Welcome to SatDump's documentation!

What is SatDump?

SatDump is a general purpose satellite data processing software. It is a one-stop-shop that provides all the necessary stages to get from the satellite transmission to actual products.

Features

- Support of many SDRs such as RTL-SDR, Airspy, HackRF, BladeRF, LimeSDR, PlutoSDR, etc.
- Recording of radio basebands from your SDR
- Decoding and processing the data from over 90 different satellites and even space probes.
- Live decoding of supported satellite links such as APT, LRPT, HRPT, LRIT, HRIT and many more.
- Image and data decoding from satellites such as NOAA 15-18-19, Meteor-M, GOES, Elektro-L, Metop, FengYun, etc.
- Calibrated and georefrenced L1b products output on select satellites, such as Sea Surface Temperature, Microphysics, etc. ready to use for scientific applications such as numerical weather forecasts.
- Support for projecting the satellite imagery over a map, including layering with other instruments or satellites.
- Inmarsat Aero and STD-C EGC messages decoding.
- Scheduler and rotator control for automated satellite stations.
- Ingestor for automated geostationary weather satellites reception.

Documentation

- Elektro/Arktika (PLUGIN_ELEKTRO_ARKTIKA)
- EOS (PLUGIN_EOS)
- FengYun-2 (PLUGIN_FY2)
- FengYun-3 (PLUGIN_FY3)
- GK-2A (PLUGIN_GK2A)
- GOES (PLUGIN_GOES)
- Himawari (PLUGIN_HIMAWARI)
- Jason-3 (PLUGIN_JASON3)
- JPSS (PLUGIN_JPSS)
- Meteor (PLUGIN_METEOR)
- NOAA/MetOp (PLUGIN_NOAA_METOP)
- OceanSat (PLUGIN_OCEANSAT)
- Others (PLUGIN_OTHERS)
- Proba (PLUGIN_PROBA)
- SpaceX (PLUGIN_SPACEX)
- Scripting (PLUGIN_SCRIPTING)

- SDR Options
  - Airspy
  - AirspyHF
  - HackRF
  - BladeRF
  - LimeSDR
  - RTL-SDR
  - RTL-TCP
  - Aaronia Spectran V6
  - SDDC (RX888, RX999, etc)
  - SDRPlay
  - SpyServer
  - SDR++ Server
  - PlutoSDR
  - USRP
  - MiriSDR
  - File

- Satellite Tracker
  - Tracker Interface
    - Polar Plot
    - Satellite Selector
      - Satellites
      - Horizons

SatDump Links

- Website
- Download

Satellite pipelines

The aim of this is to list all possible satellite processing pipelines as well as their parameters. This does not explain the pipelines by any means. This should just simplify the CLIs use.

NOAA

- `noaa_hrpt`: NOAA HRPT
- `noaa_gac`: NOAA GAC
  - `backward`: Over the US, NOAA satellites often transmit in reverse order. This mode allows decoding those backward replays properly.
- `noaa_dsb`: NOAA DSB
- `noaa_apt`: NOAA APT
  - `satellite_number`: For apt it is required to know what satellite is being received for projections and overlays. Options are `15, 18, 19`
  - `start_timestamp`: Required for projections and overlays. If your .wav file is a supported format it will be read automatically. Unix timestamp of the start of the file. Must be UTC Unix timestamp in seconds.
  - `align_timestamps`: Aligns pass to the correct time using timing marks in the APT space data. Start time must still be within 15 seconds for this to work! Default: true
  - `sdrpp_noise_reduction`: Uses the APT noise reduction originally implemented in SDR++. Default: True
  - `save_unsynced`: Saves the image before it's syncronized. Useful for weak signals. Default: True
  - `autocrop_wedges`: This will automatically crop the image to only include telemetry wedges considered valid. May discard a lot on bad images! Default: False
  - `max_crop_stddev`: If autocropping, the amount of noise to allow at the top/bottom of the image. Lower values = more cropping. Default: 3500. Useful range is about 400 - 10,000
  - `save_wav`: Saves the wav file. Default: False

Meteor M

- `meteor_hrpt`: METEOR HRPT
  - `year_override`: Override of the year used for timestamps. Required when you are processing your file in a different year than the data was created.
- `meteor_m2_lrpt`: METEOR M2 LRPT 72k
  - `satellite_number`: If specified, overrides automatic satellite dection, and treats the

data as the specified satellite. Valid options: “M2”, “M2-2”, “M2-3”, or “M2-4”.
    - `fill_missing`: Fills in black lines caused by signal drop-outs or interference
    - `max_fill_lines`: Maximum contiguous lines to correct. Default is 50
- `meteor_m2-x_lrpt`: METEOR M2-X LRPT 72k: Same parameters as meteor_m2_lrpt, plus
    - `rs_usecheck`: Discards data that fails the Reed-Solomon check. True by default, and it is recommended you leave it on - but you can set it to “false” if the satellite is transmitting bad RS codes, which may happen from time to time.
- `meteor_m2-x_lrpt_80k`: METEOR M2-x LRPT 80k (Same parameters as meteor_m2_lrpt)
- `meteor_m_dump_narrow`: METEOR-M Narrow Dump (WIP!)
- `meteor_m_dump_wide`: METEOR-M Wide Dump (WIP!)

MetOp

- `metop_ahrpt`: MetOp AHRPT
    - `write_hpt`: Generate a .hpt file to load into HRPT Reader for processing.
- `metop_dump`: MetOp X-Band dump

AIM

- `aim_dump`: AIM dump

AWS

- `aws_pfm`: AWS PFM L-Band (DB and Dump)
    - `use_ephemeris`: Use ephemeris data from the satellite for projection

BlueWalker-3

- `bluewalker3_wide`: BlueWalker-3 S-Band Wide
- `bluewalker3_narrow`: BlueWalker-3 S-Band Narrow

Cloudsat

- `cloudsat_link`: Cloudsat S-Band link

Coriolis

- `coriolis_db`: Coriolis S-Band Tactical DB

Cosmos

- `cosmos_2558_dump`: Cosmos 2558 S-Band dump

Cryosat

- `cryosat_dump`: Cryosat X-Band dump

DMSP

- `dmsp_rtd`: DMSP RTD
  - `satellite_number`: DMSP does not transmit a satellite ID. As instrument configurations do vary between them, it is required to identify the satellite. Options are `17, 18`.

EOS

- `aqua_db`: Aqua DB
- `terra_db`: Terra DB
- `aura_db`: Aura DB

Elektro / Arktika

- `elektro_rdas`: ELEKTRO-L RDAS
- `arktika_rdas`: ARKTIKA-M RDAS
- `elektro_lrit`: ELEKTRO-L LRIT
- `elektro_hrit`: ELEKTRO-L HRIT
- `elektro_ggak`: ELEKTRO-L GGAK
- `arktika_ggak`: ARKTIKA-M GGAK

Fengyun-2

- `fengyun_svissr`: Fengyun-2 S-VISSR

Fengyun-3

- `fengyun3_ab_hrpt`: FengYun-3 A/B AHRPT
  - `write_c10`: Generate a .C10 file to load into HRPT Reader for processing.
- `fengyun3_c_hrpt`: Fengyun_3 C HRPT
  - `write_c10`: Generate a .C10 file to load into HRPT Reader for processing.
- `fengyun3_abc_mpt`: FengYun-3 A/B/C MPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!

- `fengyun3_d_ahrpt`: Fengyun-3 D AHRPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_e_ahrpt`: Fengyun-3 E AHRPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_g_ahrpt`: Fengyun-3 G AHRPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_abc_dpt`: Fengyun-3 A/B/C DPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_d_dpt`: Fengyun-3 D DPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_e_dpt`: Fengyun-3 E DPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_f_ahrpt`: FengYun-3 F AHRPT
  - `dump_mersi`: Dump raw MERSI frames for processing with other software, such as Fred's WeatherSat!
- `fengyun3_tlm_old`: FengYun-3 TLM (Old) A/B/C/D
- `fengyun3_tlm`: FengYun-3 TLM E/F
-

Fengyun-4

- `fengyun4_lrit`: Fengyun-4 LRIT
  - `ts_input`: Input TS instead of BBFrame
- `fengyun4_hrit23`: Fengyun-4 HRIT-II/III
  - `ts_input`: Input TS instead of BBFrame

GCOM

- `gcom_w1_link`: GCOM-W1 link
- `gcom_c1_link`: GCOM-C1 link

GEO-KOMPSAT_2A (GK-2A)

- `gk2a_lrit`: GK-2A LRIT
- `gk2a_lrit_tcp`: GK-2A LRIT to xrit-rx
- `gk2a_hrit`: GK-2A HRIT

- `gk2a_cdas`: GK-2A CDAS

GOES

- `goes_gvar`: GOES GVAR
- `goes_hrit`: GOES-R HRIT
  - `write_images`: Saves FD, Meso, etc images (Default: True)
  - `write_emwin`: Save EMWIN Data (Default: True)
  - `write_messages`: Save Admin Messages (Default: True)
  - `write_unknown`: Save Unknown LRIT data(Default: True)
  - `parse_dcs`: Parse DCS files and save as json (Default: False)
  - `tracked_addresses`: When parsing DCS, only save info from the provided comma-separated addresses (Default: Save all DCS Messages)
  - `write_dcs`: Save DCS LRIT files (Default: False)
  - `write_lrit`: Write all LRIT files (Default: False)
  - `fill_missing`: Corrects black lines caused by interference or signal drop-outs (Default: False)
  - `max_fill_lines`: If `fill_missing` is enabled, sets the maximum number of lines that can be filled in (Default: 50)
- `goes_hrit_tcp`: GOES-R HRIT to goestools
- `goes_grb`: GOES-R GRB
- `goesr_cda`: GOES_R CDA
- `goes_md1`: GOES-N MDL
- `goes_lrit`: GOES-N LRIT
- `goesn_cda`: GOES-N CDA
- `goesn_sounder`: GOES-N Sounder SD
- `goesn_sd`: GOES-N Sounder Data
- `goesr_raw`: GOES-R Raw Data

GeoNetCast

- `geonetcast`: GeoNetCast
  - `ts_input`: Input TS instead of BBFrame

Himawari

- `himawaricast`: HimawariCast
  - `ts_input`: Input TS instead of BBFrame

Inmarsat

- `inmarsat_std_c`: Inmarsat STD-C
- `inmarsat_aero_6`: Inmarsat Aero 0.6k (WIP)
- `inmarsat_aero_12`: Inmarsat Aero 1.2k (WIP)
- `inmarsat_aero_84`: Inmarsat Aero 8.4k
- `inmarsat_aero_105`: Inmarsat Aero 10.5k (WIP)

JPSS

- `npp_hrd`: Suomi NPP / JPSS-1 HRD
- `jpss_hrd`: JPSS-2/3/4 HRD
- `jpss_tlm`: JPSS-2/3/4 Telemetry

Jason-3

- `jason3_link`: Jason-3 S-Band link

Lucky-7

- `lucky7_link`: Lucky-7 UHF link

MATS

- `mats_dump`: MATS dump

Oceansat

- `oceansat2_db`: OceanSat-2 DB
- `oceansat3_argos`: Oceansat-3 L-Band

Orbcomm

- `orbcomm_stx`: Orbcomm STX

Proba

- `proba1_dump`: Proba-1 dump
- `proba2_dump`: Proba-2 dump
- `probav_s_dump`: Proba-V S-Band dump
- `probav_x_dump`: Proba-V X-Band dump

SpaceX

- `falcon9_tlm`: Falcon 9 S-Band TLM
- `starship_tlm`: Starship S-Band TLM
- `crew_dragon_tlm`: Crew Dragon S-Band TLM

Stereo

- `stereo_lr`: Stereo-A/B Low Rate
- `stereo_hr`: Stereo-A/B High Rate

TGO

- `tgo_link`: Mars TGO X-Band Link

TUBSAT

- `tubin_x_dump`: TUBIN X-Band Dump
  - `check_crc`: Checks frames for errors. This is usually desireable, but sometimes ignoring errors may decode a bit more!

UVSQ

- `inspiresat7_tlm`: INSPIRE-Sat7 TLM

UmKA

- `umka_1_dump`: UmKA-1 dump

Others

- `saral_l_band`: Salral L-Band
- `angels_l_band`: Angels L-Band
- `gazelle_l_band`: OTB-3/Gazelle L-Band
- `yunhai_ahrpt`: Yunhai AHRPT - Encrypted ;(
- `syracuse3b_tlm`: Syracuse 3B TLM
- `scisat1_dump`: SciSat-1 dump
- `CALIPSO`: Calipso S-Band dump
- `youthsat_dump`: YouthSat dump

Chandrayaan

- `chandrayaan3_link_1k`: Chandrayaan-3 1k Link

- `chandrayaan3_link_2k`: Chandrayaan-3 2k Link
- `chandrayaan3_link_4k`: Chandrayaan-3 4k Link
- `chandrayaan3_link_8k`: Chandrayaan-3 8k Link

DISCOVR

- `dscovr_tlm`: DSCOVR TLM Link
- `dscovr_hr`: DSCOVR High-Rate Link

Hinode

- `hinode_s_dump`: Hinode S-Band Dump
- `hinode_s_tlm`: Hinode S-Band TLM

Iris

- `iris_s_dump`: IRIS S-Band Dump
- `iris_dump`: IRIS X-Band Dump

KPLO

- `kplo_sband_link`: KPLO (Danuri) S-Band Link

Landsat

- `landsat_ldcm_tlm`: LandSat 8/9 S-band
- `landsat_ldcm_link`: LandSat 8/9 X-band

Orion

- `orion_link`: Orion S-Band

Sentinel-6

- `sentinel6_dump`: Sentinel-6 Dump
- `sentinel6_tlm`: Sentinel 6 S-Band TLM

Tianwen

- `tianwen1_link`: Tianwen-1 Link

ViaSat

- `viasat3_tlm`: ViaSat-3 TLM

MSG

- `msg_raw`: MSG Raw Data

TODO: add Test, WIP

Plugin List

Since SatDump 1.0.0, most if not all code that is not common between many satellites or features is not in the main library anymore, but instead in plugins. Those plugins are entirely optional, and there is no requirement anymore to build the entire codebase.

By default `-DPLUGINS_ALL=ON` is set in CMake, but setting it to `OFF` will allow selecting what you wish to build. The syntax to enable a specific plugin is `-DPLUGIN_NAME=ON`.

The list provided belows aims at being a “quick-look” of what every plugin covers to help unfamiliar users decide on what they actually need.

The string in paranthesis after each plugin is the argument to be passed to CMake.

Satellite support

Elektro/Arktika (PLUGIN_ELEKTRO_ARKTIKA)

This plugin cover the ELEKTRO and ARKTIKA Russian weather satellites, both being very similar.

Supported downlinks : - X-Band RDAS from both - ELEKTRO-L LRIT - ELEKTRO-L HRIT - Low-Rate radiation payload downlink

EOS (PLUGIN_EOS)

This covers NASA’s Earth Observation System. - Aqua (Direct Broadcast) - Terra (Direct Broadcast) - Aura (Direct Broadcast)

FengYun-2 (PLUGIN_FY2)

Covers the first generation of FengYun GEO sats. Only S-VISSR is supported.

FengYun-3 (PLUGIN_FY3)

Supports all FengYun-3 satellites : - FengYun-A/B/C AHRPT on L-Band - FengYun-A/B/C MPT on X-Band - FengYun-A/B X-Band dumps (DPT) - FengYun-D/E X-Band AHRPT

GK-2A (PLUGIN_GK2A)

Supported downlinks : - GK-2A LRIT - GK-2A HRIT

GOES (PLUGIN_GOES)

This covers the NOAA GOES weather satellites : - GOES-N GVAR - GOES-R HRIT - GOES-R CDA - GOES-R GRB

Himawari (PLUGIN_HIMAWARI)

This provides support for the HimawariCast broadcast.

Jason-3 (PLUGIN_JASON3)

Supports the Jason-3 Altimetry mission on S-Band.

JPSS (PLUGIN_JPSS)

Provides support for the JPSS satellite mission : - Suomi NPP and JPSS-1 HRD - JPSS-2/3/4 HRD

Meteor (PLUGIN_METEOR)

Covers the russian METEOR missions : - L-Band AHRPT - LRPT on VHF

NOAA/MetOp (PLUGIN_NOAA_METOP)

This covers the POES missions, including the first generation MetOps and NOAA POES, carrying very similar instruments.

Support downlinks : - NOAA DSB - NOAA HRPT - NOAA GAC - MetOp AHRPT - MetOp X-Band dumps

OceanSat (PLUGIN_OCEANSAT)

Provides support for the OceanSat Indian satellite missions. - OceanSat-2 Direct Broadcast

Others (PLUGIN_OTHERS)

This plugin acts as a temporary place before some satellites are moved into their own plugi Currently, this includes : - Angels - CloudSat - Coriolis - Saral

Proba (PLUGIN_PROBA)

Covers the ESA Project for OnBoard Autonomy series of satellites : - Proba-1 - Proba-2 - Proba-V

SpaceX (PLUGIN_SPACEX)

This contains old support for the SpaceX Falcon-9 and StarShip telemetry downlinks. This is currently not updated as SpaceX has made the decision to encrypt the telemetry since then, but the code is kept to allow processing older data and in the eventuallity encryption is turned off in the future.

