
gr-satellites Documentation

Release 3.4.0

Daniel Estévez

Oct 08, 2020

Contents:

1	Introduction	3
2	Installing from source	5
2.1	Dependencies	5
2.2	Optional dependencies	6
2.3	Downloading	6
2.4	Building and installing	6
2.5	PYTHONPATH	6
2.6	Downloading sample recordings	7
3	Installing using conda	9
3.1	Miniconda	9
3.2	GNU Radio	10
3.3	gr-satellites	10
3.4	Acknowledgments	10
4	Overview	11
4.1	Command line tool	11
4.2	Satellite decoder block	11
4.3	Components	12
4.4	Low level blocks	12
4.5	Utilities	12
5	gr_satellites command line tool	13
5.1	Basic usage	13
5.2	Ouput options	18
5.3	Telemetry submission	20
5.4	File and image receiver	22
5.5	Other topics	23
6	Satellite decoder block	25
6.1	Command line options	26
7	Components	27
7.1	Data sources	27
7.2	Demodulators	28
7.3	Deframers	32

7.4	Transports	36
7.5	Data sinks	38
8	SatYAML files	43
9	Low level blocks	49
10	Miscellaneous utilities	51
10.1	JY1SAT SSDV decoder	51
10.2	SMOG-P spectrum plot	51
11	Supported satellites	53
12	Indices and tables	71

gr-satellites is a GNU Radio out-of-tree module with a collection of telemetry decoders for Amateur satellites.

CHAPTER 1

Introduction

`gr-satellites` is a [GNU Radio](#) out-of-tree module encompassing a collection of telemetry decoders that supports many different [Amateur satellites](#). This open-source project started in 2015 with the goal of providing telemetry decoders for all the satellites that transmit on the [Amateur radio bands](#).

It supports most popular protocols, such as [AX.25](#), the GOMspace NanoCom U482C and [AX100](#) modems, an important part of the [CCSDS stack](#), the [AO-40 protocol](#) used in the [FUNcube](#) satellites, and several ad-hoc protocols used in other satellites.

This out-of-tree module can be used to decode frames transmitted from most Amateur satellites in orbit, performing demodulation, forward error correction, etc. Decoded frames can be saved to a file or displayed in hex format. For some satellites the telemetry format definition is included in `gr-satellites`, so the decoded telemetry frames can be printed out as human-readable values such as bus voltages and currents. Additionally, some satellites transmit files such as JPEG images. `gr-satellites` can be used to reassemble these files and even display the images in real-time as they are being received.

`gr-satellites` can be used as a set of building blocks to implement decoders for other satellites or other groundstation solutions. Some of the low level blocks in `gr-satellites` are also useful for other kinds of RF communications protocols.

Installing from source

gr-satellites is a GNU Radio out-of-tree module, and can be installed as such, by building it from source in a system where GNU Radio is already installed. Alternatively, it is possible to *install gr-satellites and GNU Radio*, and this might provide an easier or quicker way of installation, especially in Linux distributions where GNU Radio is not so easy to install, or in macOS.

The general steps for installing gr-satellites from source include making sure that all the dependencies are installed and then building and installing the out-of-tree module.

2.1 Dependencies

gr-satellites requires GNU Radio version at least 3.8.

Warning: There are some build dependencies for GNU Radio out-of-tree modules that are not required to run GNU Radio, so some distributions might not install them by default when GNU Radio is installed. The main ones that may cause problems are:

- swig
- liborc (in Debian-based distributions `liborc-0.4-dev` is needed)

Additionally, the following libraries are required:

- `construct`, at least version 2.9.
- `requests`

Note: `construct` and `requests` are Python packages and can be installed with `pip` by doing

```
$ pip3 install --user --upgrade construct requests
```

Alternatively, `construct` and `requests` can be installed from your distribution's package manager

2.2 Optional dependencies

To use the realtime image decoders, gr-satellites needs `feh`

Note: `feh` is best installed through your distribution's package manager

2.3 Downloading

gr-satellites is developed in the [daniestevez/gr-satellites](#) Github repository. It is recommended that you download the [latest stable release](#). You can also browse the list of [all releases](#) to see older versions and pre-releases.

Users interested in collaborating with testing or developing gr-satellites can clone the git repository and use the master branch. There is more information about the organization in branches in the [README](#).

2.4 Building and installing

gr-satellites can be built and installed using `cmake`. The following can be run inside the directory containing the gr-satellites sources:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
$ sudo ldconfig
```

After running `make`, you can run the tests by doing `make test` in the `build/` directory.

Note: There are systems where the AO-73 and similar decoders fail if `volk_profile` has not been run ever in the system. This seems to be caused by the Viterbi decoder chosen by Volk by default when there is no `~/ .volk/ volk_config` file. If problems with these decoders are seen, it is recommended to run `volk_profile` to see if it fixes the problems.

2.5 PYTHONPATH

After installing gr-satellites, it is necessary to ensure that Python is able to locate the gr-satellites Python module. Depending on the configuration of Python and the location where gr-satellites has been installed, it might be necessary to set the `PYTHONPATH` environment variable.

If Python is not able to locate the gr-satellites module, it will produce an error like this:

```
ModuleNotFoundError: No module named 'satellites'
```

Often, gr-satellites is installed into `/usr/local/lib/python3/dist-packages/` or a similar directory, in a subdirectory called `satellites`. Therefore,

```
$ export PYTHONPATH=/usr/local/lib/python3/dist-packages/
```

can be used to allow Python to find the gr-satellites module. More information about the PYTHONPATH can be found in Python's documentation description of the [PYTHONPATH](#).

Note: A permanent configuration of the PYTHONPATH can be added to a script such as `~/.bashrc` or `~/.bash_profile`. This applies the correct PYTHONPATH when `gr_satellites` or `gnuradio-companion` are run from a bash session. If `gnuradio-companion` is run directly from the graphical environment, then it is necessary to set the PYTHONPATH in `xinitrc` or `xprofile`. See the [Arch Linux documentation on environment variables](#) for more information.

2.6 Downloading sample recordings

The `satellite-recordings/` directory is a [git submodule](#) that contains many short sample recordings of different satellites that can be used to test the decoders. From a clone of the gr-satellites git repository, the submodule can be cloned (downloaded) by running

```
$ git submodule update --init
```

inside the `gr-satellites/` directory.

Alternatively, it is possible to run

```
$ git clone --recursive https://github.com/daniestevez/gr-satellites
```

when cloning the gr-satellites repository to download both gr-satellites and the satellite-recordings submodule.

The satellite-recordings sample recordings can also be downloaded from its [own git repository](#), which is necessary if gr-satellite has not been installed from the git repository.

Installing using conda

gr-satellites and GNU Radio can also be installed using `conda`, in Linux, macOS, and Windows. Conda is a multiplatform open-source package management system that can install packages and their dependencies in different virtual environments, independently from the rest of the packets installed in the OS. Using conda is the easiest and recommended way of installing gr-satellites in Windows.

This section shows how to install `miniconda`, GNU Radio, and gr-satellites from scratch.

3.1 Miniconda

`Miniconda` is a minimal installer for conda, so it is the recommended way to get GNU Radio and gr-satellites quickly running in an OS that does not have conda already installed. Miniconda can be installed by downloading and running the installer for the appropriate platform from Miniconda's page. The installer can be run as a regular user. It does not need root access.

After installing Miniconda, its `(base)` virtual environment will be active by default. This means that `(base)` will be shown at the beginning of the command line prompt and software will be run from the version installed in the `(base)` virtual environment (when it is installed), and otherwise from the OS.

Users might prefer to run things from the conda virtual environment only upon request. To disable the activation of the `(base)` environment by default, we can run

```
$ conda config --set auto_activate_base false
```

When the `(base)` environment is not enabled by default, we can enter it by running

```
$ conda activate base
```

and exit it by running

```
$ conda deactivate
```

When the (base) environment is activated, the prompt will start by (base). The (base) environment needs to be activated in order to install applications through conda into this environment, and also to run applications that have been previously installed in this environment.

3.2 GNU Radio

To install GNU Radio, the (base) environment (or another conda virtual environment) needs to be activated as described above. Installing GNU Radio and all its dependencies is as simple as doing

```
$ conda install -c conda-forge gnuradio
```

Then GNU Radio may be used normally whenever the virtual environment where it was installed is activated. For instance, it is possible to run

```
$ gnuradio-companion
```

3.3 gr-satellites

gr-satellites needs to be installed into a virtual environment where GNU Radio has been previously installed (the (base) environment, if following the instructions here). To install gr-satellites and its dependencies, we do

```
$ conda install -c conda-forge gnuradio-satellites
```

After installation, the `gr_satellites` command line tool might be run as

```
$ gr_satellites
```

(provided that the virtual environment where it was installed is activated) and blocks from gr-satellites may be used in GNU Radio companion.

It might be convenient to download the *sample recordings* manually.

3.4 Acknowledgments

Thanks to [Ryan Volz](#) and [Petrus Hyvönen](#) for their work in putting together recipes to install gr-satellites and its dependencies through Conda and for helping me make gr-satellites build on Windows.

gr-satellites can be used in different ways depending on the experience of the user and the desired degree of customization. This section gives an overview of the possibilities.

4.1 Command line tool

The main way of using gr-satellites is through the `gr_satellites` command line tool. This allows users to decode the satellites officially supported by gr-satellites by using a command line tool. Decoding options can be specified as command line parameters. The tool supports processing both RF samples streamed in real-time from an SDR receiver or a conventional radio connected to the computer's soundcard, and recordings in different formats.

The command line tool is the most simple way of using gr-satellites and so it is recommended as the starting point for beginners. The usage of this tool is described in depth in the *gr_satellites command line tool* section.

4.2 Satellite decoder block

The Satellite decoder block gives most of the functionality of the `gr_satellites` command line tool encapsulated as a GNU Radio block.



Fig. 1: Satellite decoder GNU Radio block

It can be included in any kind of [GNU Radio Companion](#) flowgraphs and it can be used to achieve a greater degree of customization and flexibility than what is possible with the command line tool.

The input to the Satellite decoder block may be pre-processed freely using GNU Radio blocks. The decoded frames are output as PDUs and can be handled in any manner by the user's flowgraph.

The Satellite decoder block is recommended for users who are already familiar with the command line tool and want a higher degree of customization. Its usage is described in the *Satellite decoder block* section.

4.3 Components

Components are the high level blocks in which gr-satellites is structured. Briefly speaking, the decoding chain is split into different tasks and the `gr_satellites` command line tool and Satellite decoder block perform decoding by selecting and connecting the appropriate components for each of these tasks depending on the satellite selected by the user.

The architecture of components is described in more detail in the *Components* section. They can be used as GNU Radio Companion blocks to customize decoders further than what is allowed by the Satellite decoder block. Additionally, they can be useful to build receivers for other RF communication protocols.

Users interesting in learning how the decoding process works or in adding new decoders to gr-satellites should be familiar with components.

4.4 Low level blocks

Finally, gr-satellites has a large number of lower level GNU Radio Companion blocks that may be useful in many different situations. Usage of these low level blocks is recommended only for users already familiar with gr-satellites or GNU Radio.

4.5 Utilities

Besides the `gr_satellites` command line tool and the GNU Radio blocks, gr-satellites also contains a few *Miscellaneous utilities* that can be used with some of the satellites.

gr_satellites command line tool

The `gr_satellites` command line tool is a complete solution that can decode frames using either real-time RF samples from an SDR or conventional radio, or a recording.

5.1 Basic usage

`gr_satellites` can be run from a terminal after `gr-satellites` has been installed. If run without any arguments, `gr_satellites` will only print some basic information about the arguments it allows.

```
$ gr_satellites
usage: gr_satellites satellite [-h] [--version]
                                (--wavfile WAVFILE | --rawfile RAWFILE | --rawint16_
↳RAWINT16 | --audio [DEVICE] | --udp | --kiss_in KISS_IN)
                                [--samp_rate SAMP_RATE] [--udp_ip UDP_IP]
                                [--udp_port UDP_PORT] [--iq]
                                [--input_gain INPUT_GAIN] [--kiss_out KISS_OUT]
                                [--kiss_append] [--hexdump] [--kiss_server [PORT]]
                                [--kiss_server_address KISS_SERVER_ADDRESS]
                                [--zmq_pub [ADDRESS]] [--hexdump]
                                [--dump_path DUMP_PATH]
```

5.1.1 Specifying the satellite

The arguments that `gr_satellites` allows depend on the satellite that has been selected. Therefore, to use `gr_satellites` it is always necessary to specify the `satellite` to be used as an argument immediately following `gr_satellites`. There are three different ways to specify the satellite:

- Using the satellite name, such as *FUNcube-1* or *LilacSat-2*. This can be used with any *satellite officially supported by gr-satellites*, and it is the most simple way of specifying a satellite.

```

$ gr_satellites FUNcube-1
usage: gr_satellites satellite [-h] [--version]
      (--wavfile WAVFILE | --rawfile RAWFILE | --rawint16_
↳RAWINT16 | --audio [DEVICE] | --udp | --kiss_in KISS_IN)
      [--samp_rate SAMP_RATE] [--udp_ip UDP_IP]
      [--udp_port UDP_PORT] [--iq]
      [--input_gain INPUT_GAIN] [--kiss_out KISS_OUT]
      [--kiss_append] [--kiss_server [PORT]]
      [--kiss_server_address KISS_SERVER_ADDRESS]
      [--zmq_pub [ADDRESS]] [--hexdump]
      [--dump_path DUMP_PATH]
      [--telemetry_output TELEMETRY_OUTPUT]
      [--f_offset F_OFFSET] [--rrc_alpha RRC_ALPHA]
      [--disable_fll] [--fll_bw FLL_BW]
      [--clk_bw CLK_BW] [--clk_limit CLK_LIMIT]
      [--costas_bw COSTAS_BW]
      [--manchester_history MANCHESTER_HISTORY]
      [--syncword_threshold SYNCWORD_THRESHOLD]
      [--verbose_rs]
gr_satellites satellite: error: one of the arguments --wavfile --rawfile --
↳rawint16 --audio --udp --kiss_in is required

```

A satellite may have several different names, known as *alternative names*. For example, FUNcube-1 is both known as AO-73 and FUNcube-1.

- Using the satellite **NORAD ID**. This can be used with any *satellite officially supported by gr-satellites*, and it can be useful when interfacing gr_satellites with other tools that use NORAD IDs to classify satellites.

Below we show gr_satellites running with NORAD ID 39444, which corresponds to FUNcube-1.

