


./ theba1dgeek.github.io

Rough notes on how to go about setting up the hardware and software for ACARS decoding.

 [View on GitHub](#)

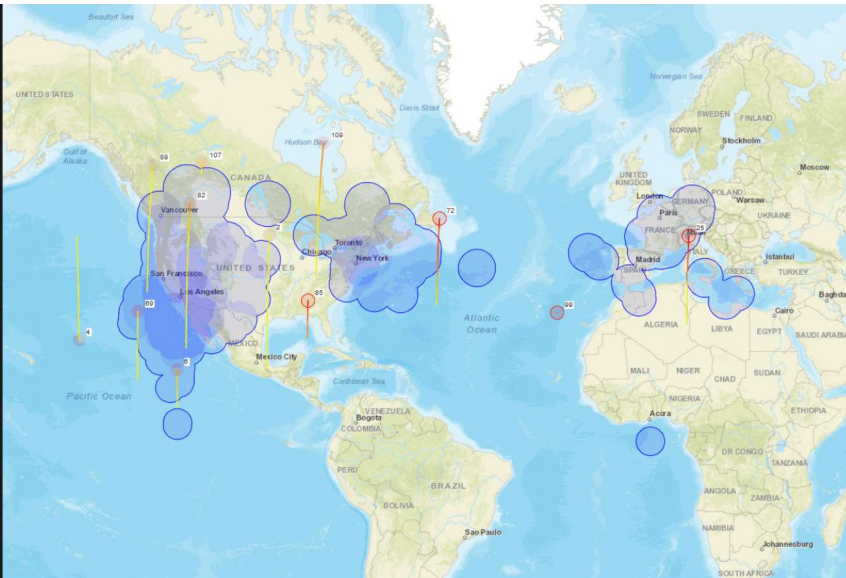
Iridium ACARS Decoding.

Navigation: [home](#)

There is a solid amount of ACARS messages being sent via the Iridium satellite constellation.

To set expectations right up front, there are no position reports, so no chance of plotting pixels on a map, also the way Iridium works, routing messages between satellites in LEO to the nearest satellite that can 'see' a ground station (gateways are located in Tempe, Arizona (USA) Fairbanks, Alaska (USA) Svalbard, Norway (Europe) Punta Arenas, Chile (South America)), you often only get a fragment of the ACARS message. And voice? Very delayed, very time consuming, very CPU intensive for just a few seconds of a random unknown call audio (ie, more than just aircraft voice calls are carried by Iridium, so you cant just decode aircraft calls), so we are not even going to discuss it.

Currently there are a solid number of avgeek ground stations scattered around mainland USA and a few around Europe / UK which when combined are seeing around 24,000 Iridium ACARS messages a day.



In Feb 2022 we started looking at it seriously, so all this is very new and thus a bit rough around the edges, but here are some tips to get you started. If you want to get technical, [this doc](#) is a good read.

Sep 2022 Iridium has picked up a LOT of interest in the past few weeks with some very interesting posts on the ACARS groups.io email list. Seems that a good amount of military aircraft (All USAF KC-135) are going to be moving from Inmarsat L-Band to Iridium.

Parts required to build an Iridium ground station

Antenna

I started out using the RTLSDR v2 patch antenna since its only meh at Inmarsat and so I had an unused one kicking around. While not the best antenna for Iridium (its directional and has a SAW filter in the LNA), its really not too bad given it's price and availability, if its all you can get, then give it a go by mounting it outside, looking straight up ie, flat, or angled in any direction that does not have obstructions (trees etc). Some people have hand built themselves a RHCP L-Band patch antenna, since it has less forward gain, the home made ones are an option if you are so inclined.

I jumped on eBay and picked up a combined GPS and iridium dome antenna and will report back on how it goes once we get some air time with it.

Do note that there are very few active (built in LNA) Iridium antennas since transmitting up to the satellites is very common.