Scripting (PLUGIN_SCRIPTING)

This allows Lua Scripting, for example to archive data when the processing is finished.

SDR Options

This aims at covering all available SDR Settings for supported devices / sources. The goal is not to explain what every option does (for that, see the manufacturer's documentation for the specific device), but rather allow users to find the right flags in CLI mode.

Where possible, consistency was kept to be rather easy to "guess" if you know what you are looking for.

Airspy

- `gain_type` : The gain mode used by the device :
  - 0 is for Sensitive
  - 1 is for Linear
  - 2 is for Manual
- `general_gain` : Overall gain for sensitive and linear mode. Ranges from 0 to 21, in dBs
- `lna_gain, mixer_gain, vga_gain` : Used for manual gain tuning only
- `bias` : Enable Bias-Tee power
- `lna_agc` : Enable the LNA AGC
- `mixer_agc` : Enable the Mixer AGC

AirspyHF

- `agc_mode` : 0 for disabled, otherwise :
  - 1 for LOW
  - 2 for HIGH
- `attenuation` : Attenuation in dBs
- `hf_lna` : Enable or disable the HF LNA

HackRF

- `amp` : Enable the main (non-programmable) amplifier
- `lna_gain` : LNA Gain in dBs
- `vga_gain` : VGA Gain in dBs
- `bias` : Enable Bias-Tee power
- `manual_bw` : Enable manual bandwidth filter
- `manual_bw_value` : The bandwidth filter width in Hz

BladeRF

- `gain_mode` :
  - 0 is device default
  - 1 is manual
  - 2 is fast AGC
  - 3 is slow AGC
  - 4 is hybrid AGC
- `gain` : General Gain in dBs
- `bias` : Bias-Tee power (BladeRF 2.0 only)
- `manual_bw` : Enable manual bandwidth filter
- `manual_bw_value` : The bandwidth filter width in Hz

LimeSDR

- `gain` : Gain in dBs
- `manual_bw` : Enable manual bandwidth filter
- `manual_bw_value` : The bandwidth filter width in Hz

RTL-SDR

- `gain` : Device Gain in dBs
- `agc` : Enable or disable the AGC
- `bias` : Enable Bias-Tee power
- `ppm_correction` : Frequency correction for dongles with drift

RTL-TCP

- `ip_address` : IPv4 Server address
- `port` : Server port. Usually 1234
- `gain` : Device Gain in dBs
- `lna_agc` : Enable or disable the built-in LNA AGC
- `bias` : Enable Bias-Tee power (if supported)
- `ppm_correction` : Frequency correction for dongles with drift

Aaronia Spectran V6

- `ref_level` : Reference Level, in dBs
- `usb_compression` :
  - 0 is auto
  - 1 is raw
  - 2 is compressed

- `agc_mode` :
  - 0 is manual
  - 1 is peak
  - 2 is power
- `enable_amp` : Enable the amp
- `enable_preamp` : Enable the preamp

SDDC (RX888, RX999, etc)

Note : Support for those is experimental. Things may not work as expected!

- `mode` : 0 for HF, 1 for VHF
- `rf_gain` : RF Gain in dBs
- `if_gain` : IF Gain in dBs
- `bias` : Enable Bias-Tee power

SDRPlay

Note : some options are device-specific!

- `lna_gain` : LNA Gain in dBs
- `if_gain` : IF Gain in dBs
- `bias` : Enable Bias-Tee power
- `am_notch` : Enable the AM notch filter
- `fm_notch` : Enable the FM notch filter
- `dab_notch` : Enable the DAB notch filter
- `am_port` : Select the AM antenna port
- `antenna_input` : Select a specific antenna input. 0 is the first input
- `agc_mode` : AGC Mode, 0 is disabled :
  - 1 is 5Hz
  - 2 is 50Hz
  - 3 is 500Hz

SpyServer

- `ip_address` : IPv4 Server address
- `port` : Server port. Usually 5555
- `bit_depth` : Bit depth to stream at. Options are 8/16/32
- `gain` : Device gain in dBs
- `digital_gain` : Software gain, in dBs

SDR++ Server

Note : Using SDR++ in CLI mode as a source is possible, but all settings have to be setup from an UI connecting to the server beforehand, including samplerate!

- `ip_address` : IPv4 Server address
- `port` : Server port. Usually 5259
- `bit_depth` : Bit depth to stream at. Options are 8/16/32
- `compression` : Use SDR++ Server compression (true / false)

PlutoSDR

- `gain` : Device Gain in dBs
- `gain_mode` : Gain mode :
  - 1 is Manual
  - 2 is Fast Attack
  - 3 is Slow Attack
  - 4 is Hybrid

USRP

- `gain` : Device Gain in dBs
- `channel` : Channel ID
- `antenna` : Antenna ID
- `bit_depth` : Bit depth. Can be 16 on all devices, 8 or 12 on some

MiriSDR

- `gain` : Device Gain in dBs
- `bias` : Enable Bias-Tee power

File

Note

It is not recommended to use the file source for live/autotrack - use offline decoding instead - but it may be useful for testing purposes.

- `file_path` : Path to the baseband file (can be `/dev/stdin` on Linux/macOS)
- `baseband_type` : baseband type - can be u8, s8, s16, f32, or ziq
- `iq_swap` : Swaps I/Q samples

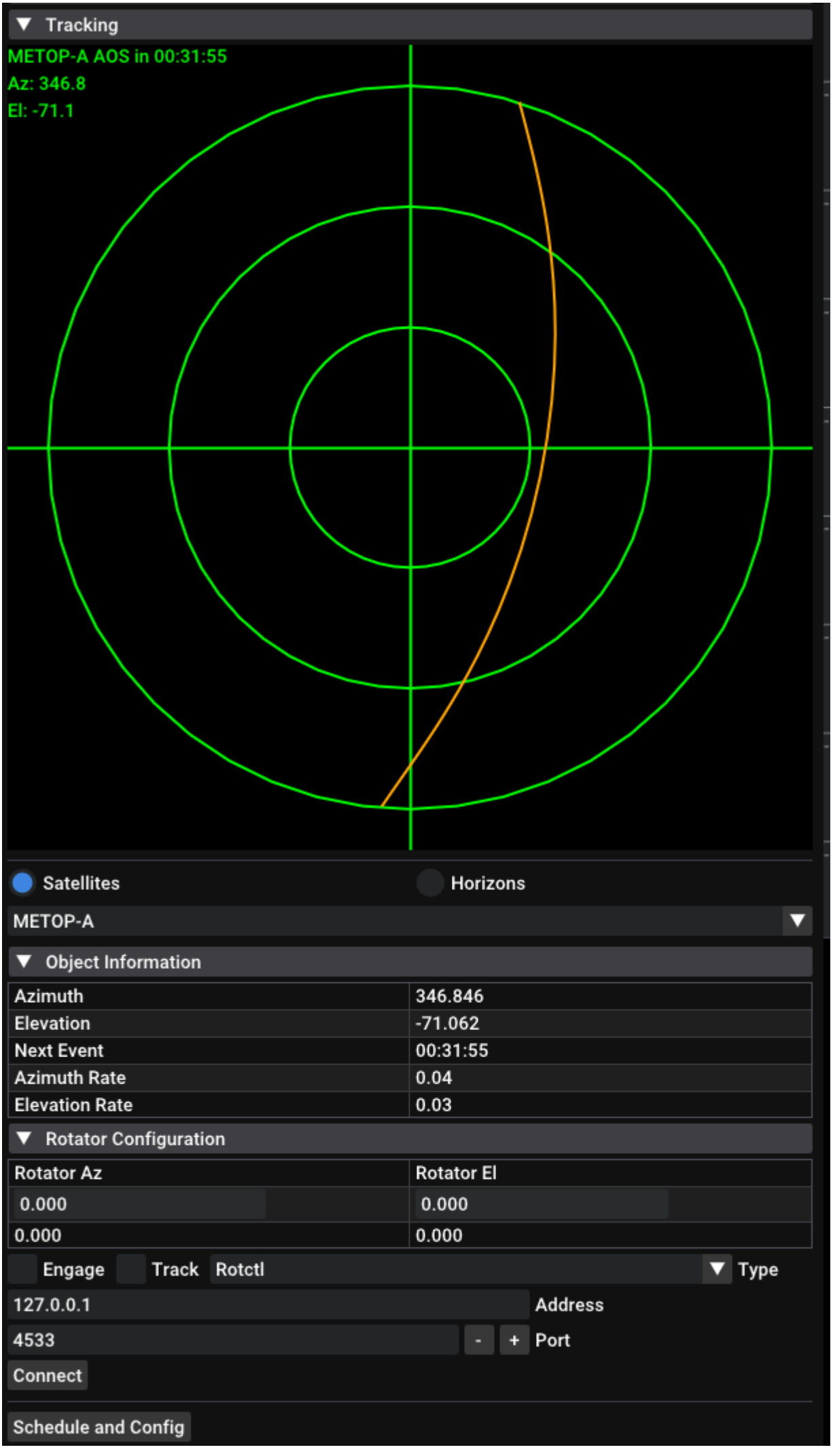
Satellite Tracker

The tracking widget can be used to automate reception of satellites.

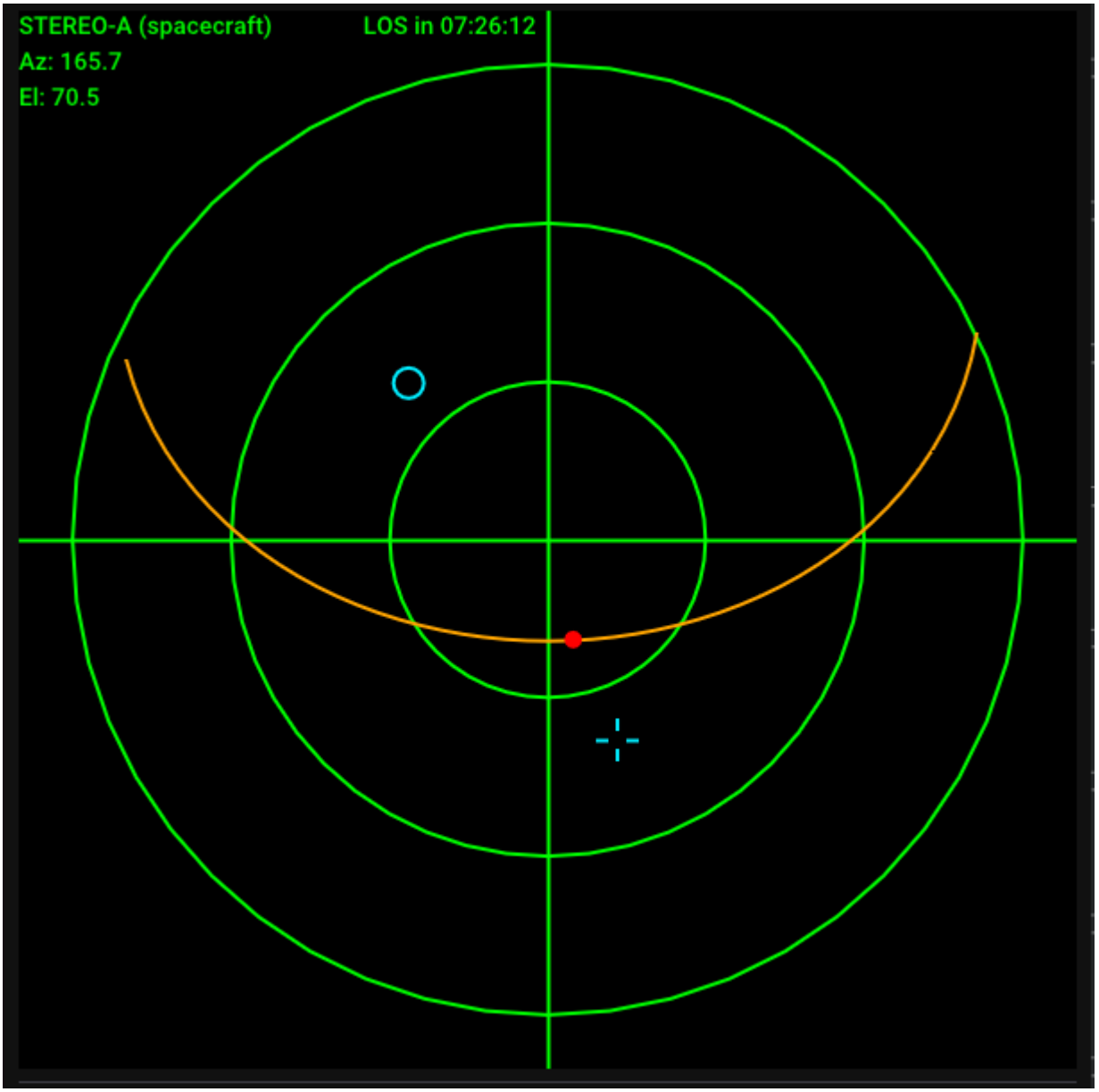
There are three supported configurations with the tracker:

1. **Manual (stand-alone)**: manually select a satellite and view information about it.
2. **Semi-automatic**: manually select a satellite and start the tracking, then automatically control an antenna rotator during that specific satellite pass.
3. **Full automatic (scheduler)**: automatically select a satellite according a schedule, start a SDR automatically, select a pipeline, control a rotator and process the pass automatically.

Tracker Interface


Tracking
METOP-A AOS in 00:31:55
Az: 346.8
El: -71.1
Satellites
Horizons
METOP-A
Object Information
Azimuth
346.846
Elevation
-71.062
Next Event
00:31:55
Azimuth Rate
0.04
Elevation Rate
0.03
Rotator Configuration
Rotator Az
Rotator El
0.000
0.000
0.000
0.000
Engage
Track
Rotctl
Type
127.0.0.1
Address
4533
-
+
Port
Connect
Schedule and Config

Polar Plot



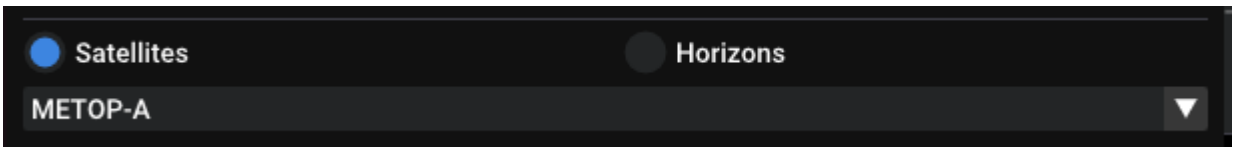
The polar plot is a chart of azimuth and elevation. The top of the plot is 0° azimuth, the right is 90° azimuth, bottom is 180°, and left is 270°. The center of the plot represents 90° elevation, while the first ring from the center is 60°, the second is 30°, and the outer ring is 0° (horizon).

- **Orange line**: represents the path of the satellite
- **Red dot:**: represents the position of the satellite
- **Blue cross**: represents the target attitude of the rotator
- **Blue circle**: represents the actual attitude of the rotator

In the top left corner, some information about the spacecraft is shown:

- Name of the spacecraft
- Current location information
- Acquisition of signal / loss of signal times

Satellite Selector



The satellite selector allows you to choose which object you are tracking. It has 2 modes, `Satellites` and `Horizons`.

Satellites

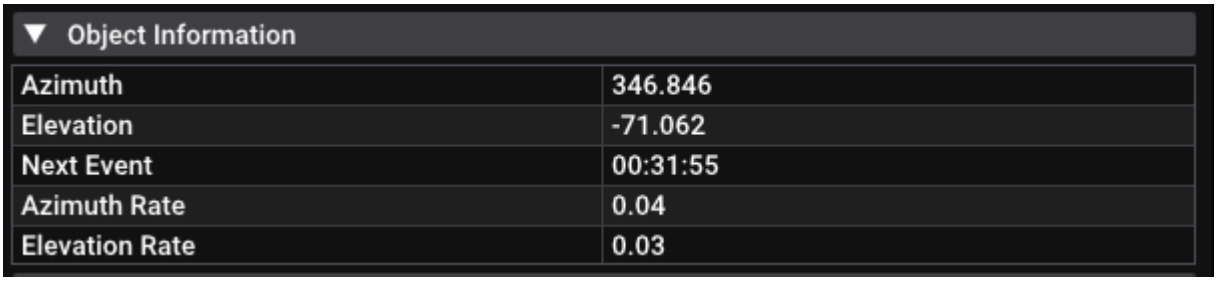
The `Satellites` mode will read from a list of TLEs (downloaded by SatDump).

Horizons

The `Horizons` selects out of Earth orbit spacecrafts from NASA's Horizons system.

> **Note**
>
> The `Horizons` mode will take a few moments to load, as it requests data from JPL's servers. For this reason, it will **not** work without an Internet connection.