```

$ gr_satellites 39444
usage: gr_satellites satellite [-h] [--version]
      (--wavfile WAVFILE | --rawfile RAWFILE | --rawint16_
↳RAWINT16 | --audio [DEVICE] | --udp | --kiss_in KISS_IN)
      [--samp_rate SAMP_RATE] [--udp_ip UDP_IP]
      [--udp_port UDP_PORT] [--iq]
      [--input_gain INPUT_GAIN] [--kiss_out KISS_OUT]
      [--kiss_append] [--kiss_server [PORT]]
      [--kiss_server_address KISS_SERVER_ADDRESS]
      [--zmq_pub [ADDRESS]] [--hexdump]
      [--dump_path DUMP_PATH]
      [--telemetry_output TELEMETRY_OUTPUT]
      [--f_offset F_OFFSET] [--rrc_alpha RRC_ALPHA]
      [--disable_fll] [--fll_bw FLL_BW]
      [--clk_bw CLK_BW] [--clk_limit CLK_LIMIT]
      [--costas_bw COSTAS_BW]
      [--manchester_history MANCHESTER_HISTORY]
      [--syncword_threshold SYNCWORD_THRESHOLD]
      [--verbose_rs]
gr_satellites satellite: error: one of the arguments --wavfile --rawfile --
↳rawint16 --audio --udp --kiss_in is required

```

- Using a path to an .yaml SatYAML file. SatYAML files are used by gr-satellites to specify the decoding parameters and configuration corresponding to each different satellite. They are described in more detail in the *SatYAML files* section.

gr-satellites comes bundled with a large number of SatYAML files corresponding to all the officially supported satellites. They can be found in the python/satyaml/ directory.

Specifying the path of a SatYAML file is useful if the user has modified some of the files bundled with gr-satellites or has created their own ones.

```
$ gr_satellites python/satyaml/AO-73.yml
usage: gr_satellites satellite [-h] [--version]
                                (--wavfile WAVFILE | --rawfile RAWFILE | --
↪rawint16 RAWINT16 | --audio [DEVICE] | --udp | --kiss_in KISS_IN)
                                [--samp_rate SAMP_RATE] [--udp_ip UDP_IP]
                                [--udp_port UDP_PORT] [--iq]
                                [--input_gain INPUT_GAIN] [--kiss_out KISS_OUT]
                                [--kiss_append] [--kiss_server [PORT]]
                                [--kiss_server_address KISS_SERVER_ADDRESS]
                                [--zmq_pub [ADDRESS]] [--hexdump]
                                [--dump_path DUMP_PATH]
                                [--telemetry_output TELEMETRY_OUTPUT]
                                [--f_offset F_OFFSET] [--rrc_alpha RRC_ALPHA]
                                [--disable_fll] [--fll_bw FLL_BW]
                                [--clk_bw CLK_BW] [--clk_limit CLK_LIMIT]
                                [--costas_bw COSTAS_BW]
                                [--manchester_history MANCHESTER_HISTORY]
                                [--syncword_threshold SYNCWORD_THRESHOLD]
                                [--verbose_rs]
gr_satellites satellite: error: one of the arguments --wavfile --rawfile -
↪rawint16 --audio --udp --kiss_in is required
```

5.1.2 Specifying the input source

Besides specifying the satellite to use for decoding, it is mandatory to specify the input source by using exactly one of the following options:

- `--wavfile` can be used to read a recording in WAV format. The sample rate of the recording needs to be specified with the `--samp_rate` argument.

By default, the WAV file is interpreted as a one-channel file containing real RF samples. To read a two-channel file containing IQ RF samples, the `--iq` argument needs to be specified.

Note: All the *sample recordings* in the `satellite-recordings/` are real 48kHz WAV files and can be read with the `--wavfile file --samp_rate 48e3` arguments.

For example, this will decode some frames from FUNcube-1:

```
$ gr_satellites FUNcube-1 --wavfile satellite-recordings/ao73.wav --samp_rate 48e3
```

- `--rawfile` can be used to read a recording in `complex64` or `float32` format (depending on whether the `--iq` argument is used or not). The sample rate of the recording needs to be specified with the `--samp_rate` argument.

Note: Files in `complex64` format contain a sequence of 32-bit floating point numbers in IEEE 754 format. The sequence alternates between the I (in-phase) and Q (quadrature) components of a stream of IQ samples. This format is used by the GNU Radio File Source and File Sink blocks when their type is set to *complex*.

Files in `float32` format contain a sequence of 32-bit floating point numbers in IEEE 754 format. The sequence contains the elements of a stream of real samples. This format is used by the GNU Radio File Source and File Sink blocks when their type is set to *float*.

- `--rawint16` can be used to read a recording in `int16` format. The file is interpreted as IQ or real data according as to whether the `--iq` argument is used or not. The sample rate of the recording needs to be specified with the `--samp_rate` argument.

Note: Files in `int16` format contain a sequence of 16-bit integers in host endianness. This format is used by GNU Radio File Source and File Sink blocks when their type is set to *short*.

- `--audio` can be used to read samples from the soundcard, using GNU Radio's [Audio Source](#). This can be used to receive audio from a conventional radio by using the soundcard or from another application via a "virtual audio cable".

The sample rate to use needs to be specified with the `--samp_rate` argument. A sample rate of 48000 is typical with audio devices.

Both real samples (by default) and IQ samples (using the `--iq` argument) are supported. IQ samples use two audio channels (stereo).

The `--audio` argument can optionally be followed by the name of the audio device to use. Details about how to specify the device name vary between platform and are described in the [Audio Source](#) documentation. If no device name is entered, the default audio device will be chosen.

- `--udp` can be used to received RF samples streamed in real-time. The sample rate of the recording needs to be specified with the `--samp_rate` argument.

The streaming format is the same as for the `--rawint16` and both real samples (by default) and IQ samples (using the `--iq` argument) are supported.

By default, `gr_satellites` will listen on the IP address `:::` (all addresses) and the UDP port 7355. A different IP address or port can be specified using the parameters `--udp_ip` and `--udp_port`.

Note: [GQRX](#) can stream audio in UDP using this format and UDP port, and a sample rate of 48ksps by following the instructions [here](#). In this case, `gr_satellites` should be run as

```
$ gr_satellites FUNcube-1 --udp --samp_rate 48e3
```

This is recommended as a simple way of interfacing `gr_satellites` with SDR hardware for beginner users.

It is also possible to use the example GNU Radio companion flographs in [gr-frontends](#) to stream samples by UDP from different sources.

For more advanced users, `nc` can also be a very useful tool for streaming.

- `--kiss_in` can be used to process a file containing already decoded frames in KISS format. All the demodulation steps are skipped and only telemetry parsing, file receiving, etc. are done.

This can be useful to view the telemetry stored in files previously decoded with `gr-satellites` or other software.

5.1.3 Getting help

`gr_satellites` prints a detailed description of all the allowed arguments by using the `-h` or `--help` argument. Note that a satellite needs to be specified, since the set of allowed arguments depends on the decoders used by that satellite.

For example, this shows all the options allowed by the FUNcube-1 decoder:

```

$ gr_satellites FUNCube-1 --help
usage: gr_satellites satellite [-h] [--version]
                                (--wavfile WAVFILE | --rawfile RAWFILE | --rawint16_
↳RAWINT16 | --audio [DEVICE] | --udp | --kiss_in KISS_IN)
                                [--samp_rate SAMP_RATE] [--udp_ip UDP_IP]
                                [--udp_port UDP_PORT] [--iq]
                                [--input_gain INPUT_GAIN] [--kiss_out KISS_OUT]
                                [--kiss_append] [--kiss_server [PORT]]
                                [--kiss_server_address KISS_SERVER_ADDRESS]
                                [--zmq_pub [ADDRESS]] [--hexdump]
                                [--dump_path DUMP_PATH]
                                [--telemetry_output TELEMETRY_OUTPUT]
                                [--f_offset F_OFFSET] [--rrc_alpha RRC_ALPHA]
                                [--disable_fll] [--fll_bw FLL_BW]
                                [--clk_bw CLK_BW] [--clk_limit CLK_LIMIT]
                                [--costas_bw COSTAS_BW]
                                [--manchester_history MANCHESTER_HISTORY]
                                [--syncword_threshold SYNCWORD_THRESHOLD]
                                [--verbose_rs]

gr-satellites - GNU Radio decoders for Amateur satellites

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit

input:
  --wavfile WAVFILE     WAV input file
  --rawfile RAWFILE     RAW input file (float32 or complex64)
  --rawint16 RAWINT16   RAW input file (int16)
  --audio [DEVICE]     Soundcard device input
  --udp                 Use UDP input
  --kiss_in KISS_IN     KISS input file
  --samp_rate SAMP_RATE
                        Sample rate (Hz)
  --udp_ip UDP_IP       UDP input listen IP [default='::']
  --udp_port UDP_PORT   UDP input listen port [default='7355']
  --iq                 Use IQ input
  --input_gain INPUT_GAIN
                        Input gain (can be negative to invert signal) [default=1]

output:
  --kiss_out KISS_OUT   KISS output file
  --kiss_append         Append to KISS output file
  --kiss_server [PORT]  Enable KISS server [default port=8100]
  --kiss_server_address
                        KISS_SERVER_ADDRESS
                        KISS server bind address [default='127.0.0.1']
  --zmq_pub [ADDRESS]  Enable ZMQ PUB socket [default address=tcp://127.0.0.1:5555]
  --hexdump            Hexdump instead of telemetry parse
  --dump_path DUMP_PATH
                        Path to dump internal signals

demodulation:
  --f_offset F_OFFSET   Frequency offset (Hz) [default=1500 or 12000]
  --rrc_alpha RRC_ALPHA
                        RRC roll-off (Hz) [default=0.35]
  --disable_fll        Disable FLL

```

(continues on next page)

(continued from previous page)

```

--fll_bw FLL_BW          FLL bandwidth (Hz) [default=25]
--clk_bw CLK_BW         Clock recovery bandwidth (relative to baudrate) [default=0.
↪06]
--clk_limit CLK_LIMIT   Clock recovery limit (relative to baudrate) [default=0.02]
--costas_bw COSTAS_BW   Costas loop bandwidth (Hz) [default=50]
--manchester_history MANCHESTER_HISTORY Manchester recovery history (symbols) [default=32]

deframing:
--syncword_threshold SYNCWORD_THRESHOLD Syncword bit errors [default=8]
--verbose_rs           Verbose RS decoder

data sink:
--telemetry_output TELEMETRY_OUTPUT      Telemetry output file [default=stdout]

The satellite parameter can be specified using name, NORAD ID or path to YAML file

```

5.1.4 Output

By default, `gr_satellites` will “do its best” to show the user the output for the decoded frames. If the telemetry format for the satellite is implemented in `gr-satellites`, the telemetry frames will be printed to the standard output in human-readable format. Otherwise, the raw frames will be printed out in hex format to the standard output.

File decoding, image decoding and other special output options of some particular satellites are enabled by default.

Customization of the output options is described in the *Output options* subsection below.

5.1.5 Examples

The `test.sh` script in the `gr-satellites/` directory runs `gr_satellites` on several of the *sample recordings* in `satellite-recordings/`. This script can be used as a series of examples of how to run `gr_satellites`.

5.2 Output options

This subsection explains in detail the different output options that can be used with the `gr_satellites` command line tool. The default behaviour when no options are specified has been described in the *Output* subsection above.

5.2.1 Hex dump

By using the option `--hexdump`, it is possible to make `gr_satellites` print the received frames in hexadecimal format, regardless of whether there is a telemetry decoder available or not. The format used to print the frames is the same as used by the GNU Radio block `Message Debug print_pdu` input.

An example of the use of this option can be seen here:

```

$ gr_satellites FUNcube-1 --wavfile ~/gr-satellites/satellite-recordings/ao73.wav \
  --samp_rate 48e3 --hexdump
* MESSAGE DEBUG PRINT PDU VERBOSE *
()
pdu_length = 256
contents =
0000: 89 00 00 00 00 00 00 00 00 1f cc 00 ce 02 d1 00
0010: 00 07 08 09 09 00 00 05 01 01 00 40 13 2f c8 f2
0020: 5c 8f 34 23 f3 ba 0b 5d 62 74 51 c7 ea fa 69 4a
0030: 9a 9f 00 09 ef a0 1f f4 a7 ea 4a c6 8f 11 40 11
0040: 1e 10 f7 01 3e 20 64 00 d7 8b f8 d7 94 c8 93 a8
0050: 2a da 52 a6 0e 58 0e c8 0f 4e 01 1d 20 5a 00 db
0060: 94 a8 aa 8a 98 13 ac 69 0a a6 a8 10 e6 10 92 0f
0070: b8 01 50 20 64 00 d7 96 a8 c1 8b 48 25 ab a9 ca
0080: ce 9d 10 76 0f c9 10 55 01 3a 20 5a 00 d7 97 29
0090: 08 8c 48 4f a9 6a 5a f2 a4 10 39 0f 7b 0f 86 01
00a0: 49 20 64 00 d7 94 08 d0 8a d8 2a ad 6a 5a 7e b4
00b0: 0e 53 0e 9b 0e b7 01 09 20 5a 00 db 99 a8 f2 8f
00c0: e8 38 af aa 8a c2 9e 0e de 0f 48 0e 31 01 31 20
00d0: 5a 00 ce 9b c8 ff 88 68 1b b2 6a 5a ca a7 0f c3
00e0: 0e 74 0e 58 01 34 20 5a 00 d7 9b 39 1b 97 b8 c5
00f0: b0 2b 3a d6 b5 01 6b 00 6a 02 9e 00 03 20 13 00
*****

```

5.2.2 KISS output

Decoded frames can be saved to a file in [KISS format](#). This is a simple format that serves to delimit frames stored in a file or sent over a serial bus, and it is frequently used to store telemetry frames.

To enable KISS output, the `--kiss_out` parameter followed by the path of the output file should be used. By default `gr_satellites` will overwrite the file if it already exists. To append to the file instead, the option `--kiss_append` can be used in addition to the `--kiss_out` option. Appending can be used to concatenate frames obtained in several decoding runs.

Files in KISS format can be read with `gr_satellites` as indicated above or with other software tools.

Note: KISS files produced with `gr_satellites` use an extension proposed by [Mike Rupprecht](#) to store the reception timestamp of the frames. Before each data frame, a KISS control frame using the control byte `0x09` and storing a timestamp with UNIX timestamp in milliseconds stored as a big-endian 64 bit integer is included in the file.

Some software, including the decoders by [Mike Rupprecht](#), will be able to read and use these timestamps. Other software that processes KISS will ignore the timestamps.

5.2.3 KISS server

A KISS TCP server can be enabled with the `--kiss_server` parameter, optionally followed by the TCP port to listen on (by default port 8100 is used). This allows other applications to connect to `gr_satellites` and receive decoded frames using the KISS protocol.

By default the KISS server will only bind on `127.0.0.1` and listen to requests from localhost only. If access from other computers on the network is needed, the `--kiss_server_address` parameter can be used to specify the address to bind to. For instance, if `--kiss_server_address ''` or `--kiss_server_address 0.0.0.0` is used, the server will bind to `0.0.0.0` and listen to requests from all addresses.

5.2.4 ZMQ PUB socket

Decoded frames can also be sent to other applications by using a [ZeroMQ PUB](#) socket. Several applications can connect to the PUB socket using SUB sockets. The frames are sent using the *ZMQ PUB Message Sink* GNU Radio block, and can be received using the *ZMQ SUB Message Source* GNU Radio block.

The ZMQ PUB socket is enabled using the `--zmq_pub` parameter, optionally followed by the socket endpoint to use. By default, the endpoint `tcp://127.0.0.1:5555` is used. This means that the ZMQ PUB socket will only listen to connections from localhost. If desired, the endpoint `tcp://*:5555` can be used to listen on all addresses.

5.2.5 Telemetry output

For satellites supporting telemetry parsing, `gr_satellites` will default to printing the decoded telemetry values to the standard output. It is possible to write these messages to a file instead by using the `--telemetry_output` parameter followed by the path of the output file.

5.2.6 Dump internal signals

For advanced users and developers, the demodulators used in `gr_satellites` can dump the internal signals used inside the demodulator. This option can be enabled by using the `--dump_path` parameter followed by a path to the directory where the different files are created. It is recommended to use this option with a short recording, to avoid creating very large files. The details of each of these files are best studied in the Python source code of the demodulators (see `python/components/demodulators/`).

The following example show how to use `--dump_path` to plot the symbols with [Numpy](#) and [Matplotlib](#) and optimize the decoding parameters for a particular recording. We first run the following to dump to the path `/tmp/fsk` the internal signals produced by decoding a sample recording of AU02.

```
$ mkdir -p /tmp/fsk
$ gr_satellites AU02 --wavfile satellite-recordings/au02.wav \
  --samp_rate 48e3 --dump_path /tmp/fsk
```

We see that we do not get any decoded packets. Then, we can plot the FSK symbols with the following Python code:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.fromfile('/tmp/fsk/clock_recovery_out.f32', dtype = 'float32')
plt.plot(x, '.')
plt.show()
```

This produces the figure below, which shows that there has been a clock cycle slip mid packet, which prevents correct decoding.

We can run `gr_satellites` again adding the parameter `--clk_bw 0.1` to increase the clock recovery loop bandwidth. With this parameter we get a successful decode and if we plot the FSK symbols again, we get the figure below, which shows that the clock recovery is working much better than before.

5.3 Telemetry submission

The `gr_satellites` command line tool can be used to submit decoded telemetry to an online database server, such as [SatNOGS DB](#) and these others servers used by certain satellite projects:

- [FUNcube Warehouse](#), which is used by the FUNcube payloads on FUNcube-1, UKube-1, Nayif-1 and JY1Sat.

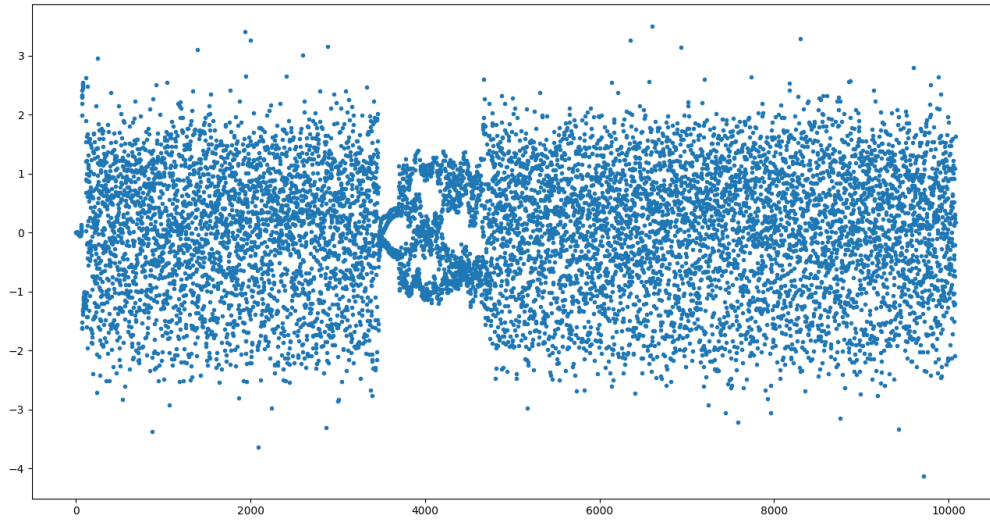


Fig. 1: FSK symbols with default parameters

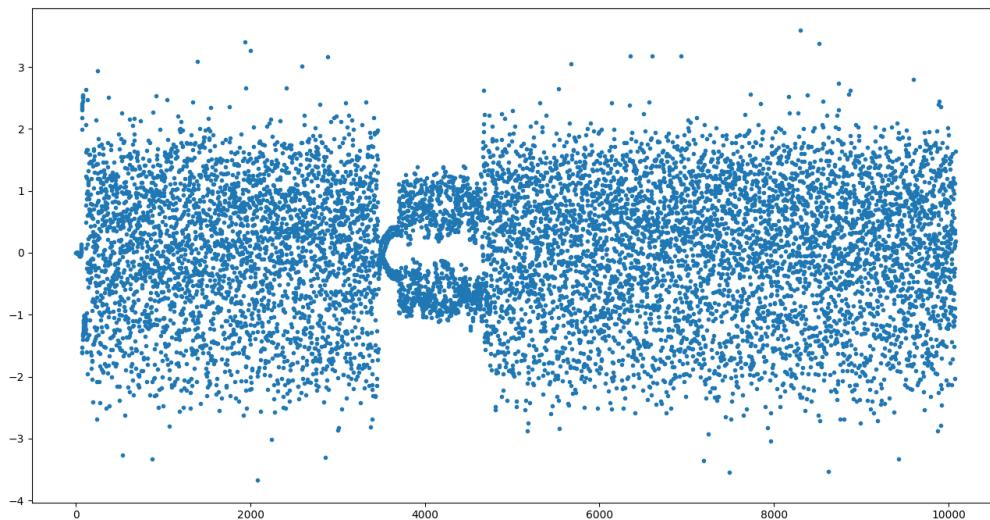


Fig. 2: FSK symbols with non-default parameters

- [PW-Sat2 Groundstation](#), which is used by PW-Sat2.
- The [BME telemetry server](#), which is used by SMOG-P, ATL-1 and SMOG-1.
- [Harbin Institute of Technology](#), which connects to the telemetry proxy included in [gr-lilacsat](#) and [gr-dslwp](#).

To enable telemetry submission, it is necessary to edit some parameters in `gr_satellites`'s config file, which is located in `~/.gr_satellites/config.ini`. If this file does not exist, it will be created with a template when `gr_satellites` is first run. The template looks like this:

To enable telemetry submission, the `submit_tlm` parameter must be set to `yes`. Additionally, the receiving stations `callsign` as well as its location (`latitude` and `longitude`) need to be set, since some of the servers need these parameters. Once this is done, telemetry submission to SatNOGS DB will be enabled for all satellites.

To enable telemetry submission to the FUNcube warehouse, it is necessary to fill in the `site_id` and `auth_code`. These can be obtained by [registering in the warehouse](#).

To enable telemetry submission to the PW-Sat2 server, it is necessary to enter the path to the credentials file in the `credentials_file` parameter. This file is a JSON file that is generated and downloaded in the “[Your credentials](#)” section of the server web interface. It is necessary to have an account registered in the server to obtain the credentials file.

To enable telemetry submission to the BME server, it is necessary to [register an account in the BME server](#). The user and password should be entered into the `gr-satellites .ini` file.

To use the Harbin Institute of Technology proxy to submit telemetry, the proxy needs to be run and started in the local computer before running `gr_satellites`. The command line tool will connect to the correct port where the proxy is listening (this is specified in the SatYAML file of each satellite). All the configuration regarding the station and the operator is done in the proxy itself. When `gr_satellites` starts, it will attempt to connect to the proxy, and print a warning if unable (in which case telemetry submission through the proxy is disabled for this run).

Note: The Harbin Institute of Technology proxy is a Python2 application that uses PyQt4. Users having more modern systems may find useful the PyQt5 version that can be found in the [pyqt5 branch of gr-lilacsat](#). This requires `tornado` version 4.5.3. It will not work with more recent versions of `tornado`.

For some telemetry servers, including SatNOGS DB, the frames are submitted together with a timestamp of reception. This timestamp is taken from the computer's clock by `gr_satellites` at the moment when it decodes the frame. This means that, in order to use telemetry submission appropriately, the computer's clock should be set accurately and a live signal rather than a recording should be decoded.

5.4 File and image receiver

Some satellites transmit files (especially image files) by splitting the files into many telemetry packets. The `gr_satellites` decoder supports reassembling and storing these files into a directory. Additionally, image files are automatically displayed in real time as they are being received, using `feh`.

Currently the satellites that have decoders supporting file reception are ATL-1 and SMOG-P (they transmit RF spectrum data), and the satellites that have decoders supporting image reception are 1KUNS-PF, BY70-1, D-SAT, LilacSat-1 and Światowid.

For satellites supporting file reception, the `--file_output_path` parameter can be used to set the directory that is used to store received files. The filenames of the received files will be automatically created using metadata or a counter (if no metadata is transmitted). By default, received files are stored in `/tmp/`.

The `--verbose_file_receiver` parameter can be used to enable additional debugging information about the functionality of the file receiver.

5.5 Other topics

This subsection deals with other topics which are relevant to the usage of `gr_satellites`.

5.5.1 Real or IQ input

The `gr_satellites` command line tool supports both real (one-channel) input and IQ input (which consists of two channels: in-phase and quadrature). A detailed description of these two ways to represent a signal is out of the scope of this document. This subsection gives some practical advice regarding the difference between real and IQ input.

By default `gr_satellites` will assume that its input is real. To use IQ input, the `--iq` option must be used.

When using the audio output of either a conventional radio or an SDR software performing SSB or FM demodulation, `gr_satellites` should be used with the real input option. Likewise, recordings produced from this kind of audio output, such as one-channel WAV recordings should also be used with the real input option.

However, most SDR softwares will also have an option to save raw samples to a file. These files are almost always IQ, and can be either a two-channel WAV file or a file in raw format. The IQ input option must be used when using `gr_satellites` to read these files. Additionally, some SDR software may support streaming IQ data by UDP. This can also be used in `gr_satellites` with the IQ input option.

5.5.2 FSK demodulation and IQ input

When using an AFSK or FSK demodulator, the usage of the `--iq` option has an additional effect. Since (A)FSK is a mode based on frequency modulation, it is common to use either a conventional FM radio or an SDR software performing FM demodulation to receive (A)FSK. Audio recordings obtained in this manner are also common. Therefore, when `gr_satellites` is run without the `--iq` signal, it will expect that (A)FSK signals have already been FM-demodulated in this way.

When the `--iq` option is used, `gr_satellites` expects an (A)FSK signal that has not been FM-demodulated, and so it will perform FM-demodulation first. This is the kind of procedure that should be employed with inputs such as raw IQ recordings of an SDR, since the (A)FSK signals present in this kind of recordings have not been FM-demodulated.

Note: The output of the radio or SDR software when running in FM mode to receive an FSK signal is actually an NRZ signal. Therefore, when `gr_satellites` is run without the `--iq` option, it will expect an NRZ signal instead of an FSK signal. When `gr_satellites` is run with the `--iq` option, it will expect an FSK signal.

Similarly, the output of the radio or SDR software when running in FM mode to receive an AFSK signal is actually an audio-frequency FSK signal. Therefore, when `gr_satellites` is run without the `--iq` option, it will expect an audio-frequency FSK signal instead of an AFSK signal. When `gr_satellites` is run with the `--iq` option, it will expect an AFSK signal.

Note that this behaviour is what the user wants in most cases, but it also means that it is not possible to run `gr_satellites` directly on an (A)FSK signal which is represented in intermediate frequency as a real signal.