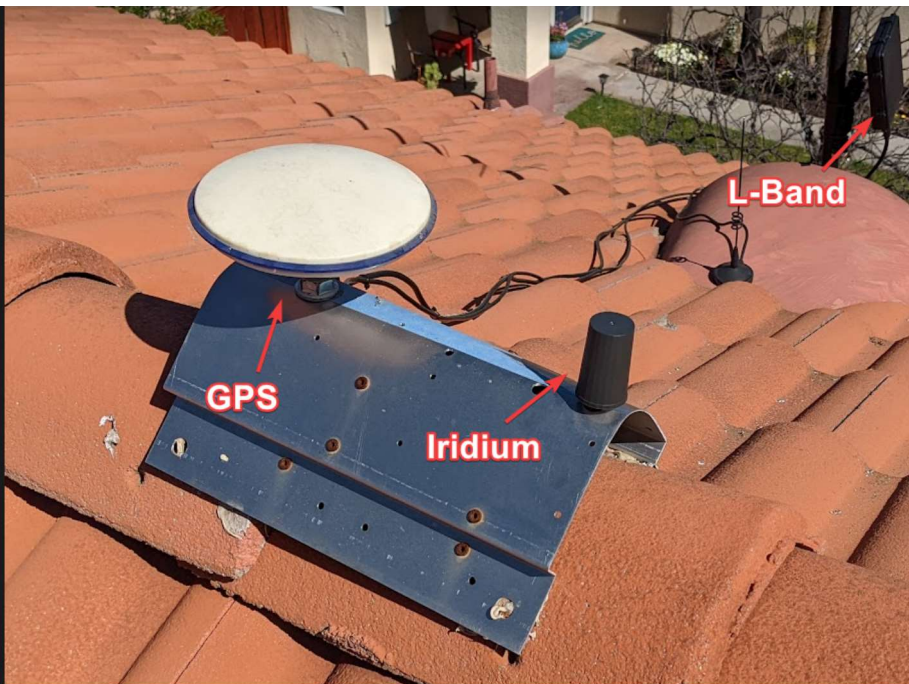
That said, I have found one that I really like: [HC610](#). (There is also a passive version of this antenna, so that might be an easier to find and buy than this active one). Note that the HC610 has some nice out of band filters to help cut down on the very busy L-Band crud we are not wanting to hear.



This is the view of the bottom of the antenna. Note the male connector.



This helix is very small, should be able to mount it outside in the clear view of the 360 deg sky very easily.



Mounted next to my GPS (galmon) antenna. Note that the L-Band patch is looking at the 54w Inmarsat, not Iridium.

If you are using an active antenna or LNA be sure and enable the Bias-T on your SDR via its config file, or use a physical Bias-T power injector (which is what I do because I am testing a lot of different SDRs and they all are a pain to turn the bias-T on in the config file - some don't even support Bias-T - some (RSP1a) only supply 10's of milliamps via the Bias-T and depending on your LNA you might have issues supplying enough power.).

Bottom line, L-band RHCP omni directorial antenna with a clear view of the sky, good LNA and good quality coax is required.

LNA

There are a few wide band amplifiers that cover 1.6Ghz, but the Nooelec Iridium+IR LNA has amazing performance. Well worth the money and Bias-T hassels to drive this amplifier.

At a quick glance, you may assume that there are a few Nooelects that seem to cover this band, they don't all cover the full range of Iridium frequencies. Nooelec have a really helpful [page on their SAWbird LNA range](#), take some time to read it before making your purchase. Here is a tip, Iridium is really loud. You may not need the extra gain of the + LNA. Also note that the + draws an extra 150mA. So for example, if you are using an RSP1a SDR, its Bias-T can only supply 50mA, so the base SAWbird LNA is (should be - might be) fine at 30mA draw, the + at 180mA will not work. In the case of Iridium, the extra 10db gain is probably not needed so this combo would work fine depending on your antenna, coax quality and length. (Of course, once again, to really press this home, you

can just use a physical Bias-T and not have to worry about any of this).

SDR

I normally like the RTLSDR v3 for this sort of thing (all things ACARS and L-Band). Its very affordable and very quick and clean to get running. The problem with the RTLSDR is that it only covers around 2Mhz bandwidth and that is only a very small number of the Iridium data channels.

I am testing the LimeSDR Mini v1 (10mhzBW), RSP1a (9mhzBW) Airspy R2 (10mhzBW), Airspy Mini (6mhzBW) and HackRF (10MhzBW) and am getting good numbers from most of these, easily more than 4x the data from the RTLSDR due to all of them covering more bandwidth of various amounts.

The RSP1a would be a viable option if not for the rather unstable API Linux driver they ship. It is very CPU intense (wasteful) and as I said, somewhat crash prone. (This is not just an gr-iridium issue, those using the RSP1a on dumphfdl have the exact same issue). The hardware is great, the Linux software is lacking. There currently no Windows Iridium decoders that I know of. (No idea about Mac).