Object Information



Displays information about the object you are tracking:

- Azimuth
- Elevation
- Next AOS / LOS events
- Rate of change of azimuth or elevation

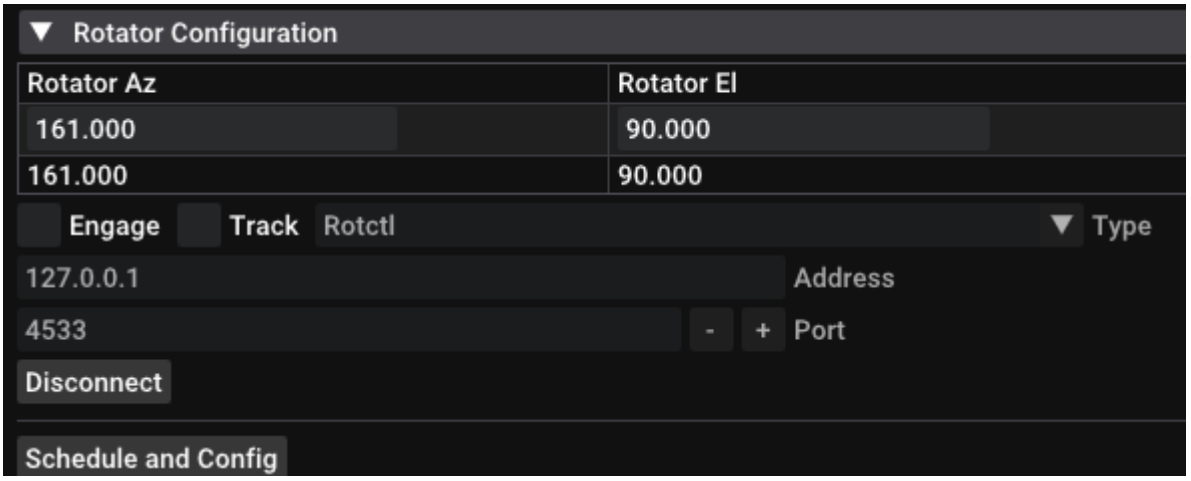
Interfacing SatDump with an antenna rotator

SatDump features support for satellite tracking using a generic antenna rotator interfacing through third party rotator controller daemons such as Hamlib's rotctld and YO3DMU's PstRotator.

While Hamlib rotctld and PstRotator handle the hardware communications requirements and the specific rotator protocols and configuration, SatDump communicates with these apps at software level using a common protocol.

This allows SatDump to agnostically control and command every type of rotator supported by these two daemons without having to deal with complex or specific rotator settings.

Such option allows to have either the rotator controller connected directly to the same PC where SatDump is running, but also to connect via network to a remote different computer that handles the rotator, for example in a shack.



Rotator UI

- `Rotator Az` and `Rotator El` control and display the rotator position.
    - **The first row is an editable text box that shows the latest commanded position.**
        - These will be automatically updated when a satellite is being tracked.
        - If the selected satellite is below horizon or not being tracked, you can manually enter the desired position (for example, to test the rotation).
    - The second row shows the actual position reported by the rotator, where is it pointing currently. In order to work, the rotator daemon must be running and connected to SatDump.
- `Engage`: enables the sending of position commands to the rotator. Whenever a new position is written and `Engage` is enabled, the rotator will be commanded to go there.
- `Track`: enables tracking. When enabled, the position entry text box will be automatically updated with the satellite position while it is above the horizon or the expected AOS position if it's not in sight yet.
- **Protocol Selection**: selects the protocol or type of controller daemon you want to use. Currently, only rotctl and PstRotator are supported.

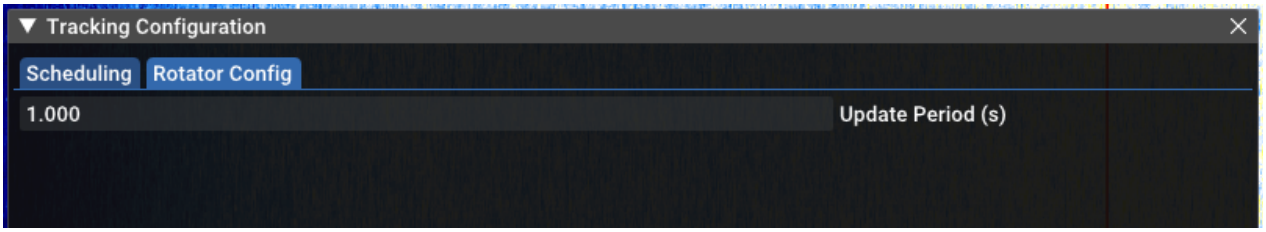
> **Note**
>
> PstRotator is untested. Rotctl is preferred and highly recommended.

- **Controller Daemon TCP/IP Settings**: allows configuration of the network communication settings with the controller daemon. * `Address`: IP address of the computer where is the controller daemon server running.

  > **Note**
  >
  > Enter `127.0.0.1` if Rotctld or PstRotator are running on the same computer as SatDump is, or the remote computer IP if the daemon is running on a different computer.

  - `Port`: **Defines the TCP port where is listening the daemon server.** `4533` is Rotctld's default port, while `4002` is the default for PstRotator.
  - `Connect/Disconnect`. **This button connects and disconnects from the** daemon server controller.

- `Schedule and Config` opens a window containing the `Scheduling` tab and a `Rotator Config` tab.

  The Rotator Config tab contains the `Update Period` text box. This is the refresh rate for the rotator controller expressed in seconds. This value defines the polling and rotating intervals.

Configuration with Hamlib's rotctld

This guide only cover the basic instructions for Rotctld. Please refer to Hamlib's documentation and guide for details on how to download and install it, as well to setup your rotator.

- Launch the rotctld app with the settings required for your specific hardware rotator controller.
- Go to SatDump and select `rotctld` in the configuration panel.
- Type the IP Address and TCP port as required (4533 the default port).
- Click `Connect`.
- Click `Engage` to start sending commands to the rotator.

Configuration with PstRotator

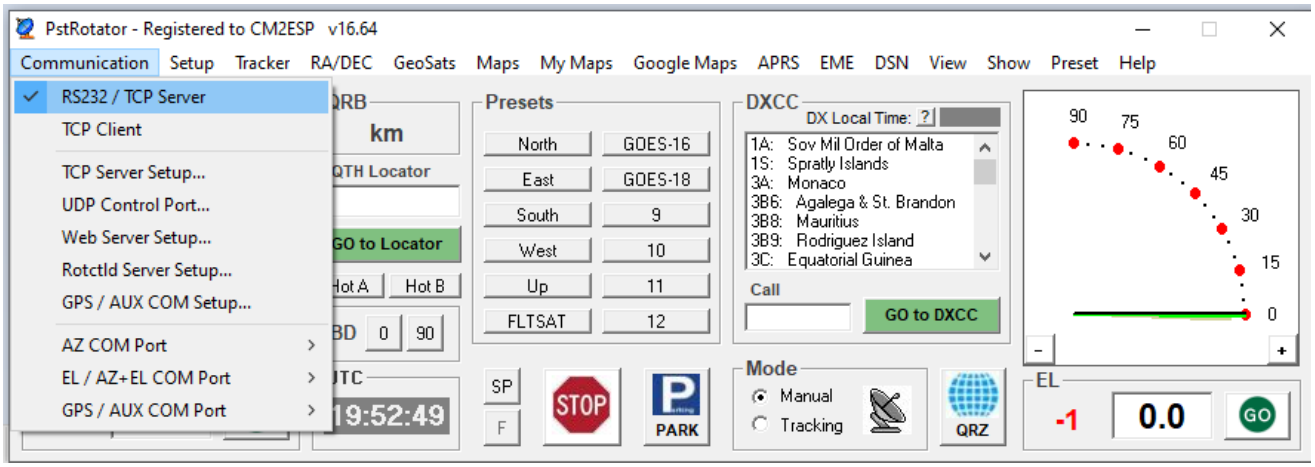
> **Note**
>
> PstRotator is untested. Rotctl is preferred and highly recommended.

This guide only cover the basic instructions for PstRotator. Please refer to YO3DMU's PstRotator Website for details on how to download and install it, as well to setup your rotator. Most of the configuration steps in PstRotator only needs to be done one time as it will save the settings when closed.

Using PstRotator protocol

> **Note**
>
> This method is **not** recommended. PstRotator's rotctld protocol emulation should be used instead.

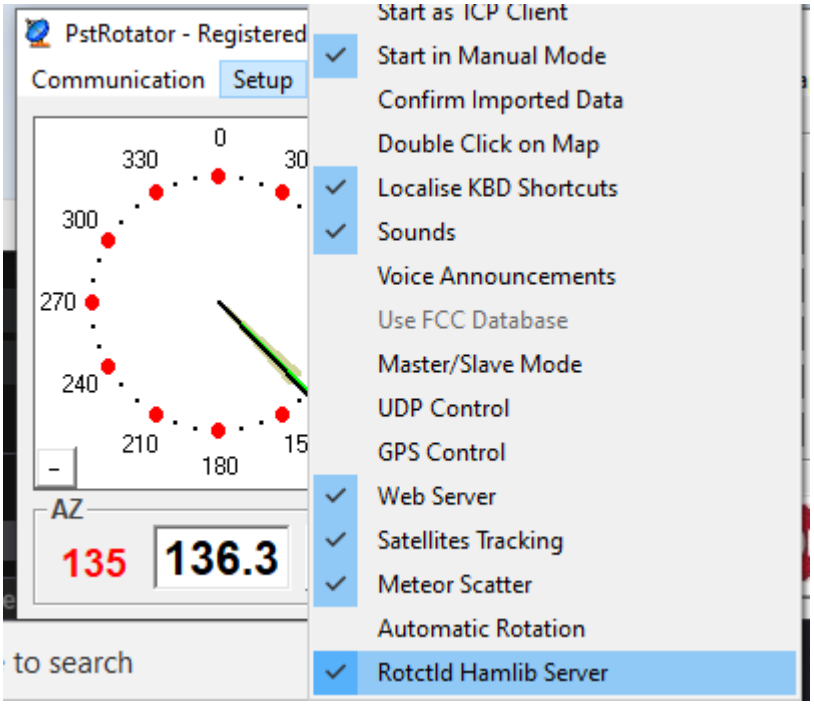
- Launch PstRotator and configure the settings for your specific hardware rotator controller.
- On the Communication Menu, select and enable RS232 / TCP Server. This setting should be enabled by default.



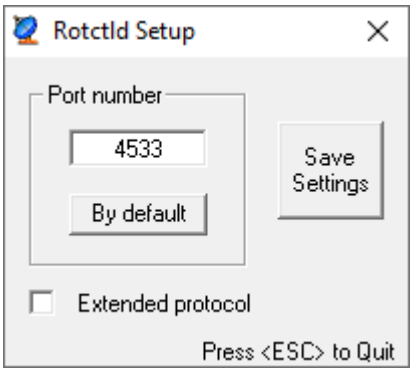
- In the `Mode` box in PstRotator select `Manual`. While PstRotator includes internal modules for both satellite and astronomical objects tracking, it's highly recommended not using them while using it with Satdump to avoid conflicts.
- It's also recommended to enable the options `Start in Manual Mode` and `Localized KBD Shortcuts` in the PstRotator Setup Menu to avoid conflicts and tracking issues.
- Go to SatDump and select `PstRotator` in the configuration panel.
- Type the IP Address and TCP port as required (4002 is the default port)
- Click `Connect`
- Click `Engage` to start sending commands to the rotator.

Using PstRotator Hamlib Rotctld protocol emulation

- Launch PstRotator and configure the settings for your specific hardware rotator controller.
- Navigate into Setup Menu and click to enable `Rotctld Hamlib Server`. Be sure you don't have an actual Rotctld instance running as both apps will try to use the same TCP port.



- On the Communication Menu, click at `Rotctld Server Setup` and check that default port 4533 is selected. Also uncheck and verify that `Extended protocol` is not selected. Click `Save Settings` and press Escape to close the config dialog.



- Go to SatDump and select `rotctl` in the configuration panel. panel

> **Warning**
>
> Do not select `PstRotator`.

- Type the IP Address and TCP port as required (4533 is the default port)
- Click in Connect.
- Click `Connect`
- Click `Engage` to start sending commands to the rotator.

Composite System

Basics

Selecting a single channel

`chX` where `X` is the number of the channel

Selecting a single calibrated channel

> Only for these satellites and instruments * AMSU: all satellites carrying it * MHS: all satellites carrying it * ATMS: all satellites carrying it * AVHRR: NOAA 18, NOAA 19, METOP B, METOP C

`cchX` where `X` is the number of the channel.

The output value will be in the range of 0 to 1. The value of 0 and 1 depends on two things:

- The default (or specified in the `satdump_cfg.json`) calibration type, such as albedo, radiance or temperature
- The default (or specified in the `satdump_cfg.json`) range, for example for AVHRR channel 4 temperature, 0 equals to 232.2 K, 1 equals to 304.9 K.

For more information on calibration, see the Advanced section below.

Simple RGB composite

`chR, chG, chB` (normal)

`cchR, cchG, cchB` (calibrated)

Math

Any math operator is supported, such as `+, -, /, *, cos(), sin(), ln(), ^` and more.

Example: `ch1^2.5, ch2, ch3` (raises ch1 by a power of 2,5).

The acceptable value range for the output of each channel is 0 to 1 (see above for notes regarding calibration).

Inverting a channel is simple, just subtract 1: `1-ch4` will invert channel 4.

Ternary operators

You can use a ternary operator to select a result based on a condition.

Syntax: `input condition ? true : false`

Example: `cch1 < 0.5 ? 1 : 0`

Explanation: If `cch1` is less than 0.5, output 1 (white). If not, output 0.

Ternary operators support these conditions: `<, >, >=, <=, ==` (less than, greater than, greater or equal than, equal to)

Ternary operators can be nested, but the algorithm will quickly become quite cumbersome to read. I recommend to use Lua for more complex algorithms.

Alpha Channel (RGBA)

By adding a fourth parameter to a RGB equation, it is possible to specify an alpha channel (this is useful in combination with the ternary operator).

Syntax: `chR, chG, chB, chA`.

It can be used for monochrome composites that are meant to be used together with the LUT option. In that case, the G and B parameters are set to 0.

Example: `cch1, cch2, cch3, cch4 > 0.5 ? 1 : 0`

Explanation: Output a RGB image composed by cch1 (red), cch2 (green), cch3 (blue). If cch4 is greater than 0.5, the alpha channel is 1 (totally opaque). If not, it's 0 (totally transparent).

The typical use case would be blending of sounder and imager data, for example to show a rainfall map from MHS on AVHRR.

Advanced

The advanced part covers syntax in the `satdump_cfg.json` configuration file, found in SatDump's configuration directory. Also, these settings can be added in the user-specific configuration file called `settings.json`.

The user-specific `settings.json` has priority over `satdump_cfg.json`, therefore editing

`settings.json` can override the default composites, without needing to edit the configuration file.

Beware that sometimes the default composites get updated, such as when SatDump is updated, therefore if you override them unexpected behavior can occur. If this happens, please rename the `settings.json` and test again, to isolate the issue.

The files can be found:

- Linux:
  - General config file: `/usr/share/satdump/satdump_cfg.json`
  - User config file: `~/.config/satdump/settings.json`
- Windows:
  - General config file: `C:\Program Files\SatDump\share\satdump\satdump_cfg.json`
  - User config file: `%appdata%\satdump\settings.json`
- MacOS:
  - The general config file **cannot** be edited due to Apple restrictions.
  - User config file: `~/.config/satdump/settings.json`

The basic syntax is as follows:

```
"Composite Name": {
   "equation": "cch1, cch2, cch3"
}
```

Additional parameters can be added (mind the commas!):

```
"Composite Name": {
  "equation": "cch1, cch2, cch3",
  "individual_equalize": true
}
```

Common parameters:

All these parameters are boolean and accept either a `true` or a `false`.

If omitted, they will be treated as `false` (for example, if you don't specify `"equalize": true`, the image will not be equalized). Forcefully setting these parameters to `false` is redundant.

- `individual_equalize`: histogram equalize each individual red, green, and blue component before combining them into an RGB output.
- `equalize`: combine red, green and blue components, then histogram equalize the result.

- `white_balance`: apply white balance to the result (useful for Meteor LRPT where black lines can sometimes mess up the equalization).
- `despeckle`: apply despeckling through the use of a Kuwahara filter (usually not needed).
- `normalize`: apply a normalising algorithm to the image.
- `apply_lut`: apply a LUT to the resulting RGB image. If only the red channel is specified, and all others are at zero, it will apply the LUT to the single channel as if it were a grayscale image.
- `geo_correct`: correct the output image to eliminate distortion caused by the Earth's curve.
- `autogen`: if set to `false`, it will prevent the autogeneration of this composite when SatDump processes the pipeline (so, the composite will only be available from the viewer).

Calibration parameters

By adding a `calib_cfg` block as follows, it is possible to select many parameters regarding calibration.

```
"Composite Name": {
  "equation": "cch1, cch2, cch3",
  "calib_cfg": {
      "cch1": {
         "type": "albedo",
         "min": 0,
         "max": 40
      },
      "cch2": {
         "type": "radiance",
         "min": 0.5,
         "max": 2.2
      },
      "cch3": {
         "type": "temperature",
         "min": 220,
         "max": 460
      },
   }
}
```

Specific configurations can be applied for each channel independently. If a calibrated channel is used, but a specific configuration is not set, the default will be used.

- `type`: selects the calibration type. Accepted values are `radiance`, `albedo` or `temperature`.
- `min` and `max`: select the range that is remapped from 0 to 1 in the viewer.

