme. Any messages received from the RockBLOCK devices in the Delivery Group will be forwarded to the destinations in the Delivery Addresses.

Adafruit IO

There is a way to get your RockBLOCK message to Adafruit IO, however it has some limitations. Rock Seven doesn't allow for any special formatting of the outgoing HTTP POST message. So we can't interface it directly with the AIO API. However, we can use the webhook feature of AIO to provide an endpoint destination for the HTTP POST data.

Start by following this guide for how to create a webhook for your AIO feed:

Webhooks with Adafruit IO
https://adafruit.io/IwF

You will end up with a URL for the feed's webhook.

Now log in to your Rock Seven account and go to Delivery Groups. Add a new Delivery Address for your RockBLOCK.

1. Add the URL for the AIO feed's webhook into the Address field.
2. Select HTTP_JSON from the Format drop down.
Now when you transmit a message from your RockBLOCK it will be forwarded as JSON data to your feed. It will show up as something like this:

![Image](image.png)

It's not super useful. All you get is a raw blob of JSON text. You will have to do further processing. But that actual data is there in the aptly named data field.

Here's an example of how you could grab that JSON data, process it to get actual values in CPython (desktop, Raspberry Pi, etc), and then send those values back to the AIO feeds of interest. The data used is based on the CircuitPython example from this guide.

```python
import struct
from Adafruit_IO import Client
from secrets import secrets

aio = Client(secrets['aio_username'], secrets['aio_key'])

# get the raw data
raw_feed = aio.feeds('test-feed')
raw_data = aio.receive(raw_feed.key).value

# the JSON blob is truncated, so just parse manually
print("Getting raw data...")
data = bytes.fromhex(raw_data.split(',')[2].split(':')[1].strip('"'))
values = struct.unpack("<6fB5f", data)
print(values)

# send parsed data back to specific feeds
print("Sending to AIO...")
aio.append(aio.feeds('rock-block.rb-humidity').key, values[7])
aio.append(aio.feeds('rock-block.rb-temperature1').key, values[8])
aio.append(aio.feeds('rock-block.rb-pressure').key, values[9])
aio.append(aio.feeds('rock-block.rb-temperature2').key, values[11])
print("DONE.")
```

More Info on Data Transmission

TLDR

The RockBLOCK sends and receives raw data bytes. It's up to your applications, both on the send and receive end, to encode and decode those as you see fit.
Text vs. Binary

The RockBLOCK essentially just sends and receives 1s and 0s. It really does not care what those are or what they mean. It sends these in groups of 8, which is what a byte is. You can send up to 340 bytes in one satellite transmission.

When we ran the Hello World example, we were able to use a simple text style entry:

```
AT+SBDWT=Hello World
```

This is generally fine since there is a known single byte representation for each character. But what if you wanted to send a value? Like the temperature from a sensor which is reading 23.6245198. Should you do something like this?

```
AT+SBDWT=Temperature is 23.6245198
```

You could, and it would work, but there are a couple of issues with is.

For one, sending the text "Temperature is" is unnecessary. You will most likely know the format of whatever data you are dealing with. So then this?

```
AT+SBDWT=23.6245198
```

That would also work, but sending the value of text is costly in terms of the number of bytes used in the transmission. Each character will consume a byte, so 10 bytes total. It's much better to send the value as an actual byte representation. There are different size bytes representations that can be used for values, but for example, a typical floating point value can be stored in 4 bytes (for single precision). So you could send that same temperature value for less than half the number of bytes it would take to send it as text.

Great, but how do you actually do this? It will depend on what programming language you are using. We'll discuss this a little further using Python.

**For sending bytes, you would also use the AT+SBDWB command instead of the AT+SBDWT command.**
Packing Data in Python

The process of turning values into raw data bytes is generally referred to as packing. The reverse process, turning the raw data bytes back into values, is referred to as unpacking. The Python module `struct` provides what is needed.

Let's look at a simple example. Our goal is to send the value of 23.6245198 with the RockBLOCK modem, through the Iridium satellites, to some other place on Earth. We want to do this as efficiently as possible since we burn credits on a per byte basis.

First, let's create our `value`:

```
$ python3
Python 3.6.9 (default, Nov 7 2019, 10:44:02) 
[GCC 8.3.0] on linux 
Type "help", "copyright", "credits" or "license" for more information. 

value = 23.6245198
```

If we wanted to send that as `text`, we would do something like:

```
import struct

data = struct.pack("f", value)

len(data)  # 4
```

See here for meaning of the "f" formatting character. (https://adafru.it/BsY)

The contents are no longer human readable. But it only requires 4 bytes.

To prove that the value is still there, we can `unpack` it:

```
struct.unpack("f", data)  # (23.62451934814453,)
```
That's what someone would do on the receiving end, after the 4 bytes went through the satellite system.

The key thing to note here is that it would take 10 bytes to send the text representation of the value vs. 4 bytes for the raw byte representation.

### Integer vs Float

Note how in the previous example when we unpacked the value we got extra digits and it wasn't exactly the same value we started with. This is due to the general issue of floating point precision. There's no 100% cure for this. About the only thing that can be done is to throw more bytes at the issue. For example, using double precision, which takes 8 bytes, we get better results:

```python
value = 23.6245198
data = struct.pack("d", value)
len(data)
8
struct.unpack("d", data)
(23.6245198,)
```

Great, but we just doubled the amount of bytes that needs to be sent.

The better solution to this problem is to use integers for everything. But how do you turn floating point numbers into integers without losing data? You generally can't. But that's not necessary.

These values will most likely have started out as integers to begin with. They originated from registers on the sensors where they were stored as a series of 1s and 0s. The datasheet would have all the information about the layout which could then be used to compute the actual values of interest. Doing this work for you is one of the main features of using a sensor library. But, for the sake of efficient satellite data transfer, instead of reading the register, computing the value of the physical property of interest, and then trying to transmit that value - just send the raw register values as integers.

The integer value can store all the 1s and 0s of the registers as they actually are. It is efficient and no information is lost. And turning those values into the engineering units of interest on the receiving end is a trivial task.