If you follow my Twitter (sorry), you might have seen a month long thread about if the Airspy R2 really can cover the full 10Mhz bandwidth required by Iridium and October 24th 2022, we have an answer. Yes!

In the airspy.conf file simply change the center frequency, set the sample rate to 10 and the bandwidth to 10 and connect the R2 to a USB 3.0 port and you will get all the Iridium data there is to get (assuming your CPU can keep up).

To be clear. We don't actually know for sure if you require a 10Mhz bandwidth SDR or not. It seems that there is no ACARS in the top bit and so an 8Mhz might be Ok. But, we do KNOW for sure that you need a computer to drive it to get all the data channels on Iridium. (Do note that the Raspberry Pi 4 is just not powerful enough for Iridium). Most bleeding edge stations are running 10meg to see whats out there in this new mode.

Software. MUCCC - iridium-toolkit and gr-iridium

If you want to build from source the repo can be found on the [Chaos Computer Club München](#) Github. I really don't recommend building from source, gnu-radio is not trivial to build and nor is the gr-iridium toolkit. Ubuntu 22 is getting a bit better with

gnuradio being in the repo, but there are still some hopes to jump through to get all the pieces and parts working together from what I am told - I simply use DragonOS_FocalX where its all plug and play.

Note that none of the Iridium tools use a gui, so you must run it all via the desktop shell terminal on the DragonOS computer (or via PuTTY with DragonOS). I did my testing on a VMware instance on my Windows PC since you need USB 3.0 and plenty of CPU power to drive your SDR to the required 10MHz BW and decode all the burst data.

Computer - CPU power is critical

The decoding of such a wide bandwidth of such bursty data is very CPU intense. You will need a USB 3.0 port to pass the data smoothly (USB 2.0 is a bit of a bottle neck).

For starters, the Raspberry Pi4 does NOT have enough power to decode more than about 2mhz, ie, the output of the RTLSDR v3. So yes, you can get started and see a few messages, but you will be missing out on over 80% of the aircraft in your area and you will have to choose between getting ACARS and seeing your map coverage, one or the other, the Pi4 and RTLSDR v3 can not decode both. (Look down this page a bit to see why, the map data is at one end and ACARS data at the other end of the 10MHz spread).

Our KBOS station is running: i3-12100 CPU, H670 chipset. That seems to be doing the job very well. A few other stations are running the Beelink NUC clones and seem to be very happy reporting good decode rates.

A few other stations are using old laptops, i5, i7 and are working well. One reported his very old i7 could not keep up and he had to upgrade the system he ran the Iridium code on.

RAM is not critical, around 2mb or more is enough. HDD space requires a minimum of 32GB for DragonOS_Focal.

Personally I am running DragonOS_Focal on a VMware player machine on my Windows PC and am very happy.

You will know if your CPU is up to the task based on how many dropped frames you see. Roughly the make/break is to have a 'ok_ave' of more than 70%.

Getting Started

Once you install DragonOS_FocalX on an i5 or better x86 computer with a USB 3.0 port, you are ready to start.

For now you are going to open a few terminals, we are working on an script to run it and keep it running (it does crash now and then), but for now, this is the best way to get going....

This guide assumes that you would like to share your Iridium data with the world and thus send it to thebaldgeek so it can be added to the main Iridium page and more. You can also open another terminal and use the same process to send data to yourself on another UDP port number in another terminal.

Here is the big picture, we are going to make a python file (acars.py) that will take the output from the Iridium decoder and send it via UDP to my ingest server. You will have one terminal to parse the raw data and another terminal to decode the ACARS messages and to send the data to my site. (Optionally, there is a third terminal for your local map if you would like to see your coverage).

Lots of terminals

Terminal One

Run the extractor from any directory: (This assumes you are testing with an RTL-SDR V3, change the .conf file to match your SDR and edit it to turn on the Bias-T if required).

```
iridium-extractor -D 4 --multi-frame /usr/src/gr-iridium/examples/rtl-sdr.conf | python3 -u /usr/src/iridium-toolkit/iridium-parser.py -o zmq
```

To be clear, if you not using an RTLSDR look in the /usr/src/gr-iridium/examples/ directory and find your SDR and tweak that file to best set it up.