The units are Kelvin for temperature, W·sr−1·m−2 for radiance, and percentage (%) of albedo (All standard SI units)

Note

If you want to work with Kelvin directly in the equation, set `min` to 0 and `max` to 1000. Then, 273K will be 0.273 in the equation.

Projections

It is possible to customise and automate projections for each composite. **Do not enable this parameter by default if you want to do a pull request**, as it will result in many unnecessary files and takes quite a lot to process. Only add it in your `settings.json` for your own convenience :)

For example, this is the simplest projection that will automatically create a GeoTIFF image (for later use in other software) of the proper size and projected using the equirectangular projection:

```
"Composite Name": {
   "equation": "ch2, ch2, ch1",
   "project": {
      "draw_map_overlay": true,
      "individual_equalize": true,
      "img_format": ".tif",
      "config": {
         "type": "equirec",
         "auto": true,
         "scalar_x": 0.016,
         "scalar_y": -0.016
      }
   }
}
```

For further information please read the projections page.

Custom LUTs

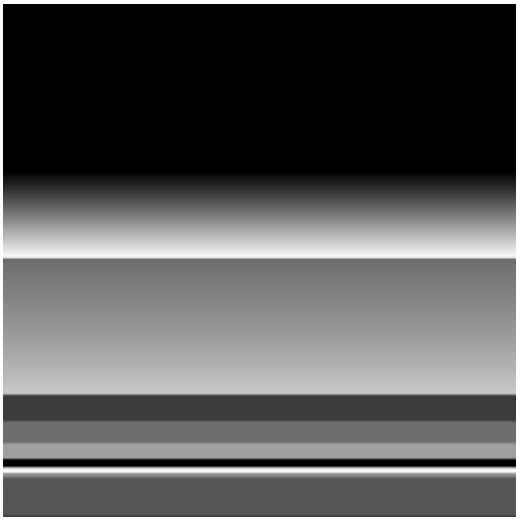
A custom square PNG LUT can be specified. It has to be put in the `resources/lut` folder of SatDump.

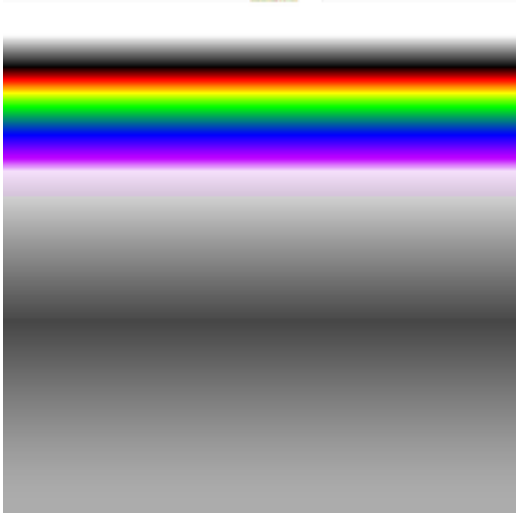
```
"Composite Name": {
   "lut": "lut/LUT_FileName.png"
   "channels": "cch1, cch2",
}
```

All of the common parameters such as `individual_equalize` can be used, although they won't affect the LUT (apart from `geo_correct`).

- `lut`: the LUT file name and path relative to `resources/`
- `channels`: specified the 2 channels for the two dimensional LUT.

LUTs can be either grayscale or color and are usually 256x256 pixels in size.





Examples

Overriding the default configuration

Thanks to the user-specific `settings.json` it is possible to override the default configuration via the `viewer` array.

Here is an example that overrides some MSU-MR composites and adds new ones.

```
//to be put after the satdump_general array
"viewer": {
   "instruments": {
      "msu_mr": {
         "rgb_composites": {
            "221": {
               //Since 221 is already present in the main satdump_cfg.json, no need to
specify the equation again, just what you want to override
               "autogen": true,
               "equalize": true,
               "white_balance": true
            },
            "421": {
               //421 is a new composite, so it has to be specified entirely.
               "equation": "ch4, ch2, ch1",
               "autogen": true,
               "geo_correct": true,
               "equalize": true
            }
         }
      }
   }
}
```

Note

If you just want to enable autogeneration of some products that have `"autogen": false` set in their options, just add this and put it in the section for the instrument name as shown above:

```
"composite_name": {
   "autogen": true
}
```

Projection System

Basic example

This is the simplest projection that will automatically create a GeoTIFF image (for later use in other software) of the proper size and projected using the equirectangular projection. It will then be explained further later.

```
"Composite Name": {
   "equation": "ch2, ch2, ch1",
   "project": {
      "config": {
         "type": "equirec",
         "auto": true,
         "scalar_x": 0.016,
         "scalar_y": -0.016
      },
      "draw_map_overlay": true,
      "individual_equalize": true,
      "img_format": ".tif"
   }
}
```

Projection types

`type` :

- `equirec` (equirectangular)
- `stereo` (sterographic)
- `utm` (Universal Transverse Mercator)
- `geos` (for geostationary satellites)
- `tpers`
- `webmerc`

Common parameters for all projections

These can be omitted in automatic mode, or can be set anyway to override. If they are omitted but needed, they default to zero.

- `lat0` (float) Latitude of the origin of the projection
- `lon0` (float) Longitude of the origin of the projection
- `offset_x` (float) Offset of the projection from the center (in meters)
- `offset_y` (float) Offset of the projection from the center (in mesters)
- `scalar_x` (float) Scale of the projection (in meters per pixel)
- `scalar_y` (float) Scale of the projection (in meters per pixel)

Extra parameters for UTM

- `zone`: (int) UTM zone
- `south`: (boolean) if you're north (false) or south (true) of the equator

Extra parameters for GEOS

- `altitude`: (float) Altitude of the satellite (in meters)
- `sweep_x`: (boolean) Sweep mode of the GEOS projection

Extra parameters for TPERS

- `altitude`: (float) Altitude of the satellite (in meters)
- `tilt`: (float) Tilt of the satellite (in degrees)
- `azimuth`: (float) Azimuth of the satellite (in degrees)

Alternative mode for Stereo

Mandatory:

- `center_lat` (float) Latitude of the center of the projection (degrees)
- `center_lon` (float) Longitude of the center of the projection (degrees)
- `scale` (float) Scale of the image(meters per pixel)
- `width` (int) Width of the image (in pixels)
- `height` (int) Height of the image (in pixels)

Alternative mode for TPERS

Mandatory:

- `center_lat` (float) Latitude of the center of the projection (degrees)
- `center_lon` (float) Longitude of the center of the projection (degrees)
- `scale` (float) Scale of the image(meters per pixel)
- `width` (int) Width of the image (in pixels)
- `height` (int) Height of the image (in pixels)

Optional:

- `altitude`: (float) Altitude of the satellite (in meters)
- `tilt`: (float) Tilt of the satellite (in degrees)
- `azimuth`: (float) Azimuth of the satellite (in degrees)

Alternative mode for UTM

Mandatory:

- `scale` (float) Scale of the image(meters per pixel)
- `width` (int) Width of the image (in pixels)
- `height` (int) Height of the image (in pixels)

Optional:

- `zone`: (int) UTM zone
- `south`: (boolean) if you're north (false) or south (true) of the equator

Products Specification

The products format in SatDump shall be the output of an instrument decoding step. The idea is not to make a format like HDF-5 entirely taking over storing the actual data, but more to store what can't be stored in let's say, an image file or other "user-friendly" formats.

Before 1.0, it was considered to move over SatDump's outputs to HDF-5 or similar formats, but that was abandonned as in most cases, all that's really required is storing some image data or readings, something... PNG for example will cover perfectly without requiring any usual SW for users to work with them. For things like radiation that cannot be stored in any really standard format, there before was no way to store the raw readings either.

Hence, such a products system should be both able to store metadata (projection parameters for example, replacing the previous .georef file format) and actual data if no other format can be used to better fit the purpose.

The products file uses a JSON-like format called CBOR, which being binary-based allows for smaller filesizes. It should be written in the instrument's output directory alongside other files such as images.

Another main goal of this format is to generalize processing as much as possible. Any imager for example should be able to utilize this system with minor changes and pre-processing, instead of having custom code for each.

Common Fields

The fields enumerated here should be common to ANY products file.

- `instrument` : Instrument name, eg `avhrr`, `modis`, `sem`. This is the unique instrument ID (lowercase) allowing to identify what satellite instrument this data is from, and hence let further processing SW decide what to do. If there are several massively different versions of a same instrument, this ID should be different.
- `type` : Products type. This holds an unique string ID (lowercase) identifying the product sub-type.
- `tle` : Optional field holding a TLE struct. This should be the TLEs revelant for processing of the provided data, given it is necessary.

Image Products

The type of those products should be set to `image`. This products type aims at storing anything that can be treated as image data. This should cover all types of satellite imagers and sounders that do cover more than a single point (and hence, do image a decent swath).

After all, further processing of a Visible & IR spectrometer will usually be either about composing several channels to get a color image or converting to radiance and perhaps temperatures to then apply a LUT or feed to some other algorithm. That's pretty much the same as you would do for a microwave sounder, and not far off what could be done with an infrared sounder capable of covering several layers of the atmosphere. Therefore, many common processing and metadata needed for processing are the same for all those instruments, allowing to cover so many in a pretty generic format.

Image data for those products should be stored in .png files of either 16 or 8-bit depth.

- `bit_depth` : Actual bit depth of the image data
- `needs_correlation` : This should be set to true if there is no guarantee all channels are synced between each-other. This is the case on LRPT and HRD for example.
- `save_as_matrix` : Some sounders, such as IASI can have several thousands channels. In those cases, it may be preferred to save them all in a single .png. Setting this option to true will do so.
- `images` : Actual image channels, which also have several sub-fields.
  - `file` : Path to the .png file to load, relative
  - `name` : Channel name. Usually a single number, but can be "m5" or similar where convenient to stay in line with official channel naming.
  - `ifov_y` : IFOV size vertically. Only required if needs_correlation is true and timestamps don't simply cover a scanline.
  - `ifov_x` : IFOV size horizontally. Same as above.
  - `offset_x` : Some instruments have channels offset from each other by a bit. This allows accounting for it.
- `wavenumbers` : Vector of wavenumbers for each channel. Optional, but may be required for calibration
- `has_timestamps` : Boolean indicating the presence or lack of timestamps alongside the image.
- `timestamps_type` : TBD, maybe switch to a string? Optional
- `timestamps` : The actual timestamps data. If it varies between images, it should be set per-image. Optional
- `projection_cfg` : Projection configuration for geo-referencing the imagery data. Optional
- `calibration` : Calibration settings to convert the raw imagery data to physical units. Optional. Rest TBD

Official Products

SatDump supports decoding and processing certain official products from space and meteorological agencies.

Note

This module is experimental and could contain bugs; users are encouraged to test and report any issues on the GitHub.

All products must be decoded with the `Data Stores/Archives Formats (NOAA/EUMETSAT/CMA/NASA), Experimental` pipeline.

List of supported products

EUMETSAT

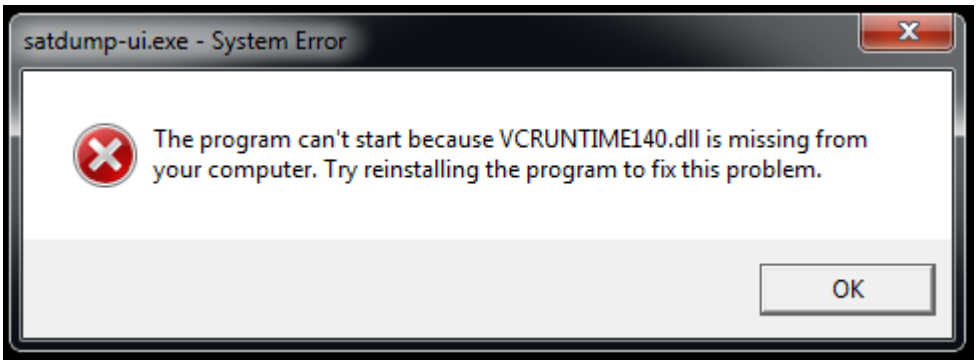
Product	Satellite	Instrument
High Rate SEVIRI Level 1.5 Image Data - MSG - 0 degree	MSG	SEVIRI
High Rate SEVIRI Level 1.5 Image Data - MSG - Indian Ocean	MSG	SEVIRI
Rapid Scan High Rate SEVIRI Level 1.5 Image Data - MSG	MSG	SEVIRI
FCI Level 1c Normal Resolution Image Data - MTG - 0 degree	MTG	FCI
FCI Level 1c High Resolution Image Data - MTG - 0 degree	MTG	FCI
AVHRR Level 1B - Metop - Global	Metop	AVHRR
IASI Level 1C - All Spectral Samples - Metop - Global	Metop	IASI
AMSU-A Level 1B - Metop - Global	Metop	AMSU-A
MHS Level 1B - Metop - Global	Metop	MHS
OLCI Level 1B Full Resolution - Sentinel-3	Sentinel-3	OLCI

FAQ/Troubleshooting

Are you running into issues getting started with SatDump? No problem! We've compiled a list of common problems and trobleshooting steps to get you going in no time. As a general rule of thumb, make sure you try the latest nightly build from the downloads page before trying anything else, as we may have fixed your issue.

Missing DLLs Reported on Windows

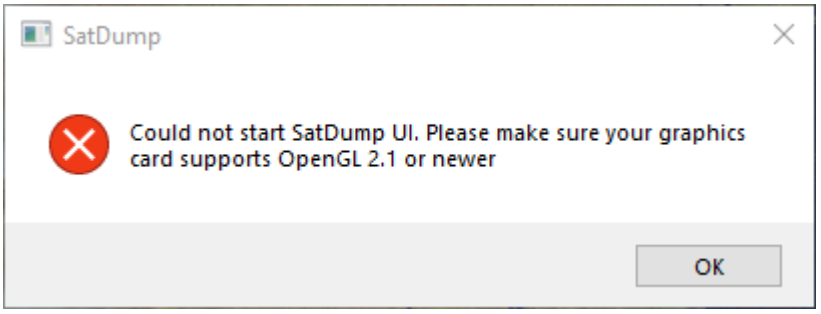
Problem: When Launching SatDump UI, CLI, or SDR Server on Windows, you are presented with an error about missing dll(s):



Solution: You have not installed the Microsoft Visual C++ Redistributable for 2015-2022. Please download and install the latest version from https://aka.ms/vs/17/release/vc_redist.x64.exe.

Issues Launching SatDump UI

Problem: When launching SatDump UI on any OS, a GLFW error appears in the console, the following error message pops up and the program exits, or you are presented with a black screen.



Solutions: This happens when your graphics card does not properly support OpenGL 2.1 or higher. Don't worry - there are a number of workarounds you can try, depending on your

Operating System.

Windows

On Windows, your first line of defense it to make sure your graphics drivers are up-to-date. The “Microsoft Basic Display Adapter” does not support OpenGL, and you should install the correct graphics drivers for your system.

If installing proper drivers is not possible, or if you have an old card where the newest drivers do not support OpenGL 2.1, there is a workaround. Download the 64-bit opengl32.dll from https://fdossena.com/?p=mesa/index.frag. Extract the download, and copy the DLL into your SatDump program folder. Afterwards, SatDump should open normally.

> **Note**
>
> The custom DLL uses software rendering to display SatDump. This will result in a low framerate, but the software should be usable.

Linux

On Linux, you also want to make sure your graphics driver is up-to-date. Most Linux distros ship with OpenGL drivers for the majority of common graphics cards, so this is typically not a concern. Instead, try the following commands from a terminal to see if you can get SatDump to launch

- `MESA_GL_VERSION_OVERRIDE=4.5 satdump-ui`: Forces SatDump to try using OpenGL 4.5
- `MESA_GL_VERSION_OVERRIDE=2.1 satdump-ui`: Forces SatDump to try using OpenGL 2.1
- `LIBGL_ALWAYS_SOFTWARE=1 satdump-ui`: Forces SatDump to use software-only rendering

Building for OpenGL ES 2.0: Some systems, especially embedded systems or SBCs like the BananaPi and OrangePi, do not support OpenGL. Instead, they support OpenGL ES 2.0. SatDump supports OpenGL ES 2.0, but you must build it specifically for these systems. Add `-DBUILD_GLES=ON` to your SatDump cmake command to enable support.

> **Note**
>
> All modern distros for the Raspberry Pi Support Desktop OpenGL 2.1 or higher, so building for GL ES on these systems is not required.

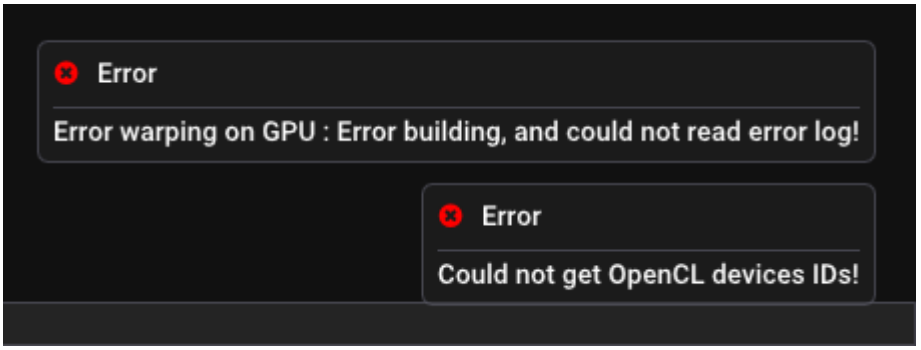
macOS

macOS ensures that OpenGL works out-of-the-box on all supported hardware/OS combinations. If you are experiencing OpenGL issues, you most likely have a misconfigured Hackintosh, you have modified your Mac’s bootloader to run newer versions of macOS than

is officially supported, or you are running macOS in a Virtual Machine. These configurations are not supported, and there are no workarounds available.