5.5.3 Frequency offsets for BPSK

A usual way of receiving a BPSK signal is to use either a conventional radio or an SDR software in SSB mode (USB mode, normally) and tune the BPSK signal in the middle of the audio passband. Audio recordings obtained in this manner are also common.

Note: The SSB filter of a conventional radio is often approximately 3kHz wide. For this reason, only BPSK signals with a baudrate of 2400 baud or lower can be received with a conventional SSB radio. For BPSK signals with larger baudrate, an SDR receiver should be used.

The `gr_satellites` command line tool needs to know the frequency at which the BPSK signal is tuned within the audio passband. If necessary, this can be specified with the `--f_offset` parameter, followed by the frequency in Hz. There are the following defaults:

- For signals with a baudrate of 2400 baud or less, a frequency offset of 1500 Hz is used. This follows the common practice of using a regular 3kHz SSB bandwidth and tuning the signal in the middle of the passband.
- For signals with a baudrate larger than 2400, a frequency offset of 12000 Hz is used. The rationale is that, for best results, a passband of 24000 Hz should be used, since this is the largest that fits in a 48kHz audio signal, and the signal should be tuned in the middle of this 24000 Hz passband. This kind of usage is sometimes called “wide SSB mode”.

These settings only apply for a real input. When `gr_satellites` is used with IQ input, the default is to expect the BPSK signal tuned at 0Hz (i.e., at baseband). A different frequency can still be selected with the `--f_offset` parameter.

5.5.4 FSK signal polarity

A conventional FM radio, or even an SDR software running in FM mode might invert the polarity of the output signal, since the polarity is not relevant for audio signals. However, the polarity is relevant when receiving an FSK signal that does not use differential coding.

An input with the inverted polarity will cause decoding to fail. In this case, the input can be inverted again by using the `--input_gain -1` parameter, which has the effect of multiplying the input signal by -1 before it is processed, thus restoring the correct polarity.

Satellite decoder block

The Satellite decoder block brings most of the functionality of the `gr_satellites` command line tool in the form of a GNU Radio block. This allows the experienced user to leverage the functionality of the satellite decoders in their own designs or to achieve a greater degree of customization than what is possible with the command line tool.

The input of the Satellite decoder block is a stream of samples, which can be either real or complex, for IQ input (see *Real or IQ input*). The output of the block is PDUs with the decoded frames. The figure below shows a very basic use of the Satellite decoder block, where the input is taken from a WAV recording using the Wav File Source block and the output is printed using the Message Debug block. This example can be found in `gr-satellites` in `examples/satellite_decoder/satellite_decoder.grc`.

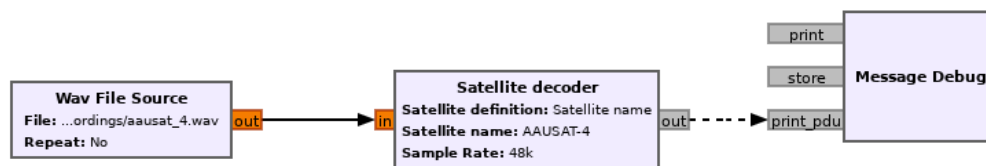
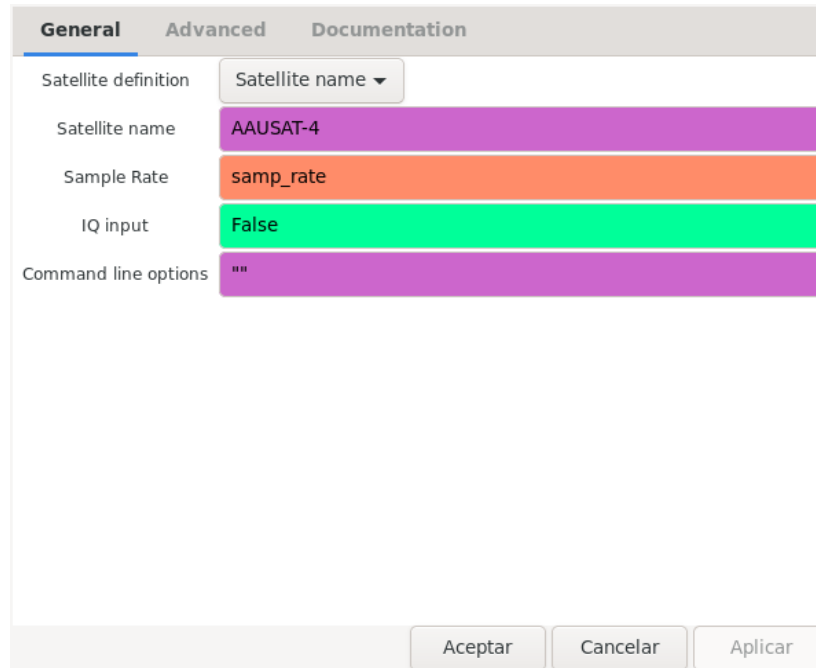


Fig. 1: Usage of Satellite decoder in a flowgraph

The figure below shows the options for the Satellite decoder block. It is possible to specify the satellite to use in the same ways as for the `gr_satellites` command line tool (see *Specifying the satellite*). The method to specify the satellite is chosen in the *Satellite definition* dropdown menu. The sample rate needs to be entered in the *Sample Rate* field, and the *IQ input* field selects real or IQ input. The *Command line options* field is described below.

Here are a few ideas of how the Satellite decoder block can be employed by users to build custom decoders which are not possible with the command line tool.

On the input side, it is possible to use all the standard GNU Radio blocks to support a large number of SDR hardware and recording formats. The different channeliser and filter blocks (especially “Frequency Xlating FIR Filter”) can be used to adapt the sample rate and bandwidth of the signal into something useful for the satellite decoder. For example, a wideband SDR might be used to receive the signal of different satellites, performing Doppler correction with `gr-predict-doppler`. The signals of these satellites might be channelised with a “Frequency Xlating FIR Filter” blocks and fed into independent Satellite decoder blocks.



The image shows a dialog box titled "Options of Satellite decoder" with three tabs: "General", "Advanced", and "Documentation". The "General" tab is selected. It contains the following fields:

Field	Value
Satellite definition	Satellite name ▾
Satellite name	AAUSAT-4
Sample Rate	samp_rate
IQ input	False
Command line options	""

At the bottom of the dialog box, there are three buttons: "Aceptar", "Cancelar", and "Aplicar".

Fig. 2: Options of Satellite decoder

On the output side, it is possible to treat the received PDUs freely. This allows classifying and storing them in different ways. Upper layer complex protocols might be completely handled inside the GNU Radio flowgraph, provided there is a suitable implementation of these protocols. Additionally, it is possible to interface the decoder to external tools with default GNU Radio blocks, by using TCP sockets or ZeroMQ.

6.1 Command line options

The satellite decoder block allows entering the same kind of command line options supported by the `gr_satellites` command line tool into the *Command line options* parameter of the block. The set of options to use needs to be specified as a Python script. To see the available options, it is possible to use `--help` as the options, just as one would do when using a command line tool. When the flowgraph is run, it will print out the allowable options and stop. In the same manner, if invalid options are specified, when the flowgraph is run it will print the correct usage and stop.

Components represent gr-satellite's way of decomposing the decoding process in high-level blocks. The decoding chain is broken into a series of steps which pass their output to the input of the next step. These are the following:

- **Data sources.** These produce the input of the decoding chain, which typically consists of RF signal samples.
- **Demodulators.** These turn RF samples into soft symbols. They filter the signal, recover the transmit clock and carrier if necessary, etc. An example is a BPSK demodulator, which turns RF samples of a BPSK signal into a stream of soft symbols.
- **Deframers.** Deframers implement the lower layer protocols related to frame boundary detection, descrambling, deinterleaving, FEC, error checking with a CRC code, etc. The output of a deframer are PDUs with the frames. Some examples are an AX.25 deframer and a CCSDS concatenated code deframer.
- **Transports.** Transports implement higher layer protocols that might be needed to get to the useful information inside the frames. For example, if frames are fragmented, a transport will handle defragmentation. An example is a KISS transport, whose input are frames that contain bytes of a KISS stream, and its output are the packets contained in that KISS stream, regardless of how they are split between different frames.
- **Data sinks.** Data sinks are the consumers of packets. They might store them, send them to another software, or parse telemetry values.

All the component blocks support *Command line options* in the same way as the satellite decoder block. The set of available options for each component block is different. It is possible to use the "--help" as the options of a particular block in order to print out the available options for that block.

Below, the main component blocks in each category are described.

7.1 Data sources

Data source components can be found under *Satellites > Data sources* in GNU Radio companion. Currently, the only data source is the "KISS File Source" block. This block will read a file in KISS format, and output the frames in the file as PDUs.

The usual operations involving reading RF samples from an SDR or recording can be achieved easily with default GNU Radio blocks, so there are no specific data sources for these. Advanced users can look at the

`setup_input()` method of the class `gr_satellites_top_block` in `apps/gr_satellites` to see how the `gr_satellites` command line tools sets up its different inputs using default GNU Radio blocks.

7.2 Demodulators

Demodulator components can be found under *Satellites > Demodulators* in GNU Radio companion. There are currently three demodulator component blocks:

- BPSK demodulator
- FSK demodulator
- AFSK demodulator

They take RF signal samples as input, and output soft symbols, as a stream of `float` normalized with amplitude one. The input can be either real or IQ (complex). See *Real or IQ input* for more information.

The demodulator blocks and their parameters are described below.

7.2.1 BPSK demodulator

The BPSK demodulator expects an input which consists of RF samples of a BPSK signal, and outputs the demodulated BPSK soft symbols. The BPSK signal can optionally be DBPSK or Manchester encoded.

The figure below shows the example flowgraph which can be found in `examples/components/bpsk_demodulator.grc`. This reads a WAV file from *satellite-recordings* which contains some BPSK packets from LilacSat-1 and uses the BPSK demodulator to obtain the symbols. The “Skip Head” and “Head” blocks are used to select a portion of the output, which is then plotted using the “QT GUI Time Sink”.

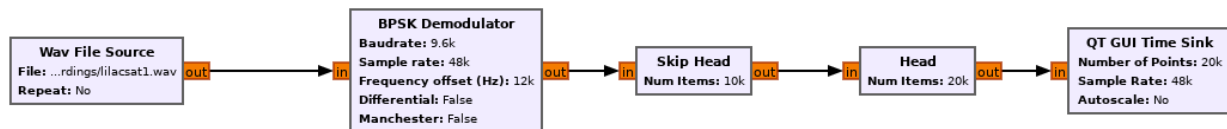


Fig. 1: Usage of BPSK demodulator in a flowgraph

When this example flowgraph is run, it displays the output shown in the figure below. There we can see the start of the BPSK packet. On the left side of the plot we have noise, before the packet starts, then the packet starts, and the clock and carrier recovery take some time to sync. After this, the symbols are demodulated properly. This can be seen because the +1 and -1 symbols are well separated.

The figure below shows the options allowed by the BPSK demodulator block. The *Baudrate* option is used to set the baudrate in symbols per second. The *Sample rate* option specifies the sample rate of the input. The *Frequency offset* specifies at which frequency the BPSK signal is centred (see *Frequency offsets for BPSK*).

The *Differential* option enables differential decoding of DBPSK. For differential decoding, the phase recovery using a Costas loop is disabled and non-coherent demodulation is used.

The *Manchester* option enables Manchester decoding. A Manchester encoded BPSK signal is decoded as if it had twice the baudrate, and then the phase of the Manchester clock is searched in the symbols and the Manchester clock is “wiped-off”, multiplying symbols by the clock and accumulating them by pairs.

The *IQ input* option enables IQ (complex) input.

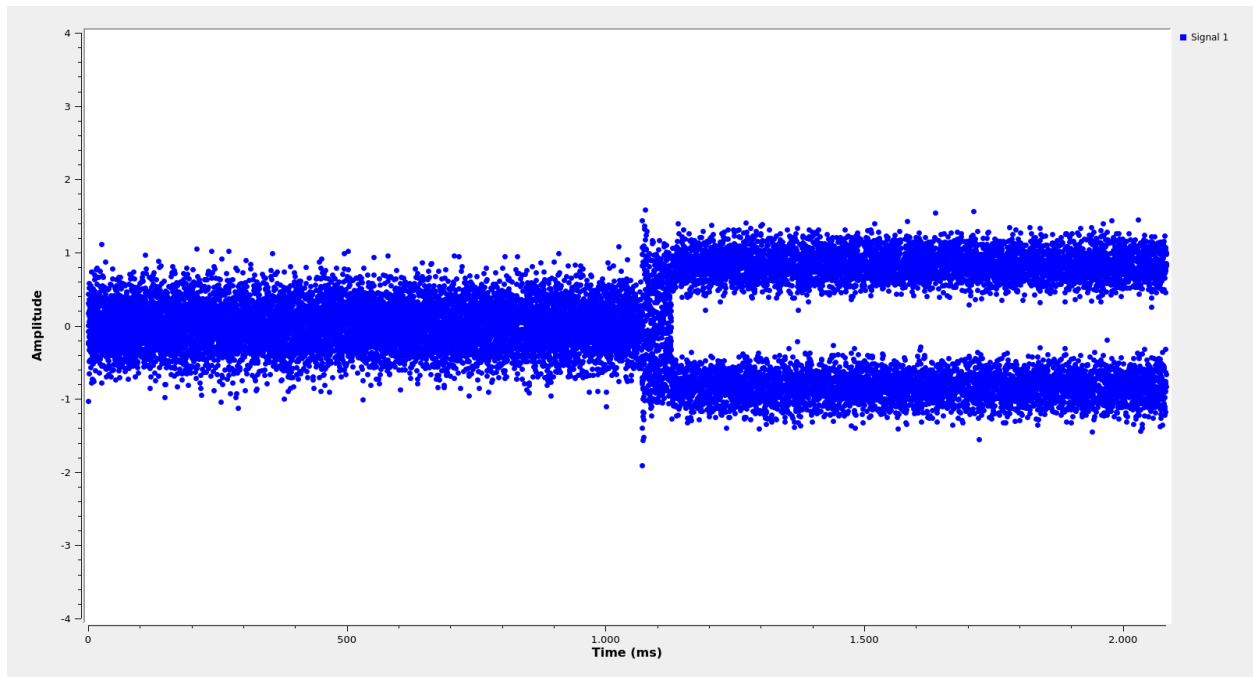


Fig. 2: Output of the BPSK demodulator example flowgraph

General	Advanced	Documentation
Baudrate	9600	
Sample rate	samp_rate	
Frequency offset (Hz)	12000	
Differential	False	
Manchester	False	
IQ input	False	
Command line options	""	

Acceptar Cancelar Aplicar

Fig. 3: Options of BPSK demodulator

7.2.2 FSK demodulator

The FSK demodulator expects an input which consists of RF samples of an FSK signal, and outputs the demodulated FSK soft symbols. Both real and IQ (complex) input are supported, but the semantics are different: with real input, the FSK demodulator expects an FM-demodulated signal; with IQ input, the FSK demodulator expects the signal before FM demodulation (see *FSK demodulation and IQ input*).