You are going to get a line of data per second:

```
1666645162 | i: 351/s | i_avg: 392/s | q_max: 24 | i_ok: 79% | o: 751/s |  
ok: 82% | ok: 291/s | ok_avg: 85% | ok: 31170861 | ok_avg: 334/s | d:  
8260
```

See the MUCCC GitHub page linked above for a full breakdown on each of these values and their meaning.

You want to see 60% to 100% in the `ok_ave:` part. Lower number means more bad packets and you need to fix your antenna, coax, LNA or gain in the .conf file.

You will spend a fair bit of time looking at these numbers scroll past as you set up and tune your station. The key value to tweak is the gain. Try both higher and lower, but max may not be the

best out the gate. Test each gain change over at least 1 hour to give time for few satellites to pass over your station location.

Terminal Two

Type `nano acars.py`, then copy/paste in this text:

```
#!/usr/bin/env python3

import sys
import select
import time
import socket

ap = ("thebaldgeek.net", 123456)
sk = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
def sendOverUdp(line):
    try:
        bytes = str.encode(line)
        print(len(bytes))
        if len(bytes) < 65300 :
            sk.sendto(bytes, ap)
    except Exception as e:
        print(e)

def no_input():
    print('no input')

while True:
    line = sys.stdin.readline()
    if line:
        sendOverUdp(line)
    else:
        time.sleep(1)
else:
    no_input()
```

Change the `thebaldgeek.net` to what ever host you want to send your UDP ACARS messages to, and also change the port number from 123456 to the port you are using or thebaldgeek gives you. Then save and exit nano.

Next run this command:

```
python3 -u /usr/src/iridium-toolkit/reassembler.py -m acars zmq: |  
python3 /home/ubuntu/acars.py
```

 (Swap out 'ubuntu' for your username).
Do note that nothing will show in this terminal until you pickup your first ACARS message. Depending on how much Iridium aircraft there are in your area, it could take a moment or a few minutes, then a single number will show up, you will see a number for every message. The number is the size of the ACARS message once its decoded.

Terminal Two point Five

if you want to see your ACARS messages stream past, then open another terminal and just connect without the pipe to the acars.py, like this....

```
python3 -u /usr/src/iridium-toolkit/reassembler.py -m acars zmq:
```

Terminal Three - Your local map.

Now, we need to get the map running: (This is optional, but cool to see)

```
cd /usr/src/iridium-toolkit/html
```

```
sudo nano example.sh
```

On the second bottom line, if not already done, add a 3 at the end of the word python and be sure and change the IP address for your PC (your PC might not be 192.168.1.122), so it should read `python3 -m http.server --bind 192.168.1.22 8888`

Save and exit nano

In the terminal run the example file: `sudo ./example.sh`

At this point, you can visit your PC's IP address from any browser on your network and look for the map, so in my case

```
http://192.168.1.122:8888/map.html
```

Let that run. You should see the sats and beams update around once a minute.

The Fun Part

Now that you have a UDP stream of ACARS messages you can perhaps use Node-RED to view them, or save them to your hard drive to view with your text filter software / application.

Very soon airframes.io will start ingesting Iridium data and when that happens you will be part of a global system using this very challenging mode.

Speaking of 'fun', it seems that once again and aircraft builders are tormenting AVGEEKS the world over as they use different ICAO/Rego/ModeS combos in their messages to identify their aircraft.

thebaldgeek is keeping a list of unknown aircraft and if you like tracking down those sorts of things, everyone would love your help to figure out the ones in the table. Once they are 'known', their Iridium 'code' is put in a CSV file and used site wide for identifying the aircraft no matter what mode they show up on. At the moment, ADSBEX does not include these...aaahhh... 'special' codes, so acars.adsbexchange is the only place these Iridium aircraft are decoded and put into tables / database.

thebaldgeek is sharing these codes with the airframes.io guys as he does not want to be a data island and wants to share all the information he can so more people can benefit.

Jump over the fold to see the global map command.....