OpenCL Issues

Problem: When processing products or generating projections, errors as shown below appear and processing takes a long time.

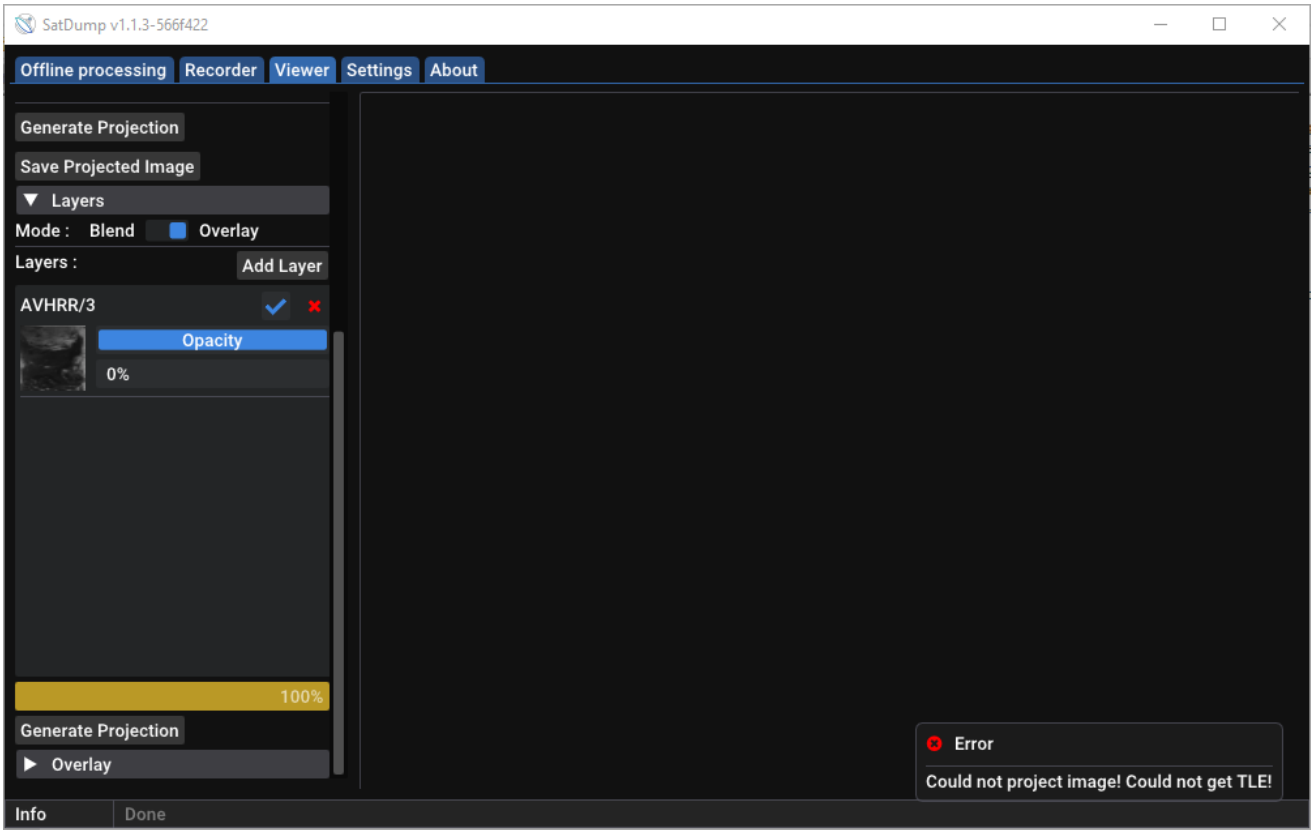


Solutions: SatDump uses OpenCL to accelerate many computations, including those used in making overlays/projections. This error happens when SatDump cannot access OpenCL for some reason. SatDump will still work as expected, but at a fraction of the speed. Resolving the error depends on your Operating System, CPU, and Graphics Card.

- **Windows - Intel/AMD/NVidia Graphics:** OpenCL is included with the latest graphics drivers for your card. Try updating your graphics drivers to resolve OpenCL issues.
- **Windows - Intel CPU:** If GPU-based OpenCL is not working for you, you can try installing the CPU-based OpenCL runtime for Intel processors from Intel's website.
- **Linux - Intel Integrated Graphics:** Install the `intel-opencl-icd` (or equivalent) package from your distro's package manager.
- **Linux - AMD Graphics:** Follow the instructions at this site to install OpenCL drivers. It is *not recommended* that you install the full AMD Pro drivers.
- **Linux - NVidia Graphics:** Install the proprietary NVidia drivers on your system and reboot. You may need to add your user to the `video` group and reboot before OpenCL works.
- **Linux - Intel CPU:** Install the `intel-oneapi-runtime-opencl` (or equivalent) package to enable OpenCL on CPU.
- **macOS:** OpenCL should be supported on all Macs as of this writing. If you experience any issues, try building SatDump from source or create an issue on GitHub.
- **Android:** Android does not officially support OpenCL, but it does work on all tested devices. If you have an Android device that does not support OpenCL, create an issue on GitHub.
- **SBCs like the Raspberry Pi:** OpenCL is not supported on SBCs at this time.

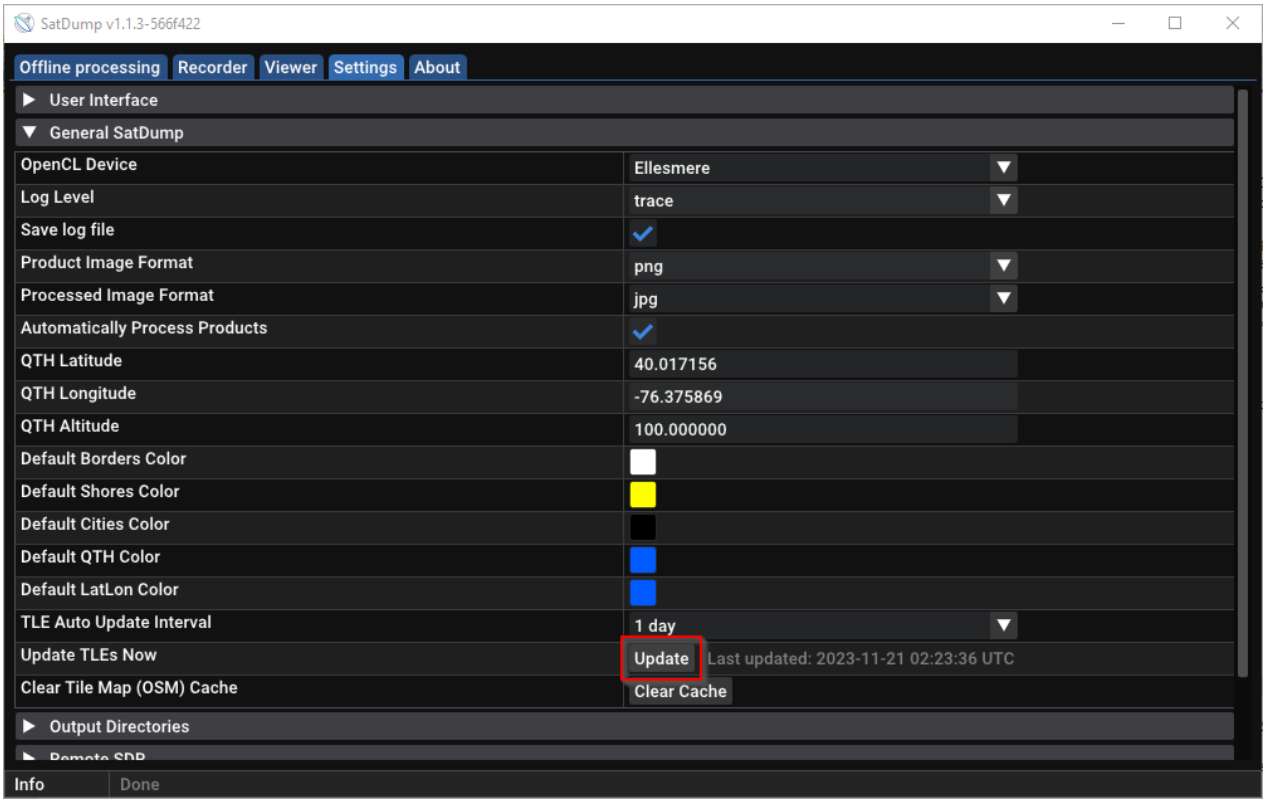
Projection or map overlays are missing/incorrect

Problem: When attempting to use the map overlay or projection features in SatDump, they either don't work at all, or the map/image is in the wrong place.



Solutions: Projections/overlays can be incorrect for a number of reasons:

- **Your TLEs are out-of-date/missing:** go to SatDump Settings > General SatDump > Update TLEs now to update your TLEs, then re-decode the data.



- **You did not set the start time of the pass:** NOAA APT requires a valid start timestamp in order to project imagery. Meteor HRPT also requires a rough estimate of the pass start time, if you're decoding it more than a day after the pass. Verify that you have specified the timestamp and try again
- **The Satellite's on-board clock is wrong:** sometimes satellites lose track of time - but do we all? The Meteor satellites are notorious for this behavior. Unfortunately, there's not

much you can do if this happens.

SDR Issues

SDR issues can be specific to your platform and SDR, but there are general troubleshooting steps you can try.

On Windows, SatDump cannot see my SDR

The inability to see/start SDRs in Windows usually comes down to improper driver installation. Follow the typical driver installation instructions for your SDR to get started.

Here's a list of drivers for popular SDRs:

- **RTL-SDR (including NooElec, etc), AirSpy, AirSpy HF, and HackRF:** WinUSB, installed automatically or via Zadig
- **Miri SDRs (often sold as RSP1 "clones"):** libusb-win32, installed via Zadig. You may need to uninstall/reinstall the driver a few times for it to work correctly. Make sure the SDRPlay API is not installed or running when using this driver!
- **SDRPlay SDRs (RSP1a, RSPDuo, RSPdx):** Use the official SDRPlay API from the SDRPlay website.

On Linux, SatDump cannot see or open my SDR

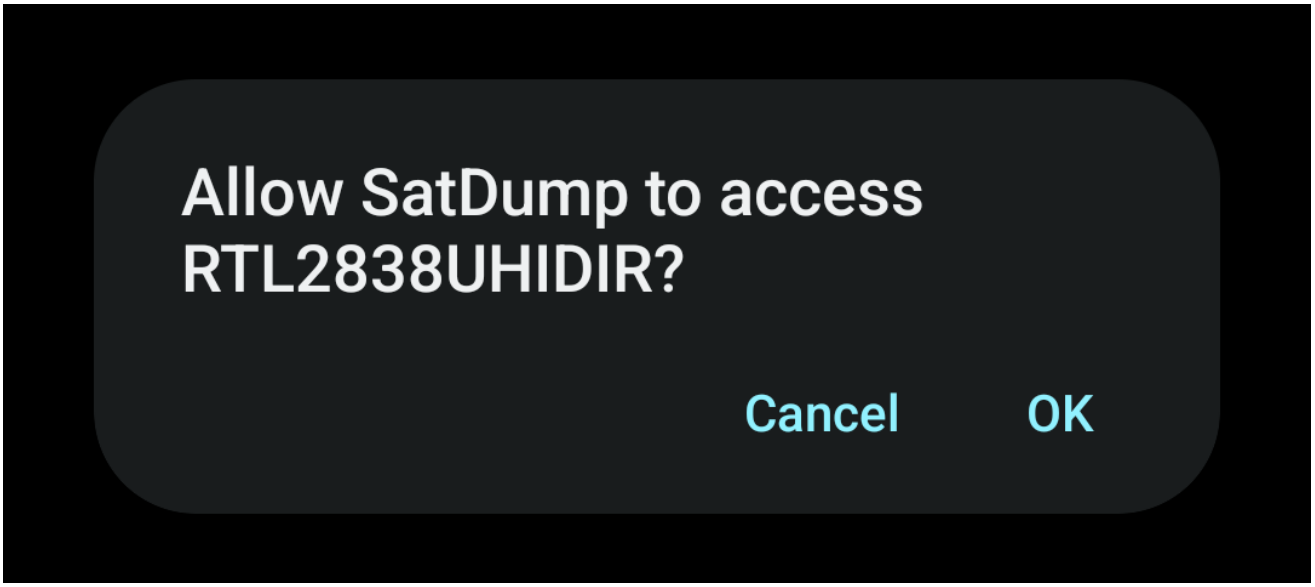
On Linux, the correct libraries to access your SDR should already be installed by the build/installation process. However, for some SDRs, you may need to blacklist kernel modules or set up udev rules.

- **RTL-SDR:** Ensure the RTL-SDR udev rules are set up under `/lib/udev/rules.d/`. On some systems, you may need to add your user to the `plugdev` group and restart.
- **MiriSDR:** You may need to blacklist some kernel modules. Edit `/etc/modprobe.d/blacklist.conf` and add the following lines:

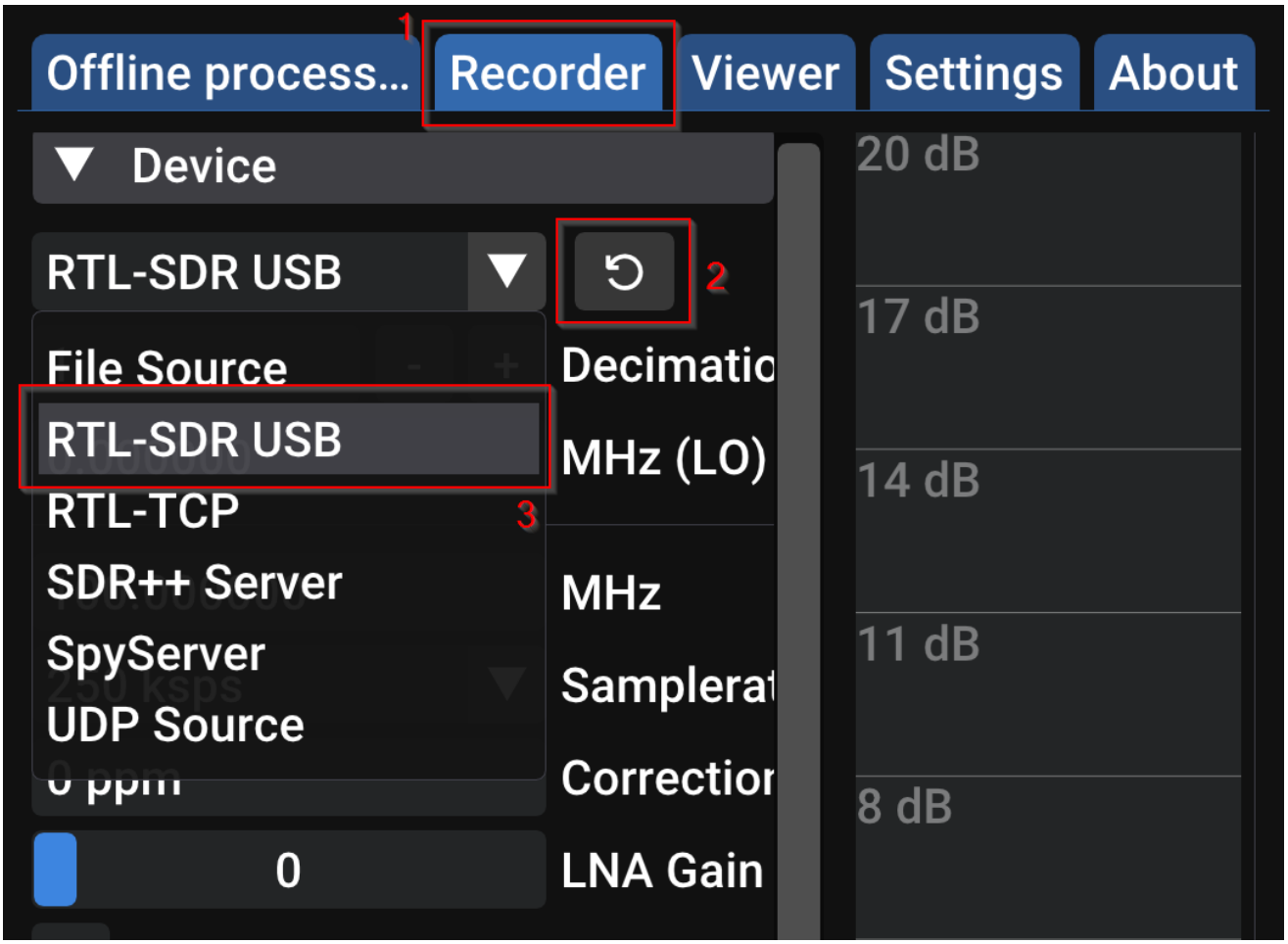
```
blacklist sdr_msi3101
blacklist msi001
blacklist msi2500
```

I cannot access my SDR on SatDump for Android

Android has support for a limited number of SDRs over USB. Supported devices include RTL-SDR, Airspy, AirspyHF, LimeSDR Mini, and HackRF. The first time you use a USB-based SDP with your Android phone, it will prompt you to allow access to the SDR. Tap OK.