The figure below shows the example flowgraph which can be found in `examples/components/fsk_demodulator.grc`. This reads a WAV file from *satellite-recordings* which contains a single FSK packet from AAUSAT-4 and uses the FSK demodulator to obtain the symbols. The output is plotted using the “QT GUI Time Sink”.

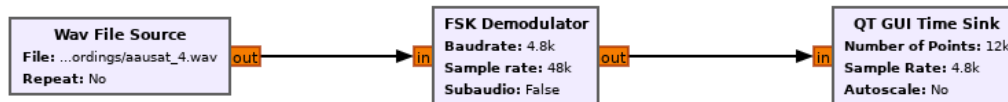


Fig. 4: Usage of FSK demodulator in a flowgraph

When this example flowgraph is run, it displays the output shown in the figure below. There we can see the FSK packet, surrounded by noise on both sides.

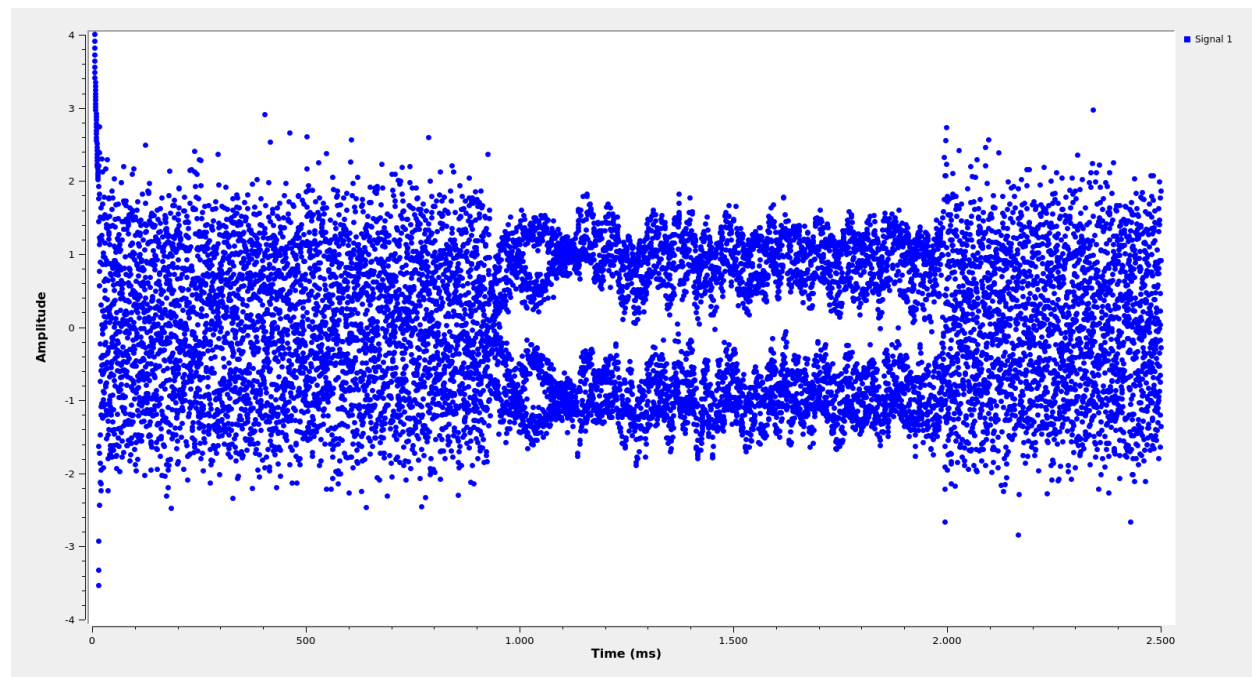


Fig. 5: Output of the FSK demodulator example flowgraph

The figure below shows the options allowed by the FSK demodulator block. The *Baudrate* option is used to set the baudrate in symbols per second. The *Sample rate* option specifies the sample rate of the input. The *IQ input* option enables IQ (complex) input. The signal is expected to be centred at baseband (0Hz) when IQ input is selected. The *Subaudio* option enables subaudio demodulation, which is intended for subaudio telemetry under FM voice and includes an additional lowpass filter to filter out the voice signal.

General	Advanced	Documentation
Baudrate	4800	
Sample rate	samp_rate	
IQ input	False	
Subaudio	False	
Command line options	""	

Fig. 6: Options of FSK demodulator

7.2.3 AFSK demodulator

The AFSK demodulator expects an input which consists of RF samples of an AFSK signal, and outputs the demodulated AFSK soft symbols. Both real and IQ (complex) input are supported, but the semantics are different: with real input, the AFSK demodulator expects an FM-demodulated signal; with IQ input, the AFSK demodulator expects the signal before FM demodulation (see *FSK demodulation and IQ input*).

The figure below shows the example flowgraph which can be found in `examples/components/afsk_demodulator.grc`. This reads a WAV file from `satellite-recordings` which contains a single AFSK packet from GOMX-1 and uses the AFSK demodulator to obtain the symbols. The “Head” block is used to select a portion of the output, which is then plotted using the “QT GUI Time Sink”.

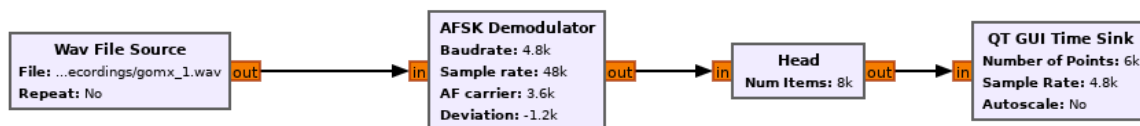


Fig. 7: Usage of AFSK demodulator in a flowgraph

When this example flowgraph is run, it displays the output shown in the figure below. There we can see the AFSK packet, surrounded by noise on both sides.

The figure below shows the options allowed by the AFSK demodulator block. The *Baudrate* option is used to set the baudrate in symbols per second. The *Sample rate* option specifies the sample rate of the input.

The *AF carrier* option specifies the audio frequency in Hz on which the FSK tones are centred. The *Deviation* option specifies the separation in Hz between each of the tones and the AF carrier. If the deviation is positive, the high tone is interpreted as representing the symbol 1, while the low tone is interpreted as representing the symbol 0 (or -1 in bipolar representation). If the deviation is negative, the low tone is interpreted as representing the symbol 1 and the high tone is interpreted as representing the symbol 0.

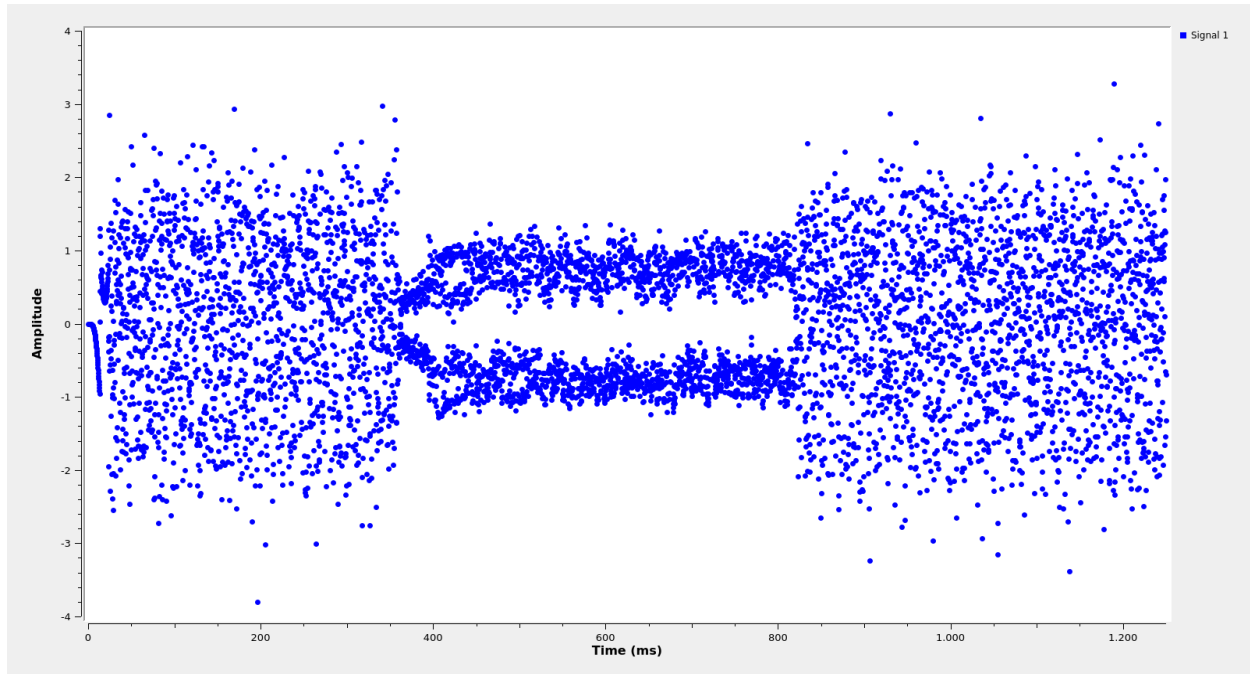


Fig. 8: Output of the AFSK demodulator example flowgraph

In this example, the AF carrier is 3600 Hz and the deviation is -1200 Hz. This means that the tone representing 1 is at 2400 Hz, while the tone representing 0 is at 4800 Hz (the signal is actually 4800 baud GMSK).

The *IQ input* option enables IQ (complex) input.

7.3 Deframers

Deframer components can be found under *Satellites > Deframers* in GNU Radio companion. There is a large number of deframer component blocks, since many satellites use ad-hoc protocols for framing, so a custom deframer is used for those satellites.

Deframers take soft symbols, produced as the output of one of the demodulator components, and detect frame boundaries, perform as necessary descrambling, deinterleaving, FEC decoding, CRC checking, etc.

Here, the most popular deframers are described. For ad-hoc deframers that are used in few satellites, the reader is referred to the documentation of each of the blocks in GNU Radio companion.

7.3.1 AX.25 deframer

The AX.25 deframer implements the [AX.25](#) protocol. It performs NRZ-I decoding, frame boundary detection, bit de-stuffing, and CRC-16 checking. Optionally, it can also perform G3RUH descrambling. G3RUH scrambling is typically used for faster baudrates, such as 9k6 FSK packet radio, but not for slower baudrates, such as 1k2 AFSK packet radio.

The figure below shows an example flowgraph of the AX.25 deframer block. This example can be found in `examples/components/ax25_deframer.grc`. The example reads a WAV file from *satellite-recordings* containing 9k6 FSK AX.25 packets from US01, demodulates them with the FSK demodulator block, deframes them with AX.25 deframer, and prints the output with the Message Debug block.

General	Advanced	Documentation
Baudrate	4800	
Sample rate	samp_rate	
AF carrier	3600	
Deviation	-1200	
IQ input	False	
Command line options	""	

Fig. 9: Options of AFSK demodulator

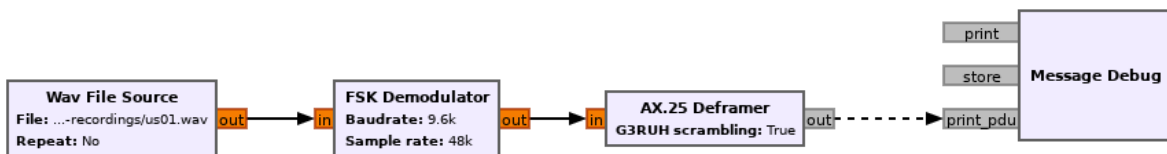


Fig. 10: Usage of AX.25 deframer in a flowgraph

The AX.25 deframer block has a single option that indicates whether G3RUH descrambling should be performed or not.

7.3.2 GOMspace AX100 deframer

The GOMspace AX100 deframer implements two different protocols used by the popular GOMspace NanoCom AX100 transceiver. These two protocols are:

- ASM+Golay. This uses a header encoded with a Golay(24,12) code that indicates the packet length. The payload is Reed-Solomon encoded with a (255,223) CCSDS code and scrambled with the CCSDS synchronous scrambler.
- Reed Solomon. This uses a G3RUH asynchronous scrambler. The first byte of the packets indicates the length of the payload and is sent unprotected. The packet payload is Reed-Solomon encoded with a (255,223) CCSDS code.

The figure below shows an example flowgraph of the AX100 deframer block running in both modes. This example can be found in `examples/components/ax100_deframer.grc`. For ASM+Golay decoding the example reads a WAV file from `satellite-recordings` containing packets from 1KUNS-PF. For Reed Solomon decoding the example reads a WAV file from `satellite-recordings` which contains packets from TW-1B. The output frames are printed with Message Debug blocks.