Simplex Frequency Allocation

Channel Number	Center Frequency(MHz)	Allocation
1	1626.020833	Guard Channel
2	1626.062500	Guard Channel
3	1626.104167	Quaternary Messaging
4	1626.145833	Tertiary Messaging
5	1626.187500	Guard Channel
6	1626.229167	Guard Channel
7	1626.270833	Ring Alert
8	1626.312500	Guard Channel
9	1626.354167	Guard Channel
10	1626.395833	Secondary Messaging
11	1626.437500	Primary Messaging
12	1626.479167	Guard Channel

Duplex Channel Band

Sub-band	Lower Edge (MHz)	Upper Edge (MHz)
1	1616.000000	1616.333333
2	1616.333333	1616.666667
3	1616.666667	1617.000000
4	1617.000000	1617.333333
5	1617.333333	1617.666667
6	1617.666667	1618.000000
7	1618.000000	1618.333333
8	1618.333333	1618.666667
9	1618.666667	1619.000000
10	1619.000000	1619.333333
11	1619.333333	1619.666667
12	1619.666667	1620.000000
13	1620.000000	1620.333333
14	1620.333333	1620.666667
15	1620.666667	1621.000000
16	1621.000000	1621.333333
17	1621.333333	1621.666667
18	1621.666667	1622.000000
19	1622.000000	1622.333333
20	1622.333333	1622.666667
21	1622.666667	1623.000000
22	1623.000000	1623.333333
23	1623.333333	1623.666667
24	1623.666667	1624.000000
25	1624.000000	1624.333333
26	1624.333333	1624.666667
27	1624.666667	1625.000000

28	1625.000000	1625.333333
29	1625.333333	1625.666667
30	1625.666667	1626.000000

Stop reading here, the notes below are mostly wrong and are just for history.

Note that I used to run a global Iridium coverage map, but the URL was getting 'attacked' to try and break into my network, so I took it down. If there is enough interest from the handful of Iridium feeders, I can put it back up and just let a few people know about it.

Sep 2022. Its back!

Terminal Four - Sending me your map data.

Almost there: DM on Twitter / Discord or email me and ask for the map2.py script that will allow you to send your local map data to the global map. (URL only for Iridium feeder sharers).

Next run this command: `pip install https://github.com/joh/when-changed/archive/master.zip`

This will install a python script that will look for changes to a file. Now go to where it was installed:

```
cd /home/ubuntu/.local/bin
```

Now run the file watch which will send me your sats.json roughly once a minute and your coverage will be added to the master map on my site:

```
./when-changed /usr/src/iridium-toolkit/html/sats.json cat  
/usr/src/iridium-toolkit/html/sats.json | python3 ~/map2.py
```

Really really stop reading now.

Getting the latest version

```
cd ~ wget https://github.com/muccc/iridium-  
toolkit/archive/refs/heads/master.zip unzip master.zip
```

Now path to where that unzipped ie, /home/yourusername/iridium-toolkit-master/

Pipe and tee

We are going to pipe the data into Node-RED via UDP. To do this, we need a way to get the stdout data into a UDP stream, we will [use the iridiumlive python code](#) to do that. Pull the code down from the github and save it in a file (or just cut paste it into a nano editor). Change the IP address in the code to match your Node-RED computer and pick a different port, but note the number. I strongly suggest putting the port number in the file name you save as you may end up using a few of these files to move data around (I have three). My files are `iudp55667.py` and `iudp66778.py`. Obviously, the port number is the same in the python code as the file name and the same number is used in the Node-RED UDP in node. You will need to chmod the files to make them executable. We can dump everything at once into Node-RED, but its a firehose, so lets just get ACARS going for a start.

```
iridium-extractor -D 4 --multi-frame rtl-sdr.conf | python3 -u ./iridium
```

You can see we are piping the data from each application to the next. The `-u` tells python to unbuffer the data. Note you may or may not have to call out 'python3' depending on what versions of python you have installed on your Pi, I have both v2 and v3, so need to call out which to use (you must use v3 for the iridium-toolkit).

Now on the Node-RED end, you can put your UDP in node and a debug node and after a few minutes you should see your first Iridium ACARS message pop up.

If you want to see short burst data messages, in the command above, swap `-m acars` for `-m sbd`. You will see more sbd messages than ACARS, so that might be a good sanity check.

What if you want both on different streams? Use a `tee`.

```
iridium-extractor -D 4 --multi-frame rtl-sdr.conf | python3 -u ./iridium
```

Now add a second UDP in node and a second debug in the Node-RED editor and you will have ACARS on port 55667 and sbd on port 66778.

How about three feeds. One to IridiumLive, one to ACARS and one to sbd...


```
sudo iridium-extractor -D 4 --multi-frame ~/iridium-toolkit-master/rtl-s
```

If you get sick of seeing the status message scrolling up the page in the terminal, you can send it to null like this:

```
iridium-extractor -D 4 --multi-frame rtl-sdr.conf 2>/dev/null | python3
```

Now you just get the UDP count of real messages.