Then, go to the Recorder screen, and tap “refresh” for your SDR to show up.



Where are SatDump logs?

SatDump’s logs are named satdump-<timestamp>.log. Logs older than 3 days old are automatically deleted. They can be found at different locations, depending on your install type and Operating system.

- **Windows (Installed):** `%AppData%\satdump`
- **Windows (Portable):** `<SatDump program folder>\config`

- **Linux and macOS** `~/.config/satdump`
- **Android:** `adb logcat -s SatDump`

Other Problems

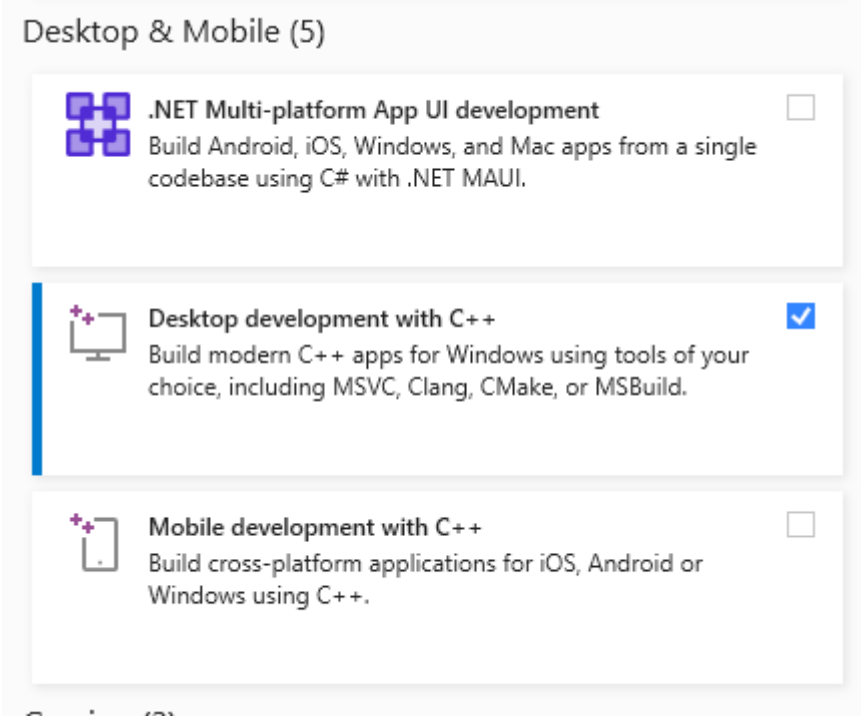
Have you encountered a problem not mentioned here? Take a look at our other docs, and if nothing seems to work, create an issue on GitHub or reach out to us on Matrix.

Building on Windows

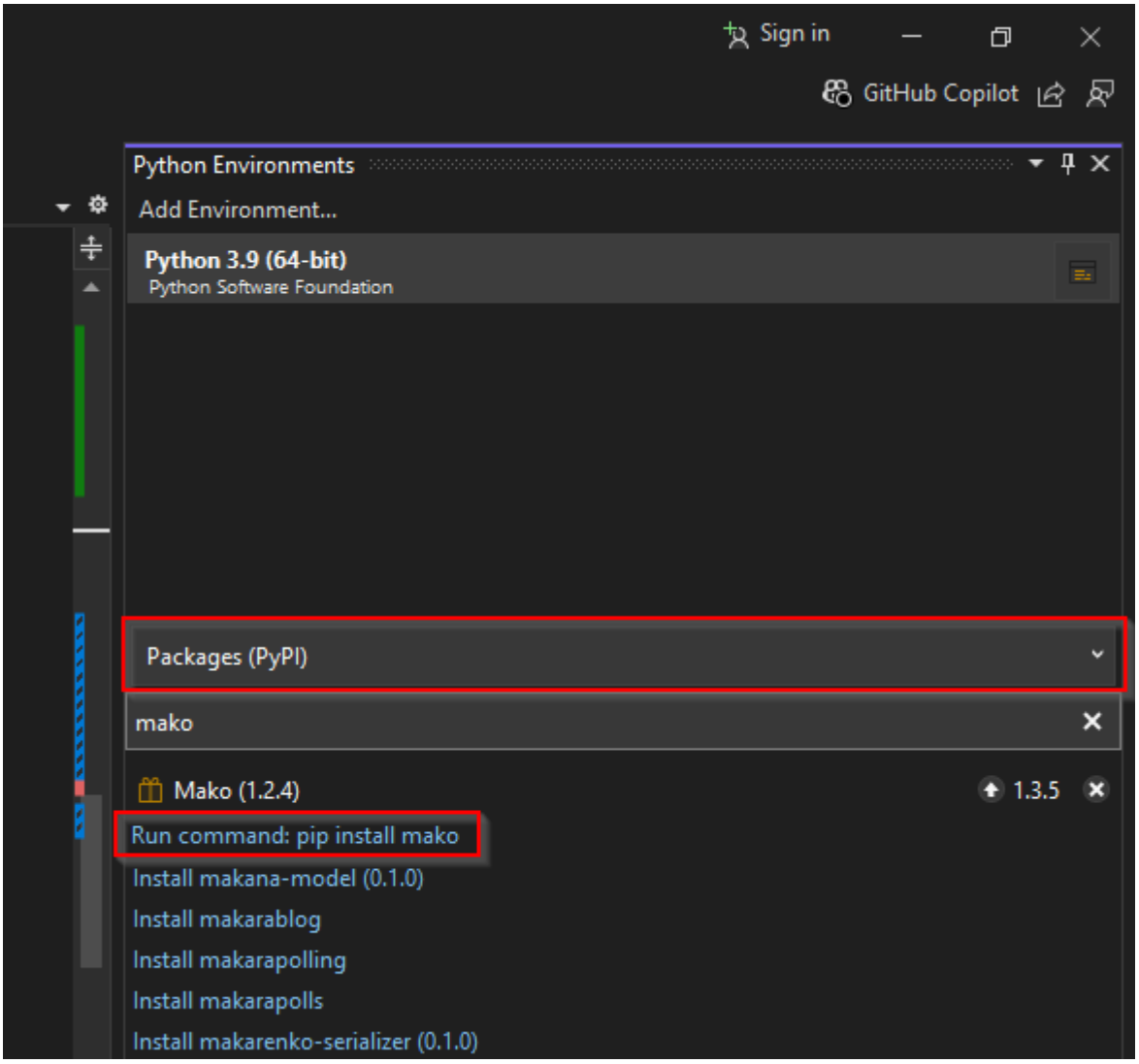
These instructions and scripts assume you will be using Microsoft Visual Studio 2022. Building SatDump on Windows requires Microsoft Visual Studio 2019 or greater. If you are using another version of Visual Studio, update `windows\Build.ps1` as necessary.

On install, set Visual Studio up for use with:

- Desktop development with C++
- Python Development Support (for building dependencies)



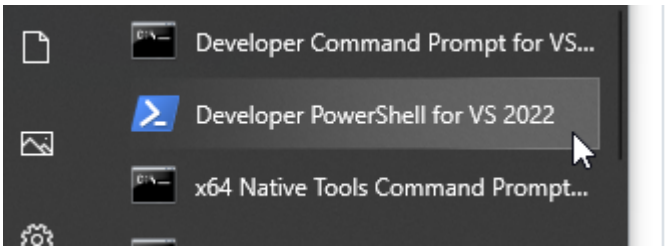
- Within Visual Studio, go to Tools > Python > Python Environments. Change the “Overview” dropdown to “Packages (PyPI)” and install mako (for building dependencies)



Automated PowerShell Build Instructions

Using the automated PowerShell build scripts is the easiest way to build SatDump on Windows.

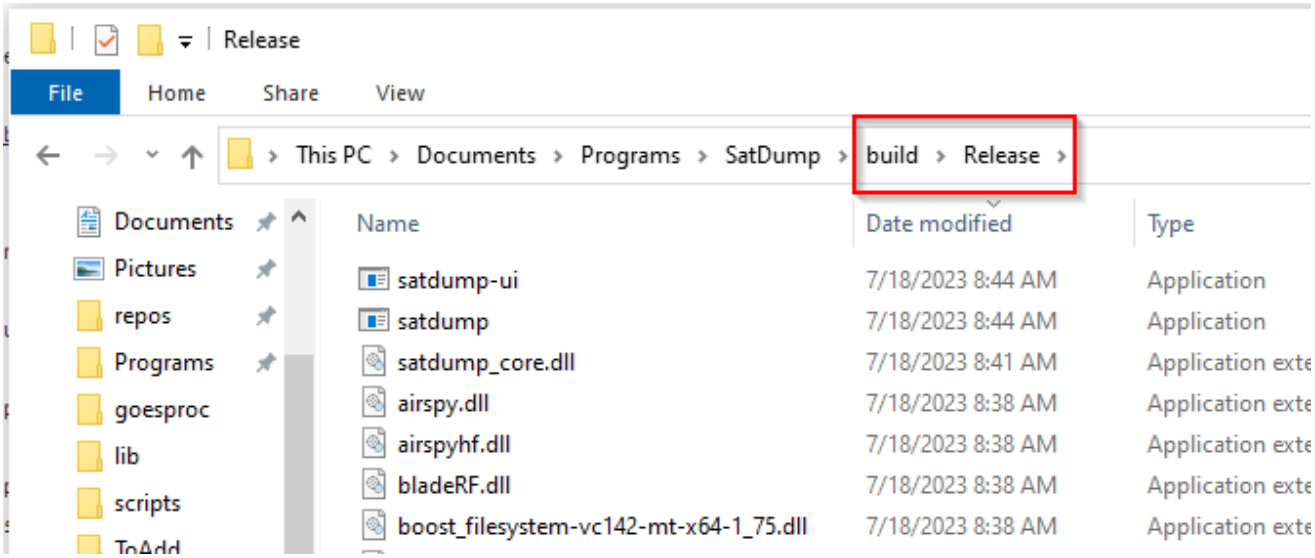
1. Open “Developer PowerShell for VS 2022” from the start Menu



2. cd to a directory where you want to build SatDump
3. Run the following commands, one at a time

```
powershell
git clone https://github.com/SatDump/SatDump
cd SatDump
.\windows\Configure-vcpkg.ps1 # Downloads vcpkg and configures it to build SatDump.
                              # This may take a while!
.\windows\Build-SatDump.ps1   # Builds SatDump
.\windows\Finish-Release.ps1  # Copies all necessary files into the Release folder
```

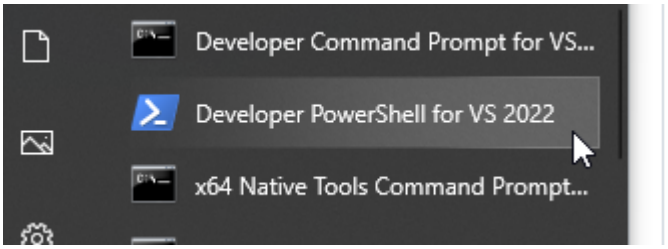
Your build SatDump will be in the `build\Release` folder of the SatDump repo.



Building in Visual Studio for Debugging

Compiling in Visual Studio for Debugging is for advanced users who aren't afriad to get their hands dirty!

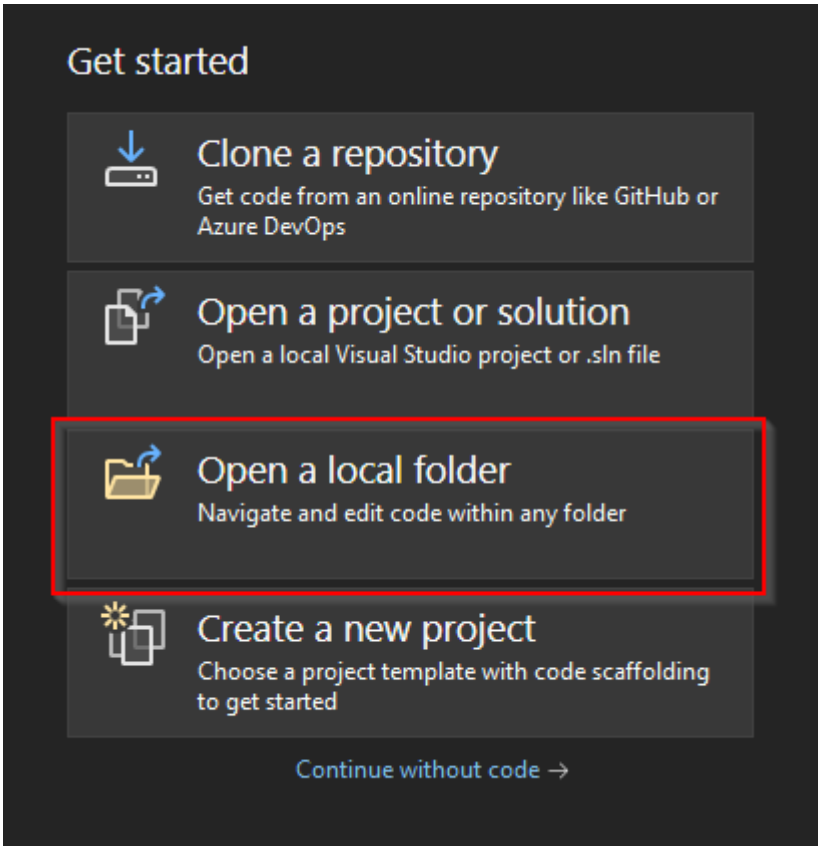
1. Open "Developer PowerShell for VS 2022" from the start Menu



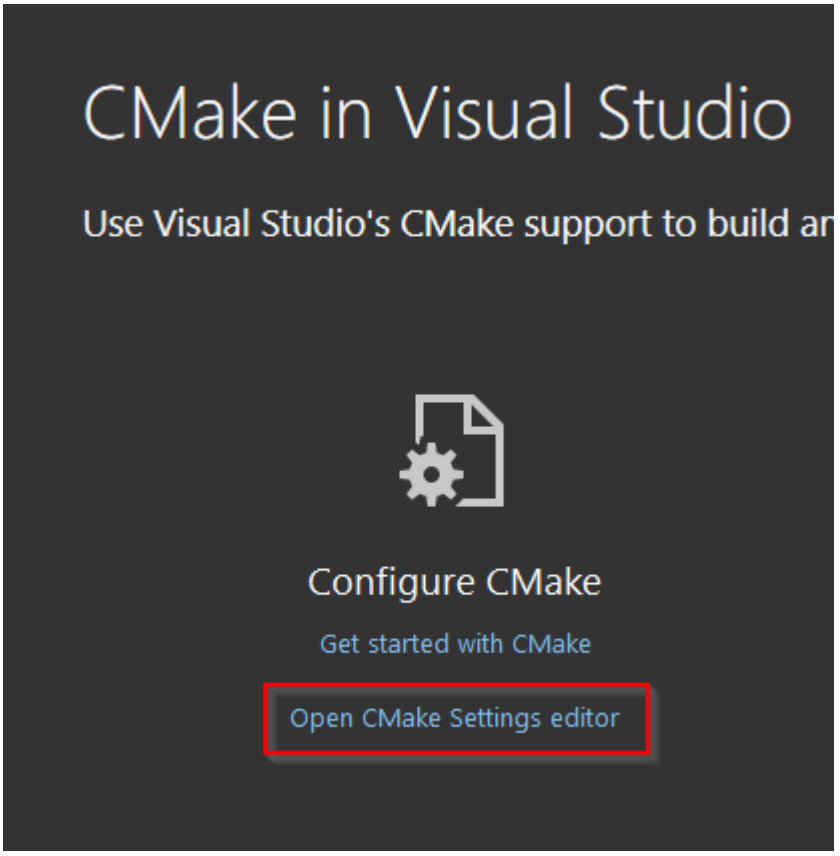
2. cd to a directory where you want to build SatDump
3. Run the following commands, one at a time

```
powershell
git clone https://github.com/SatDump/SatDump
cd SatDump
.\windows\Configure-vcpkg.ps1 # Downloads vcpkg and configures it to build SatDump.
                              # This may take a while!
```

4. Keeping the developer console open in the background, open Microsoft Visual Studio. On the launch screen, select "Open a local folder", and select your cloned SatDump repo.