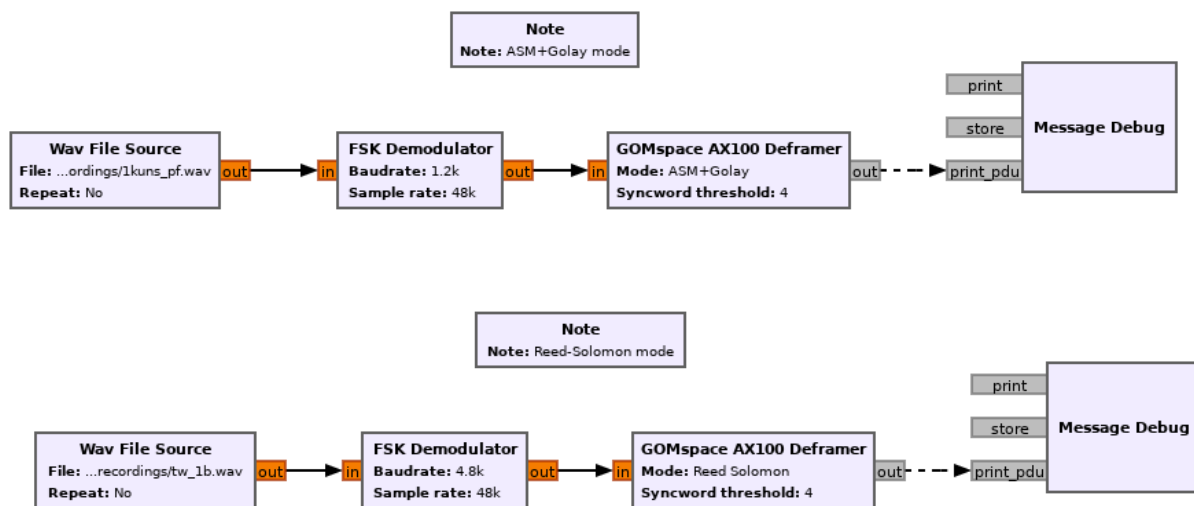


Fig. 11: Usage of AX100 deframer in a flowgraph

The AX100 deframer only has two options, the *Mode* option indicates the mode, as described above, and the *Syncword threshold* option specifies how many bit errors are allowed in the detection of the 32 bit syncword.

7.3.3 GOMspace U482C deframer

The GOMspace U482C deframer implements the protocol used by the GOMspace NanoCom U482C transceiver, which is an older transceiver from GOMspace that is still seen in some satellites.

The protocol used by the U482C is similar to the ASM+Golay mode used by the AX100. The packet payload can be optionally:

- Encoded with the CCSDS $r=1/2$, $k=7$ convolutional encoder
- Scrambled with the CCSDS synchronous scrambler

- Encoded with a CCSDS (255,223) Reed-Solomon code

The packet header has flags that indicate which of these options are in use, in addition to the length field.

The U482C modem uses AFSK with a 4800 baud audio-frequency GMSK waveform.

The figure below shows an example flowgraph of the U482C deframer block. This example can be found in `examples/components/u482c_deframer.grc`. The example reads a WAV file from *satellite-recordings* containing a packet from GOMX-1. The packet is demodulated and deframed, and the output is printed in hex using the Message Debug block.

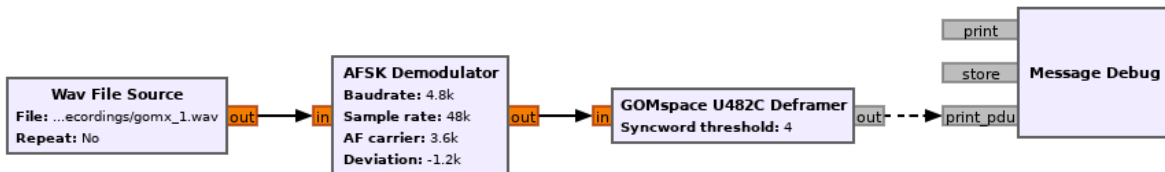


Fig. 12: Usage of U482C deframer in a flowgraph

The U482C deframer has a single option, which indicates the number of bit errors that are allowed in the syncword detection.

7.3.4 AO-40 FEC deframer

The AO-40 FEC deframer implements the protocol designed by Phil Karn KA9Q for the AO-40 FEC beacon. This protocol is currently used in the FUNcube satellites and others.

The FEC is based on CCSDS recommendations and uses a pair of interleaved Reed-Solomon (160,128) codes, the CCSDS synchronous scrambler, the CCSDS $r=1/2$, $k=7$ convolutional code, interleaving and a distributed syncword.

The figure below shows an example flowgraph of the AO-40 FEC deframer block. This example can be found in `examples/components/ao40_fec_deframer.grc`. It reads a WAV file from *satellite-recordings* containing a packet from AO-73 (FUNcube-1). The packet is first BPSK demodulated and then deframed with the AO-40 FEC deframer. The output is printed out using the Message Debug block.

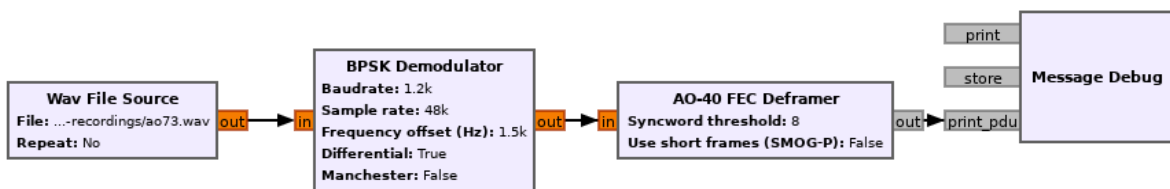


Fig. 13: Usage of AO-40 FEC deframer in a flowgraph

The AO-40 FEC deframer block has two options. The *Syncword threshold* option indicates the number of bit errors to allow in the syncword detection. The *Use short frames* option toggles the usage of short frames. This is a variant of the AO-40 FEC protocol which is based on a single Reed-Solomon codeword and is used by SMOG-P and ATL-1.

7.3.5 CCSDS deframers

The CCSDS Concatenated deframer and CCSDS Reed-Solomon deframer blocks implement some of the CCSDS protocols defined in the TM Synchronization and Channel Coding Blue Book (see the [CCSDS Blue Books](#)).

The CCSDS Reed-Solomon deframer implements Reed-Solomon TM frames, which use a Reed-Solomon (255, 223) code (or a shortened version of this code) and the CCSDS synchronous scrambler. The CCSDS Concatenated deframer implements concatenated TM frames, which add an $r=1/2$, $k=7$ convolutional code as an inner coding to the Reed-Solomon frames. The usage of both deframers is very similar.

The figure below shows an example flowgraph of the CCSDS Concatenated deframer block. This example can be found in `examples/components/ccsds_deframer.grc`. It reads a WAV file from *satellite-recordings* containing some packets from BY70-1. These are concatenated TM frames with a frame size of 114 bytes and differential encoding (to solve the BPSK phase ambiguity). The packet is first BPSK demodulated and then deframed. The output is printed using the Message Debug block.

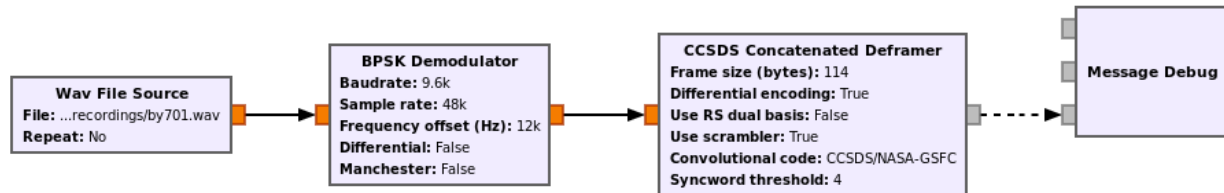


Fig. 14: Usage of CCSDS Concatenated deframer in a flowgraph

The figure below shows the options used by the CCSDS Concatenated deframer. The CCSDS Reed-Solomon deframer block allows exactly the same options, except for the *Convolutional code* option, since all the other options refer to the convolutional inner code.

The *Frame size* option indicates the size of the frame in bytes (after Reed-Solomon decoding). The *Differential encoding* option enables differential decoding, which is often used to solve the BPSK 180° phase ambiguity. The *Use RS dual basis* option enables the usage of the dual basis definition for the Reed-Solomon code. The *Use scrambler* option enables or disables the use of the CCSDS synchronous descrambler. Most satellites using CCSDS frames use scrambling. The *Convolutional code* option can be used to toggle between the CCSDS/NASA-GSFC (most usual and default) and NASA-DSN conventions for the convolutional code. The *Syncword threshold* option can be used to choose the number of bit errors that are allowed in the detection of the syncword.

7.4 Transports

Transport components can be found under *Satellites > Transports* in GNU Radio companion. Transports are designed to implement upper layer protocols. They take as input the output of a demodulator, which contains physical layer or link layer frames and process it to obtain upper layer packets. Some of the typical functionalities implemented by these upper layer protocols include fragmentation/defragmentation.

The only transport available so far in gr-satellites is the KISS transport.

7.4.1 KISS transport

The KISS transport implements fragmentation/defragmentation according to the KISS protocol for packet boundary detection. Its input should be PDUs containing the bytes of a KISS stream. The frames are joined and the KISS stream is followed, detecting packet boundaries and extracting the packets. The packets are output as PDUs.

The figure below shows an example flowgraph of the KISS transport, which can be found in `examples/components/kiss_transport.grc`. It is based on the CCSDS Concatenated deframer example described above. BY70-1 sends frames which contain the bytes of a KISS stream, so the KISS transport can be used to extract the packets from this stream. There are two Message Debug blocks that can be enabled or disabled in order to see the input or the output of the KISS transport block.

General	Advanced	Documentation
Frame size (bytes)	114	
Differential encoding	True	
Use RS dual basis	False	
Use scrambler	True	
Convolutional code	CCSDS/NASA-GSFC	
Syncword threshold	4	
Command line options	""	

Fig. 15: Options of CCSDS Concatenated deframer

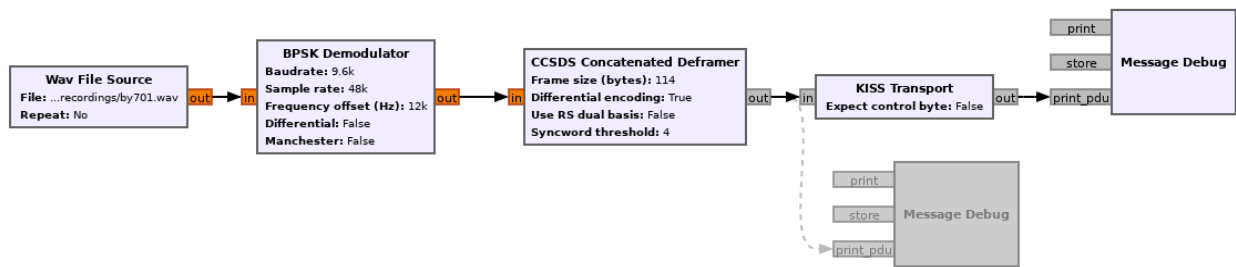


Fig. 16: Usage of KISS transport in a flowgraph

When the example is run, the frames at the input of the input of the KISS transport look like the one below. We see that there is a single packet embedded into the 114 byte Reed-Solomon frame, using `c0` KISS idle bytes for padding.

```
pdu_length = 114
contents =
0000: c0 b8 64 3d 00 12 00 00 00 00 c8 3a 00 80 00 00
0010: 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
0020: 32 32 32 32 32 32 32 32 32 32 32 32 32 32 ff c4
0030: 00 1f 00 00 01 05 01 01 01 01 01 01 01 00 00 00
0040: 00 00 00 00 01 02 03 04 05 06 07 08 09 0a 0b ff
0050: 18 21 00 00 db dc 4b f7 07 c0 c0 c0 c0 c0 c0
0060: c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0
0070: c0 c0
```

The frames at the output of the KISS transport look like the following. We see that the `c0` KISS idle bytes have been stripped. The KISS transport can also handle the case when a packet is longer than 114 bytes and has been fragmented into several 114 byte frames.

```
pdu_length = 87
contents =
0000: b8 64 3d 00 12 00 00 00 00 c8 3a 00 80 00 00 32
0010: 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
0020: 32 32 32 32 32 32 32 32 32 32 32 32 32 32 ff c4 00
0030: 1f 00 00 01 05 01 01 01 01 01 01 00 00 00 00 00
0040: 00 00 00 01 02 03 04 05 06 07 08 09 0a 0b ff 18
0050: 21 00 00 c0 4b f7 07
```

The KISS transport has a single option, called *Expect control byte*. When it is set to `True`, the first byte before the packet payload is interpreted as a control byte according to the KISS protocol. If it is set to `False`, it is assumed that there is no control byte preceding the packet payload. When using KISS as a means to fragment/defragment upper layer packets it is more common not to use control bytes.

7.5 Data sinks

Data sink components are the final consumers of the PDUs that contain the decoded frames. They can be used for several things, such as printing telemetry values, saving frames to a file, sending frames to an online telemetry database server, and reassembling files and images. The different data sinks available in gr-satellites are described below.

7.5.1 Telemetry parser

The telemetry parser uses `construct` to parse a PDU containing a telemetry frame into the different fields and prints the parsed values to the standard output or a file.

The parser uses *telemetry definitions*, which are either `Construct` objects (typically a `Struct`) or any other object supporting the `parse()` method in case more complex parsing behaviour is needed. The list of available telemetry definitions can be seen in `python/telemetry/__index__.py`, or by calling `import satellites.telemetry; help(satellites.telemetry)` in python3.