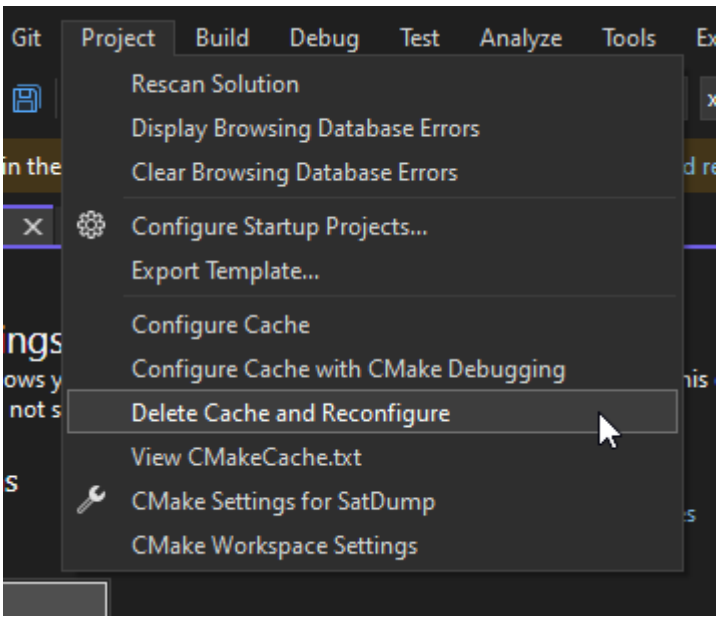
5. Once the project opens, select “Open CMake Settings Editor”



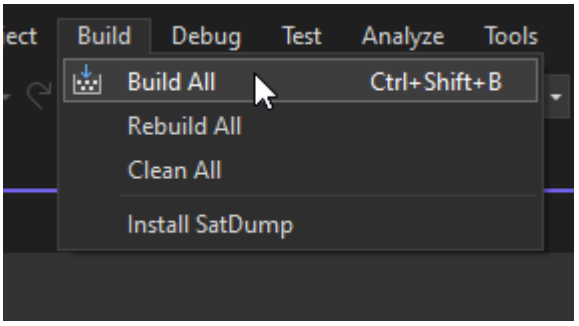
6. In CMake Settings, change the following items:

- **CMake toolchain file:** set this to `<full path of your satdump repo>\vcpkg\scripts\buildsystems\vcpkg.cmake`
- **CMake valiables and cache:** check “BUILD_MSVC”
- **CMake Generator (under “show advanced settings”):** set this to `Visual Studio 17 2022 Win64`

7. Press Ctrl+S to save the CMake Settings. You will usually get an error right away - this is normal. Either way, click Project > Delete Cache and Reconfigure. Confirm the prompt and let cmake rebuild its cache



8. Once cmake finishes, click Build > Build All. Go get come coffee because this takes several minutes

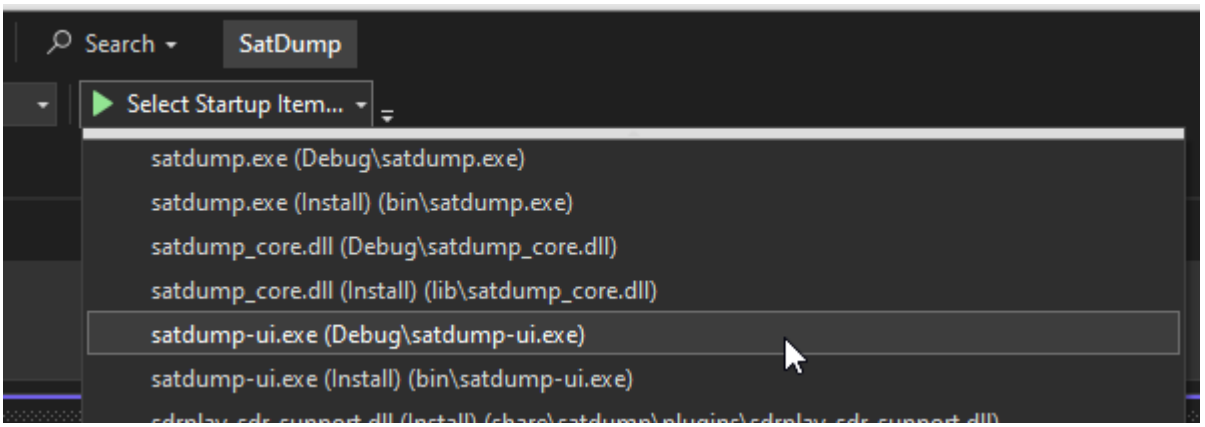


9. Once you get the “Build all succeded” message, go back to your developer terminal (it’s still open, right?) and run `.\windows\Finish-Debug.ps1`. This will copy all necessary DLL files into the debug folder.
10. That’s it, you’re ready to debug! To make any changes to the code and build again, just repeat steps 8 and 9 before launching the debugger. **Step 9 is important, because it copies the plugin DLLs into the right spot for SatDump to load them!**

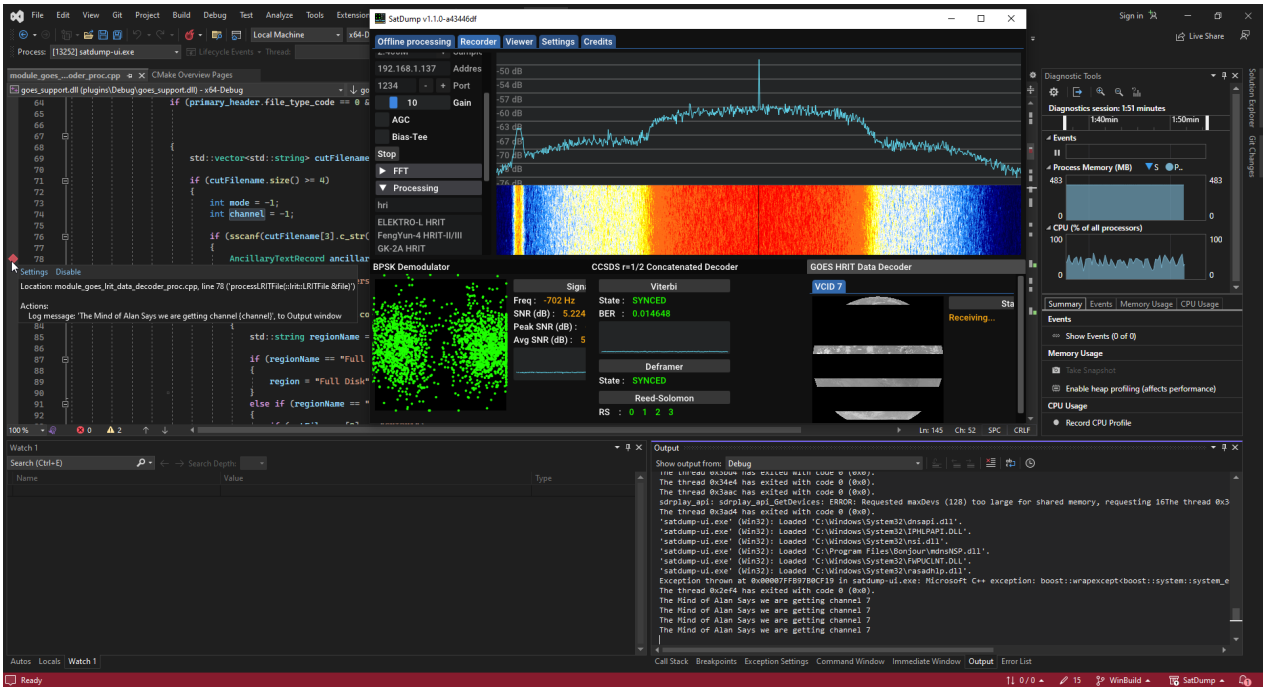
Debugging SatDump UI

Once you built SatDump for debugging in Windows, using the debugger is the same as it is with any other Visual Studio project.

1. At the top, set the startup item to “satdump-ui.exe” (not the installed one!)



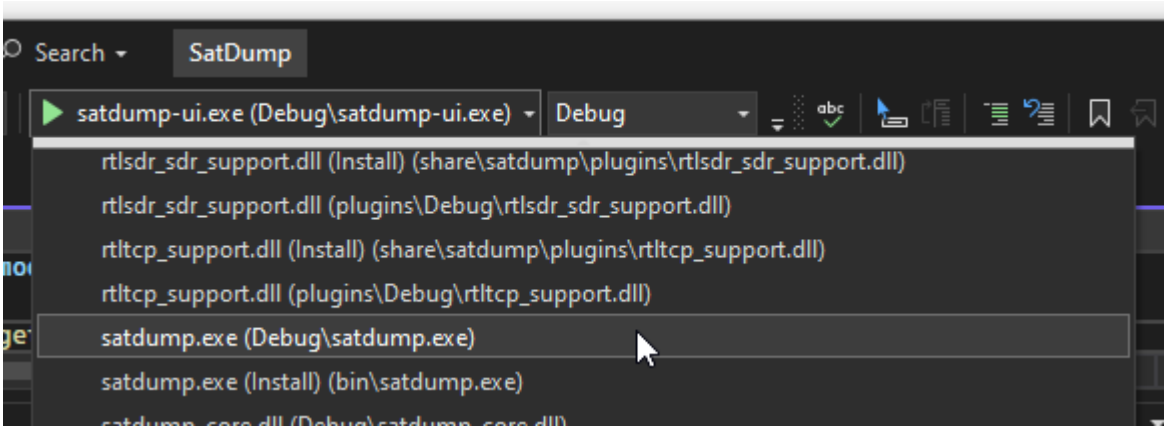
2. Click the green start icon to start debugging SatDump-UI. Breakpoints and other tested debugger functionality works as expected



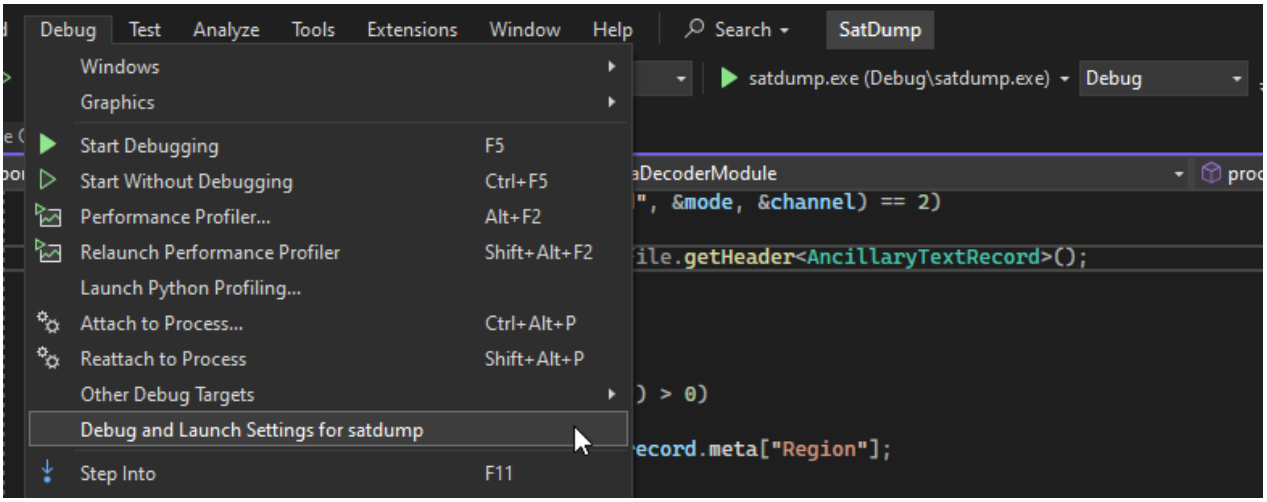
Debugging SatDump CLI

Debugging the CLI version is similar to the UI, but you need to provide command line arguments

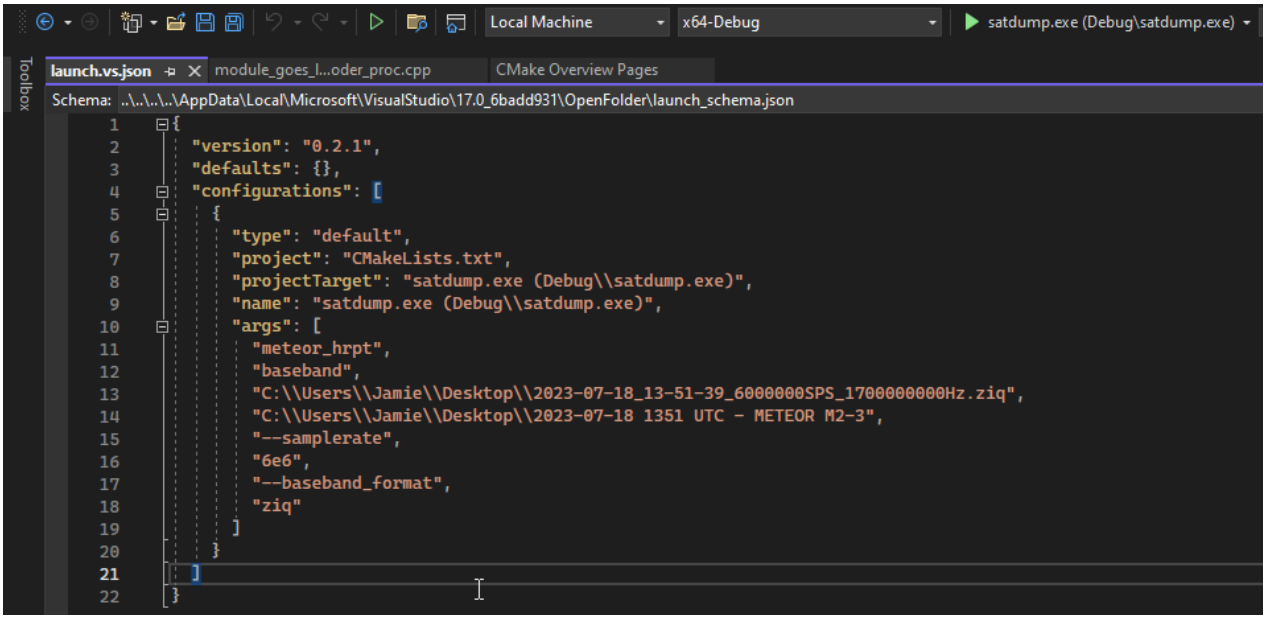
1. At the top, change the startup item to “satdump.exe”



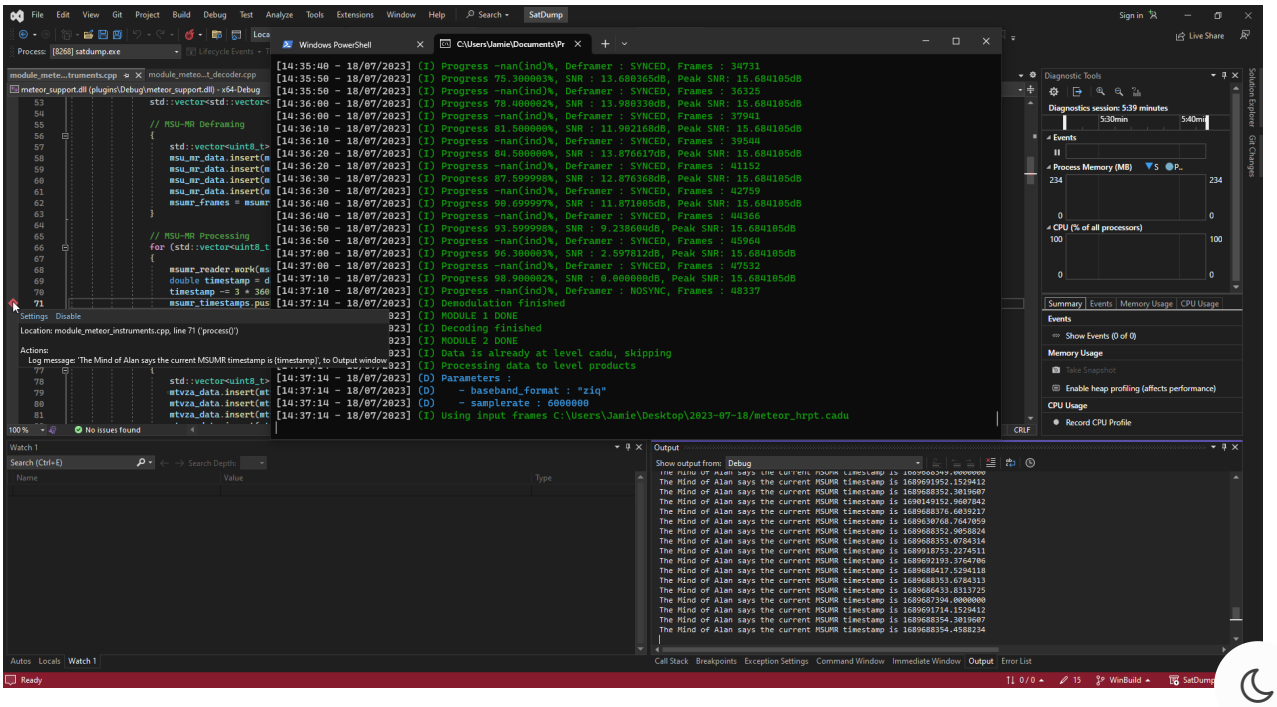
2. Go to Debug > Debug and Launch setting for SatDump



3. Add the satdump launch arguments in the form of an array. If you compose the argument list elsewhere and copy/paste it here, it will turn it into an array for you. You may need to clean it up, though



4. Click the green start icon to start debugging SatDump.



Scripting

SatDump supports Lua scripts that can automatically be triggered when a certain action has been performed.

Warning

This module is **NOT** built by default. You need to compile SatDump yourself and specify the `-DPLUGIN_SCRIPTING=ON` CMake option.