The figure below shows an example flowgraph of the Telemetry parser block, which can be found in `examples/components/telemetry_paser.grc`. It is based on the U482C example described above. The packets sent by GOMX-1 are deframed and the the Telemetry parser is used to print out the telemetry values to the standard output.

The beginning of the ouput produced by the Telemetry parser block can be seen below.

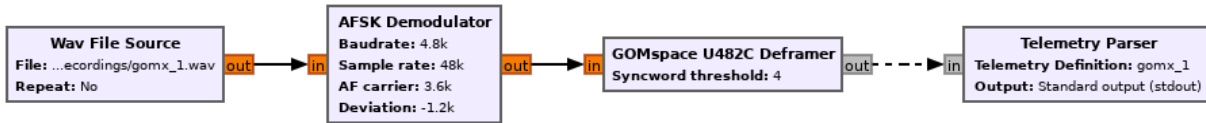


Fig. 17: Usage of Telemetry parser in a flowgraph

```

Container:
  csp_header = Container:
    priority = 2
    source = 1
    destination = 10
    destination_port = 30
    source_port = 0
    reserved = 0
    hmac = False
    xtea = False
    rdp = False
    crc = False
  beacon_time = 2015-03-31 20:57:01
  beacon_flags = 121
  beacon = Container:
    obc = Container:
      boot_count = 573
      temp = ListContainer:
        -6.0
        -4.0
      panel_temp = ListContainer:
        0.0
        -28.5
        -26.75
        -13.25
        -28.25
        -20.0
  
```

The options used by the Telemetry parser are the following. The *Telemetry definition* option indicates the telemetry definition object, which must be an object in the `satellites.telemetry` module as described above. The *Output* drop down list can be used to select the standard output or a file as the destination for the parser's output. If a file is selected, an additional option to select the file path appears.

7.5.2 Telemetry submit

The telemetry submit block implements *Telemetry submission* to several different online telemetry servers. Its input consists of PDUs with frames, which are then submitted to the selected telemetry server.

This block uses the gr-satellites config file located in `~/gr_satellites/config.ini` to configure the different options of the telemetry servers, such as the login credentials. See the *information regarding the command line tool* for how to set up this configuration file.

The telemetry submit block has only one option, which is a drop down list that is used to select the telemetry server to use.

7.5.3 Hexdump sink

The hexdump sink prints PDUs in hex to the standard output. It is a wrapper over the Message Debug standard GNU Radio block, so it uses the same output format. This block is used internally by the `gr_satellites` command line tool (see *Hex dump*), and can also be used in custom flowgraphs instead of Message Debug.

7.5.4 KISS file sink

The KISS file sink can be used to store PDUs in a file using the [KISS protocol](#). This protocol is a simple format to mark frame boundaries. Files containing frames with the KISS protocol can then be read with the KISS file datasource (see *Data sources*) and with the `gr_satellites` command line tool (see *Specifying the input source*), as well as with external tools.

The KISS file sink block has two options. The *File* option is used to select the path of the output file. The *Append file* option can be used to overwrite or append to the output file.

The KISS files produced by the KISS file sink store timestamps as described in the *KISS output* of the `gr_satellites` command line tool.

7.5.5 KISS server sink

The KISS server sink spawns a TCP server that sends decoded PDUs to connected clients using the [KISS protocol](#). A number of tools can act as clients using this protocol.

The KISS file sink block has a *Port* option to specify the TCP port to listen on.

The KISS server sink sends timestamps as described in the *KISS output* of the `gr_satellites` command line tool.

7.5.6 File and Image receivers

The File and Image receiver blocks are used to reassemble files transmitted in chunks, using a variety of different formats. The only difference between the File receiver and the Image receiver is that the Image receiver is able to display image files in realtime using `feh` as they are being received.

These receiver blocks use *FileReceiver definitions*, which are classes derived from `FileReceiver`. The list of available definitions can be seen in `python/filereceiver/__index__.py`, or by calling `import satellites.filreceiver; help(satellites.filreceiver)` in python3. Classes used by the Image receiver must be derived from `ImageReceiver`.

The figure below shows an example flowgraph of the Image receiver block, which can be found in `examples/components/image_receiver.grc`. The example reads a WAV file from *satellite-recordings* containing an image transfer from LilacSat-1. The WAV file is played back in real time using the Throttle block. The Satellite decoder block is used to demodulate and deframe the packets. Since these packets contain a KISS stream, the KISS transport is used to obtain the image packets. These are sent into the Image receiver block, which will print some information to the standard output and when the beginning of the image is receive, will launch `feh` to display the image.



Fig. 18: Usage of Image receiver in a flowgraph

The figure below shows the options of the Image receiver block. The option *ImageReceiver class* indicates the definition to use for reassembling the image (which is implemented by a class derived from *ImageReceiver*). The *Path* option specifies the path of the directory where received files are saved to. The names of the files depend on metadata in the image packets. The *Verbose* option enables printing information to the standard output, such as the frames being received. The *Display* option enables the use of *feh* to display the image. The *Fullscreen* option is used to run *feh* in fullscreen.

General	Advanced	Documentation
ImageReceiver class	by70_1	
Path	/tmp/	
Verbose	True	
Display	True	
Fullscreen	True	
Command line options	""	

Fig. 19: Options of Image receiver

The options of the File receiver block are the same as those of the Image receiver block, except for the *Display* and *Fullscreen* options, which are specific to image reception.

7.5.7 Codec2 UDP sink

The Codec2 UDP sink is used internally by the `gr_satellites` command line tool when decoding LilacSat-1. The LilacSat-1 decoder supports outputting Codec2 digital voice frames by UDP. These frames can then be fed into the Codec2 command line decoder.

The Codec2 frames are 7 bytes long, and each is sent in a different UDP packet to ensure minimum latency.

The Codec2 UDP sink has two options, which indicate the IP and port to send the frames to. By default, address `127.0.0.1` and port `7000` are used.

The Codec2 frames can be decoded and played in real time by the Codec2 decoder as shown here.

```
$ nc -lu 7000 | c2dec 1300 - - | play -t raw -r 8000 -e signed-integer -b 16 -c 1 -
```

The `c2dec` command line decoder can be obtained by building from source the [codec2 library](#)

SatYAML files

SatYAML files are used by `gr-satellites` to describe the properties of each specific satellite, such as what kind of protocols and telemetry formats it uses. They are [YAML](#) files and are based around the concept of components. Using SatYAML files, the `gr_satellites` command line tool and the Satellite decoder block can figure out which components to put together to decode a particular satellite.

SatYAML files are stored in the `python/satyaml` directory. Below we show the SatYAML file `1KUNS-PF.yml` to give an overall idea of the format of these files.

```
name: 1KUNS-PF
alternative_names:
norad: 43466
data:
  &tlm Telemetry:
    telemetry: sat_1kuns_pf
  &image JPEG Images:
    image: sat_1kuns_pf
transmitters:
  1k2 FSK downlink:
    frequency: 437.300e+6
    modulation: FSK
    baudrate: 1200
    framing: AX100 ASM+Golay
    data:
      - *tlm
      - *image
  9k6 FSK downlink:
    frequency: 437.300e+6
    modulation: FSK
    baudrate: 9600
    framing: AX100 ASM+Golay
    data:
      - *tlm
      - *image
```

First we can see some fields that give basic information about the satellite. The `name` field indicates the main name

of the satellite, which is used by `gr_satellites` and the Satellite decoder block when calling up the satellite by name. There is an optional list of `alternative_names` which can also be used to call up the satellite by name. The `norad` field gives the NORAD ID of the satellite and it is used when calling up the satellite by NORAD ID.

Additional telemetry servers used for this satellite can be specified with the `telemetry_servers` field (see `python/satyaml/PW-Sat2.yml` for an example).

The `data` section indicates the different kinds of data transmissions that the satellite makes, and gives the decoders for them. The following can be used:

- `telemetry`, which specifies a telemetry decoder giving out the telemetry definition (see *Telemetry parser*)
- `file` or `image`, which specify a file receiver or image receiver giving out the `FileReceiver` or `ImageReceiver` class (see *File and Image receivers*)
- `decoder`, which specifies a custom decoder from `satellites.components.datasinks`. This is used for more complex decoders not covered by the above.
- `unknown`, which specifies that the data format is not known, so hex dump should be used to show the data to the user

The `transmitters` section lists the different transmitters used by the satellite, their properties, and ties them to the entries in the `data` section according as to which data is sent by each of the transmitters. A transmitter is understood as a specific combination of a frequency, modulation and coding.

Each transmitter has a name (such as `1k2 FSK downlink`) which is currently used only for documentation purposes, a `frequency`, which gives the downlink frequency in Hz (currently used only for documentation), a `modulation`, that specifies the demodulator component to use, a `baudrate`, in symbols per second, a `framing`, that specifies the deframer to use, and a list of `data` that has entries referring to the items in the `data` section.

The modulations allowed in the `modulation` field are the following:

- `AFSK`, for which the *AFSK demodulator* is used
- `FSK`, for which the *FSK demodulator* with `Subaudio` set to `False` is used
- `FSK subaudio`, for which the *FSK demodulator* with `Subaudio` set to `True` is used
- `BPSK`. Coherent BPSK, for which the *BPSK demodulator* with `Differential` and `Manchester` set to `False` is used
- `BPSK Manchester`. Coherent Manchester-encoded BPSK, for which the *BPSK demodulator* with `Differential` set to `False` and `Manchester` set to `True` is used
- `DBPSK`. Differentially-encoded BPSK, for which the *BPSK demodulator* with `Differential` set to `True` and `Manchester` set to `False` is used to perform non-coherent demodulation
- `DBPSK Manchester`. Differentially-encoded and Manchester-encoded BPSK, for which the *BPSK demodulator* with `Differential` and `Manchester` set to `True` is used to perform non-coherent demodulation

The AFSK modulation also needs the `deviation` and `af_carrier` fields that indicate the AFSK tone frequencies in Hz, as in the AFSK demodulator.

The framings allowed in the `framing` field are the following:

- `AX.25`, `AX.25` with no scrambling (see *AX.25 deframer*)
- `AX.25 G3RUH`, `AX.25` with G3RUH scrambling (see *AX.25 deframer*)
- `AX100 ASM+Golay`, GOMspace NanoCom AX100 in ASM+Golay mode (see *GOMspace AX100 deframer*)
- `AX100 Reed Solomon`, GOMspace NanoCom AX100 in Reed-Solomon mode (see *GOMspace AX100 deframer*)
- `U482C`, the GOMspace NanoCom U482C (see *GOMspace U482C deframer*)

- AO-40 FEC, the AO-40 FEC protocol (see *AO-40 FEC deframer*)
- AO-40 FEC short, AO-40 FEC protocol with short frames, as used by SMOG-P and ATL-1
- CCSDS Reed-Solomon, CCSDS Reed-Solomon TM codewords with conventional RS basis (see *CCSDS deframers*)
- CCSDS Reed-Solomon dual, CCSDS Reed-Solomon TM codewords with dual RS basis (see *CCSDS deframers*)
- CCSDS Reed-Solomon differential, CCSDS Reed-Solomon TM codewords with differential encoding and conventional RS basis (see *CCSDS deframers*)
- CCSDS Reed-Solomon dual differential, CCSDS Reed-Solomon TM codewords with differential encoding and dual RS basis (see *CCSDS deframers*)
- CCSDS Concatenated, CCSDS Concatenated TM codewords with conventional RS basis (see *CCSDS deframers*)
- CCSDS Concatenated dual, CCSDS concatenated TM codewords with dual RS basis (see *CCSDS deframers*)
- CCSDS Concatenated differential, CCSDS Concatenated TM codewords with differential encoding and conventional RS basis (see *CCSDS deframers*)
- CCSDS Concatenated dual differential, CCSDS Concatenated TM codewords with differential encoding and dual RS basis (see *CCSDS deframers*)
- CCSDS Reed-Solomon no-scrambler, CCSDS Reed-Solomon TM codewords with conventional RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Reed-Solomon dual no-scrambler, CCSDS Reed-Solomon TM codewords with dual RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Reed-Solomon differential no-scrambler, CCSDS Reed-Solomon TM codewords with differential encoding, conventional RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Reed-Solomon dual differential no-scrambler, CCSDS Reed-Solomon TM codewords with differential encoding, dual RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Concatenated no-scrambler, CCSDS Concatenated TM codewords with conventional RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Concatenated dual no-scrambler, CCSDS concatenated TM codewords with dual RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Concatenated differential no-scrambler, CCSDS Concatenated TM codewords with differential encoding, conventional RS basis and scrambler disabled (see *CCSDS deframers*)
- CCSDS Concatenated dual differential no-scrambler, CCSDS Concatenated TM codewords with differential encoding, dual RS basis and scrambler disabled (see *CCSDS deframers*)
- NASA-DSN Concatenated, CCSDS Concatenated TM codewords with conventional RS basis and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- NASA-DSN Concatenated dual, CCSDS concatenated TM codewords with dual RS basis and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- NASA-DSN Concatenated differential, CCSDS Concatenated TM codewords with differential encoding, conventional RS basis and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- NASA-DSN Concatenated dual differential, CCSDS Concatenated TM codewords with differential encoding, dual RS basis and NASA-DSN convolutional code convention (see *CCSDS deframers*)