Basics

Scripts are simple Lua scripts. They do not have any function that needs to be called, the file will be executed as a normal Lua file by a regular Lua interpreter.

For example, consider a script containing the following:

```
print("Hello world")
```

This will be evaluated as a regular Lua script and return `Hello World` to standard output.

Note

`print` will always return to stdout. It is not currently possible to output in SatDump's logger.

Scripts need to be stored in SatDump's user script directory.

- Linux: `~/.config/satdump/scripts`
- Windows: `%appdata%\satdump\scripts`
- MacOS: `~/.config/satdump/scripts`

Scripts need to have a specific name in order to be called, which is detaied below.

pipeline_done_processing.lua

This script will be executed when the pipeline is done processing (i.e. when the satellite has

been decoded and processed).

A typical use case is to archive the frame files such `.cadu` or `.frm` in a specific directory.

The Lua script supports these variables:

- `pipeline_id`: the ID of the pipeline that has just finished procesing. See Satellite pipelines for a list.
- `pipeline_output_directory`: the full path of the output directory where the pipeline's results have been saved.

Note

For example, if you offline decoded Meteor-M N°2-3 into a directory called `/home/user/MeteorM`, `pipeline_id` will output `meteor_hrpt` and `pipeline_output_directory` will output `/home/user/MeteorM`.

Autotrack - Or automated satellite tracking

Introduction

Automated satellite tracking (usually referred to as “Autotrack”, for AUTOmated-TRACKing) in SatDump is meant to provide a solution to design (more or less) fully automated satellite reception stations. Such a system will usually consist of an antenna (either omnidirectional or directional), optionally a rotator, and a computer running SatDump with some SDR hardware. The design and assembly of such a system will not be described here as it is not within the scope of this documention.

SatDump will handle the *scheduling*, *tracking*, *live-processing/recording* and final *post-processing* of the data via different main components :

- The Scheduler
- The Satellite Tracker
- The Recorder / other signal processing

All of these features are available both in the GUI and CLI versions of SatDump.

Disclaimer : This is NOT a tutorial, rather a description of the capabilities and configuration options. If you wish to see a more specific tutorial there are better resources available.

The Scheduler

The scheduler is the core part of the autotrack system, its role is to predict upcoming satellite passes for the station, filter them to generate a realistic list of what the hardware can actually do and triggering other events to actually start recording/processing each pass.

Operating Modes

As outlined in the introduction, 2 type of antennas exist : omnidirectional and directional. In the case of a directional installation the antenna has to be steered toward the satellite during the pass. Meanwhile, an omnidirectional setup is technically capable of receiving any satellite currently above the horizon at the same time given they do not interfere with each other and the SDR hardware used has enough bandwidth to cover both frequencies at the same time. While it is possible to keep such a system operating in the more widespread “single-satellite” mode, for many common systems (such as a 137Mhz station) not being affected by passes overlapping is desirable. This is why such a capability is present in SatDump.

By default, SatDump will operate in single-satellite mode. In this case predicted satellite passes will be filtered as to eliminate or minimize overlap as the station can - obviously - only follow a single one at a time. For each, the frequency of the SDR will be set accordingly to the downlink frequency set in the scheduler's configuration, or if you set more than one the average of all frequencies will be set instead in an attempt to cover them all (this is meant to be used in case of satellites with several downlinks closeby in frequency) - if your SDR allows. Additionally the satellite tracker, which will both display the ongoing pass if in UI mode and control a rotator if setup will also be set upon AOS / LOS.

If **Multi Mode** is enabled, no pass filtering will be done at all. Instead, all passes will always be selected as it is left assumed the system is always capable of handling any visible satellite. It is also left up to the user to setup the SDR properly as the frequency or banwidth will never be changed. Therefore, the satellites and frequency configuration has to be set by the user to match what the hardware is capable of handling within the SDR's bandwidth and center frequency.

Note : It is planned to automate this process further later as to not make these assumptions and cover more usecases

CLI Configuration

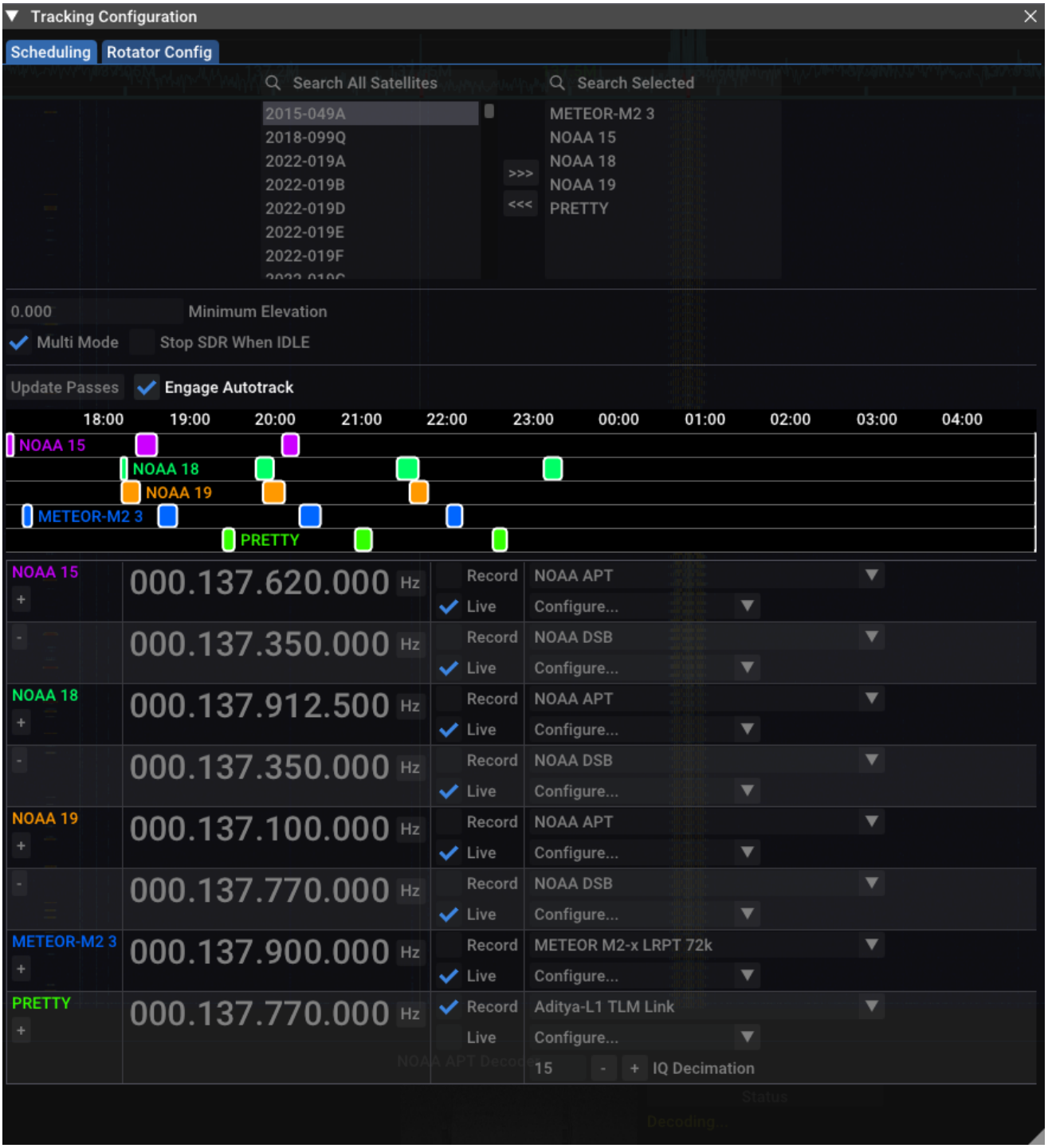
```
{
    // This is the general parameters block, holding SDR configuration options and a
few others
    "parameters": {
        // SDR Options
        "source": "rtlsdr", // The type of SDR Device you wish to use. See SDR Options
for more info
        "samplerate": 2.4e6, // The samplerate of the SDR device in use
        "initial_frequency": 137.5e6, // The initial frequency the SDR device will be
tuned at. If in Multi-Mode, that's also where it will be staying forever
        "gain": 49, // Other SDR options (see SDR Options), such as gain
        // FFT Options
        "fft_enable": true, // Whether the FFT in the WebUI should be enabled or not -
enabling can lead to higher CPU usage, but it will be disabled dynamically if unused
        "fft_size": 65536, // The size of the FFT to use - 1:1 identical to the GUI
        "fft_rate": 30, // The rate at which to run the FFT - 1:1 identical to the GUI
        "fft_avgn": 1 // Number of FFTs to average
    },
    "finish_processing": true, // Enables post-processing of the data. Eg, decoding the
.cadu or .frm down to images
    "output_folder": "./test_auto_cli", // Output folder, where all basebands, decoded
data, etc will be present
    // QTH Configuration. In degrees and meters ASL (Above Sea Level)
    "qth": {
        "lat": 50, // Latitude
        "lon": -70, // Altitude
        "alt": 123 // Altitude
    },
    // If present, enables the integrated HTTP server with a Web interface showing
various information
    // as well as an API (JSON) server to pull information about the autotrack status.
    "http_server": "0.0.0.0:8081",
    // Tracking configuration. This covers the "Satellite Tracker" part
    "tracking": {
        // Rotator configuration. This can be ommited if not required
        "rotator_config": {
            // For now, only Hamlib's Rotctl is supported
            "rotctl": {
                "address": "127.0.0.1", // Address of the rotctld server to connect to
                "port": 4533 // Port of the server to connect to
            }
        },
        // Configuration of the rotator tracking. This can be ommited if not required
        "rotator_algo": {
            // This section is also optional, and sets the rotator to
            // park itself to a specific position when no satellites are in sight,
            // for example setting a prime-focus dish to Zenith in order to
            // reduce wind load
            "park_while_idle": true, // Enables this feature
            "park_position": {
                "az": 0.0, // Azimuth to set the rotator to when IDLE
                "el": 90.0 // Elevation to set the rotator to when IDLE
            },
            "unpark_at_minus": 60.0, // How manys seconds before a pass the rotator
should un-park at prior to a pass
            // Other general parameters
            "update_period": 0.1 // How often the rotator position should be updated,
in seconds
        },
        // Configuration of the scheduler
        "autotrack_cfg": {
            "autotrack_min_elevation": 0, // Minimum elevation (at TCA!) of passes to
select, for all satellites.
```

```
            "stop_sdr_when_idle": false, // Whether to stop the SDR when no satellites
are in sight
            "multi_mode": true // Puts the scheduler in "Multi" mode - meant for
omnidirectional setups
        }
    },
    // This is the list and configuration of objects autotrack should be following.
    // Do note everything except *norad* and for *downlinks*, *frequency*, *record*
    // and *live* is fully optional.
    "tracked_objects": [
        {
            "norad": 25338, // NORAD ID of the satellite to track
            // Enumeration of downlinks to record. Usually one, but can be more than
one
            "downlinks": [
                {
                    "frequency": 137620000, // Frequency of the downlink, in Hz
                    "record": false, // Whether to record baseband or not
                    "live": true, // Enables live-processing with a SatDump pipeline
                    "pipeline_name": "noaa_apt", // ID of the pipeline, to be found in
the "pipelines/*.json" files
                    // Parameters fo the pipeline, which vary depending on the pipeline
                    "pipeline_params": {
                        "autocrop_wedges": false,
                        "satellite_number": "15",
                        "sdrpp_noise_reduction": true
                    }
                },
                {
                    "frequency": 137350000,
                    "record": false,
                    "live": true,
                    "pipeline_name": "noaa_dsb",
                    "pipeline_params": {} // Many pipelines will not require any
parameters, this can be left empty safely
                }
            ]
        },
        {
            "norad": 58023,
            "downlinks": [
                {
                    "frequency": 137.77e6, // You can also use exponent notation!
                    "record": true,
                    "baseband_format": "ziq2", // Baseband format to record in
                    "baseband_decimation": 6 // Decimation to be applied to the
recording, this effectively divides the samplerate of the SDR
                }
            ]
        }
    ]
}
```

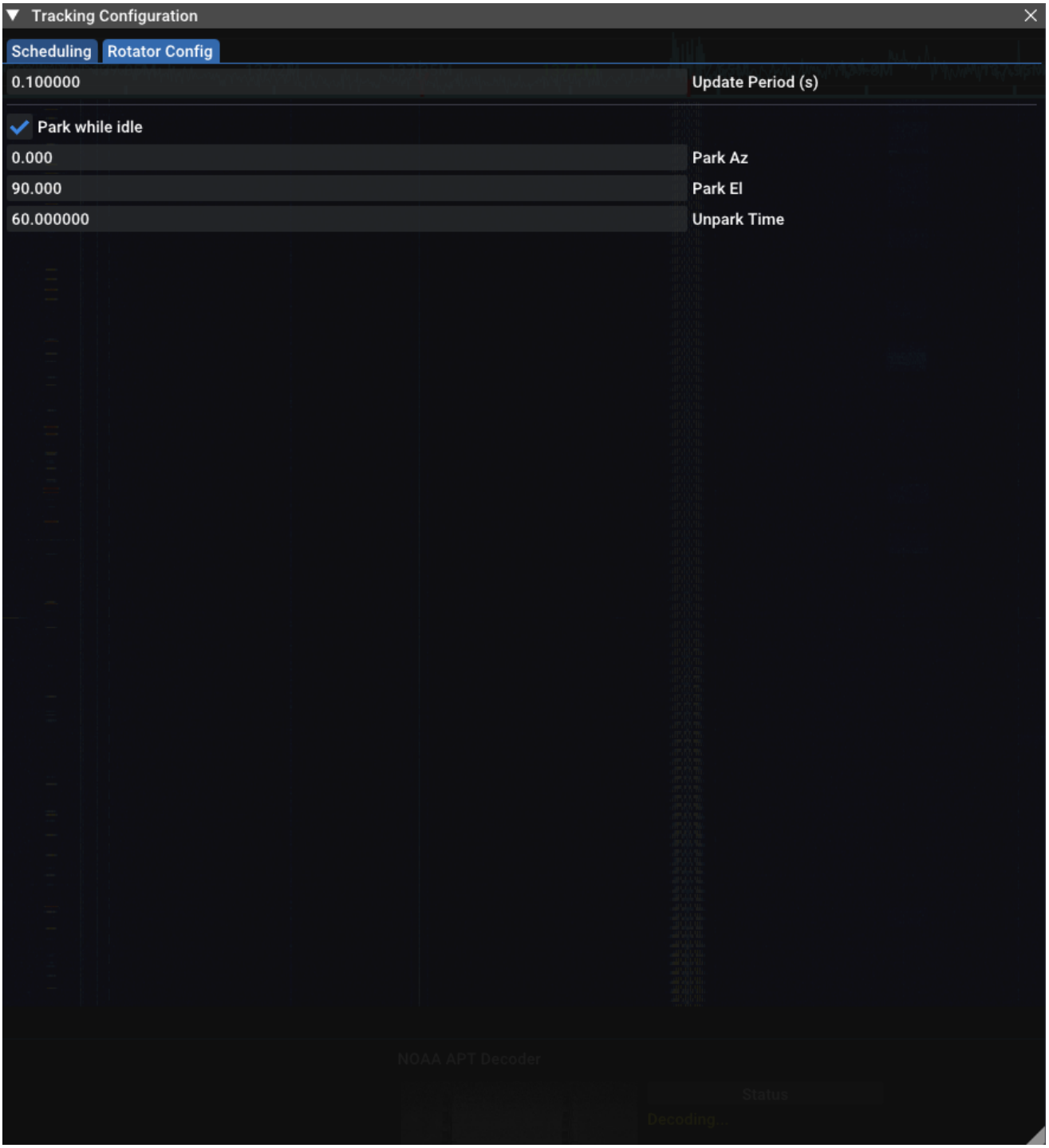
GUI Configuration

Most of the GUI configuration is covered in the *Satellite Tracker* section of this documentation. Please refer to it for rotator configuration, setting your QTH and similar.

Otherwise, everything will be done the following 2 menus :



- **Satellite Selector :** This is the menu in the top part of the window. By selecting a satellite in either and clicking the buttons in the center you can select the satellites you wish to track. The right panels is those currently selected.
- **Minimum Elevation :** This filters passes to be above a specific minimum elevation at TCA
- **Multi Mode :** Puts the scheduler in multi-satellite mode, for omnidirectional installations
- **Stop SDR When IDLE :** Stops the SDR when no satellites are being tracked
- **Timetable :** The “timetable” graph shows upcoming passes for each satellites, and highlights those selected in white when Autotrack is engaged
- **Update Passes :** Refreshes upcoming passes now. This is done automatically if engaged
- **Engage Autotrack :** Enables/Disables autotrack. Configuration is locked out when enabled
- **Satellite List :** Lets the user configure the satellites being tracked. The frequency can be set in the 2nd column, baseband recording/live-processing can be set in the 3rd column and furhter configuration in the last. The “+” and “-” buttons in the left column let you add/delete downlinks if more than one frequency is needed per satellite


Tracking Configuration
Scheduling
Rotator Config
0.100000
Update Period (s)
Park while idle
0.000
Park Az
90.000
Park El
60.000000
Unpark Time
NOAA APT Decoder
Status
Decoding...