- `NASA-DSN Concatenated no-scrambler`, CCSDS Concatenated TM codewords with conventional RS basis, scrambler disabled and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- `NASA-DSN Concatenated dual no-scrambler`, CCSDS concatenated TM codewords with dual RS basis, scrambler disabled and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- `NASA-DSN Concatenated differential no-scrambler`, CCSDS Concatenated TM codewords with differential encoding, conventional RS basis, scrambler disabled and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- `NASA-DSN Concatenated dual differential no-scrambler`, CCSDS Concatenated TM codewords with differential encoding, dual RS basis, scrambler disabled and NASA-DSN convolutional code convention (see *CCSDS deframers*)
- `3CAT-1`, custom framing used by 3CAT-1. This uses a CC1101 chip with PN9 scrambler and a (255,223) Reed-Solomon code for the payload
- `Astrocast FX.25 NRZ-I`, custom framing used by Astrocast 0.1. This is a somewhat non compliant `FX.25` variant.
- `Astrocast FX.25 NRZ`, custom framing used by Astrocast 0.1. This is a somewhat non compliant `FX.25` variant that is identical to the `FX.25 NRZ-I` mode except that NRZ is used instead of NRZ-I.
- `Astrocast 9k6`, custom framing used by Astrocast 0.1. It uses five interleaved Reed-Solomon (255,223) codewords and the CCSDS synchronous scrambler.
- `AO-40 uncoded`, uncoded AO-40 beacon. It uses 512 byte frames and a CRC-16
- `TT-64`, custom framing used by QB50 AT03, which uses a Reed-Solomon (64,48) code and CRC16-ARC
- `ESEO`, custom framing used by ESEO. It uses a custom protocol vaguely similar to `AX.25` with some form of G3RUH scrambling and a (255,239) Reed-Solomon code
- `Lucky-7`, custom framing used by Lucky-7, which uses a SiLabs Si4463 transceiver with a PN9 scrambler and a CRC-16
- `Reaktor Hello World`, custom framing used by Reaktor Hello World. It uses a Texas Instruments CC1125 transceiver with a PN9 scrambler and a CRC-16
- `S-NET`, custom framing used by S-NET, which uses BCH FEC and interleaving
- `Swiatowid`, custom framing used by Swiatowid for image transmission, which includes a (58,48) Reed-Solomon code and a CRC-16CCITT.
- `NuSat`, custom framing used by ÑuSat with a (64, 60) Reed-Solomon code and a CRC-8
- `K2SAT`, custom framing used by K2SAT for image transmission. This uses the CCSDS $r=1/2$, $k=7$ convolutional code and the IESS-308 (V.35) asynchronous scrambler.
- `LilacSat-1`, low latency decoder for LilacSat-1 codec2 digital voice and image data. This uses the CCSDS $r=1/2$, $k=7$ convolutional code and interleaved telemetry and Codec2 digital voice
- `AAUSAT-4`, custom framing used by AAUSAT-4, which is similar to the CCSDS Concatenated coding
- `NGHam`, `NGHam` protocol
- `NGHam no Reed Solomon`, `NGHam` protocol without Reed-Solomon, as used by FloripaSat-1
- `SMOG-P RA`, Repeat-Accumulate FEC as used by SMOG-P and ATL-1
- `SMOG-P Signalling`, custom signalling frames as used by SMOG-P and ATL-1
- `OPS-SAT`, custom framing used by OPS-SAT, which consists of `AX.25` frames with CCSDS Reed-Solomon codewords as payload

- UA01, non-AX.25 compliant framing used by QB50 UA01, which is like regular AX.25 but with two layers of NRZ-I encoding

Some framings, such as the CCSDS protocols need the additional field `frame size` to indicate the frame size.

The following example shows how transports are indicated in SatYAML files.

```
name: KS-1Q
norad: 41845
data:
  &t1m Telemetry:
    telemetry: csp
transports:
  &kiss KISS:
    protocol: KISS KS-1Q
    data:
      - *t1m
transmitters:
  20k FSK downlink:
    frequency: 436.500e+6
    modulation: FSK
    baudrate: 20000
    framing: CCSDS Concatenated dual
    frame size: 223
    transports:
      - *kiss
```

Instead of specifying a `data` entry in the transmitter, a `transports` entry is used instead. Transports are defined in a section above. They have a name, used for documentation purposes, a `protocol`, and a list of `data` entries to tie them with the appropriate data decoders.

The allowable transport protocols are the following:

- KISS, KISS protocol with a control byte (see *KISS transport*)
- KISS no control byte, KISS protocol with no control byte (see *KISS transport*)
- KISS KS-1Q, KISS variant used by KS-1Q, which includes a header before the KISS bytes

CHAPTER 9

Low level blocks

Low level blocks are the custom blocks offered by gr-satellites to implement the required functionality that is not available in the standard GNU Radio blocks. There are many different low level blocks, and some of them date back to the first versions of gr-satellites and have become somewhat outdated. A complete description of the low level blocks is outside the scope of the documentation for this version of gr-satellites.

It is likely that low level blocks will be classified and documented better in the future, perhaps deprecating some of the most outdated old blocks.

For the mean time, the user interested in learning how the different low level blocks can be used can explore which low level blocks are used by the components, by looking at the Python sources inside `python/components/`.

Some small utilities are included in `gr-satellites`. These are described below.

10.1 JY1SAT SSDV decoder

The JY1SAT SSDV decoder `jy1sat_ssdv.py` can be used to extract and decode SSDV images transmitted by JY1SAT. To use the decoder, an `ssdv fork` supporting the JY1SAT SSDV frame format needs to be installed. The decoder operates over a KISS file containing JY1SAT frames. The KISS file can be produced with the `--kiss_out` option of `gr_satellites` and might contain information for one or several images collected over one or several passes.

The decoder is run as

```
$ jy1sat_ssdv.py frames.kss /tmp/output
```

This will create files `/tmp/output_n.ssdv` with the extracted SSDV frames and `/tmp/output_n.jpg` with the decoded JPEG image data, where `n` is the number of the image.

10.2 SMOG-P spectrum plot

The SMOG-P spectrum plot tool `smog_p_spectrum.py` can be used to plot spectrum data files transmitted by SMOG-P and ATL-1. These files are produced by the *file receiver component*. The `smog_p_spectrum.py` script can be run by using the name of the spectrum data file as argument. For instance,

```
$ smog_p_spectrum.py spectrum_start_824000000_step_24000_rbw_6_measid_312
```

This will create an image `spectrum_312.png` in the same directory as the spectrum file (here 312 is the ID of the measurement, and is contained at the end of the spectrum file name).

Supported satellites

This is a list of all the satellites supported by gr-satellites. The list is auto-generated by reading the SatYAML files and using the script `docs/generate_supported_satellites.py`.

1KUNS-PF NORAD ID: 43466

Transmitters:

- **1k2 FSK downlink** (437.300 MHz): FSK modulation with AX100 ASM+Golay framing
- **9k6 FSK downlink** (437.300 MHz): FSK modulation with AX100 ASM+Golay framing

3CAT-1 NORAD ID: 43728

Transmitters:

- **9k6 FSK downlink** (437.250 MHz): FSK modulation with 3CAT-1 framing

3CAT-2 NORAD ID: 41732

Transmitters:

- **9k6 BPSK downlink** (145.970 MHz): BPSK modulation with AX.25 framing

AALTO-1 NORAD ID: 42775

Transmitters:

- **9k6 FSK downlink** (437.216 MHz): FSK modulation with AX.25 G3RUH framing

AAUSAT-4 NORAD ID: 41460

Transmitters:

- **2k4 FSK downlink** (437.425 MHz): FSK modulation with AAUSAT-4 framing
- **9k6 FSK downlink** (437.425 MHz): FSK modulation with AAUSAT-4 framing

ACRUX-1 NORAD ID: 44369

Transmitters:

- **9k6 FSK downlink** (437.200 MHz): FSK modulation with AX.25 G3RUH framing

AISAT NORAD ID: 40054

Transmitters:

- **4k8 AFSK downlink** (437.250 MHz): AFSK modulation with U482C framing

AISTECHSAT-2 NORAD ID: 43768

Transmitters:

- **4k8 FSK downlink** (436.730 MHz): FSK modulation with AX100 ASM+Golay framing
- **9k6 FSK downlink** (436.730 MHz): FSK modulation with AX100 ASM+Golay framing

AISTECHSAT-3 NORAD ID: 44103

Transmitters:

- **4k8 FSK downlink** (436.730 MHz): FSK modulation with AX100 ASM+Golay framing
- **9k6 FSK downlink** (436.730 MHz): FSK modulation with AX100 ASM+Golay framing

al-Farabi-2 Alternative names: UN1GWA

NORAD ID: 43805

Transmitters:

- **4k8 FSK downlink** (436.500 MHz): FSK modulation with AX.25 G3RUH framing

AmicalSat NORAD ID: 46287

Transmitters:

- **1k2 AFSK telemetry downlink** (436.100 MHz): AFSK modulation with AX.25 framing

AO-27 Alternative names: EYESAT-1, AO27

NORAD ID: 22825

Transmitters:

- **1k2 AFSK telemetry downlink** (436.795 MHz): AFSK modulation with AX.25 framing

AO-40 NORAD ID: 26609

Transmitters:

- **400baud uncoded BPSK beacon** (2400.200 MHz): DBPSK Manchester modulation with AO-40 uncoded framing
- **400baud FEC BPSK beacon** (2400.200 MHz): DBPSK Manchester modulation with AO-40 FEC framing

AO-73 Alternative names: FUNcube-1

NORAD ID: 39444

Transmitters:

- **1k2 BPSK downlink** (145.935 MHz): DBPSK modulation with AO-40 FEC framing

ARMADILLO NORAD ID: 44352

Transmitters:

- **19k2 FSK downlink** (437.525 MHz): FSK modulation with AX.25 G3RUH framing

Astrocast 0.1 NORAD ID: 43798

Transmitters:

- **1k2 FSK FX.25 NRZ-I downlink** (437.175 MHz): FSK modulation with Astrocast FX.25 NRZ-I framing
- **1k2 FSK FX.25 NRZ downlink** (437.175 MHz): FSK modulation with Astrocast FX.25 NRZ framing
- **9k6 FSK downlink** (437.175 MHz): FSK modulation with Astrocast 9k6 framing

Astrocast 0.2 Alternative names: HB9GSF

NORAD ID: 44083

Transmitters:

- **9k6 FSK downlink** (437.175 MHz): FSK modulation with AX.25 G3RUH framing
- **1k2 FSK FX.25 NRZ-I downlink** (437.175 MHz): FSK modulation with Astrocast FX.25 NRZ-I framing
- **1k2 FSK FX.25 NRZ downlink** (437.175 MHz): FSK modulation with Astrocast FX.25 NRZ framing

AT03 Alternative names: Pegasus, QB50 AT03

NORAD ID: 42784

Transmitters:

- **9k6 FSK downlink** (436.670 MHz): FSK modulation with TT-64 framing

ATHENOXAT-1 NORAD ID: 41168

Transmitters:

- **4k8 AFSK downlink** (437.485 MHz): AFSK modulation with U482C framing

ATL-1 Alternative names: MO-106

NORAD ID: 44830

Transmitters:

- **1k25 FSK long concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC framing
- **1k25 FSK short concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC short framing
- **1k25 FSK long RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **1k25 FSK short RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **1k25 FSK signalling** (437.175 MHz): FSK modulation with SMOG-P Signalling framing
- **2k5 FSK long concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC framing
- **2k5 FSK short concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC short framing
- **2k5 FSK long RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **2k5 FSK short RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **5k FSK long concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC framing
- **5k FSK short concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC short framing
- **5k FSK long RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **5k FSK short RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **12k5 FSK long concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC framing
- **12k5 FSK short concatenated FEC** (437.175 MHz): FSK modulation with AO-40 FEC short framing
- **12k5 FSK long RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing
- **12k5 FSK short RA FEC** (437.175 MHz): FSK modulation with SMOG-P RA framing

ATLANTIS Alternative names: US02 ON02US

NORAD ID: 42737

Transmitters:

- **9k6 FSK downlink** (436.388 MHz): FSK modulation with AX.25 G3RUH framing

AU02 Alternative names: QB50 AU02, UNSW-EC0

NORAD ID: 42723

Transmitters:

- **4k8 AFSK downlink** (436.525 MHz): AFSK modulation with U482C framing

AU03 Alternative names: QB50 AU03, i-INSPIRE II

NORAD ID: 42731

Transmitters:

- **4k8 AFSK downlink** (436.330 MHz): AFSK modulation with U482C framing

AztechSat-1 NORAD ID: 45258

Transmitters:

- **9k6 FSK downlink** (437.300 MHz): FSK modulation with AX100 ASM+Golay framing

BISONSAT Alternative names: N7SKC

NORAD ID: 40968

Transmitters:

- **9k6 FSK downlink** (437.375 MHz): FSK modulation with AX.25 G3RUH framing

BRICSat-2 Alternative names: USNA-P1, USNAP1, NO-103

NORAD ID: 44355

Transmitters:

- **1k2 AFSK downlink** (145.825 MHz): AFSK modulation with AX.25 framing
- **9k6 FSK downlink** (437.600 MHz): FSK modulation with AX.25 G3RUH framing

BUGSAT-1 Alternative names: TITA

NORAD ID: 40014

Transmitters:

- **9k6 FSK downlink** (437.445 MHz): FSK modulation with AX.25 G3RUH framing

BY02 Alternative names: BY70-2

NORAD ID: 45857

Transmitters:

- **9k6 BPSK downlink** (436.200 MHz): BPSK modulation with LilacSat-1 framing

BY70-1 NORAD ID: 41909

Transmitters:

- **9k6 BPSK downlink** (436.200 MHz): BPSK modulation with CCSDS Concatenated differential framing

CA03 Alternative names: QB50 CA03, ExAlta-1

NORAD ID: 42734

Transmitters:

- **4k8 FSK downlink** (436.705 MHz): FSK modulation with AX100 Reed Solomon framing
- **9k6 FSK downlink** (436.705 MHz): FSK modulation with AX100 Reed Solomon framing

CAS-4A NORAD ID: 42761

Transmitters:

- **4k8 FSK downlink** (145.836 MHz): FSK modulation with AX.25 G3RUH framing

CAS-4B NORAD ID: 42759

Transmitters:

- **4k8 FSK downlink** (145.893 MHz): FSK modulation with AX.25 G3RUH framing

CAS-6 Alternative names: TIANQIN-1

NORAD ID: 44881

Transmitters:

- **9k6 FSK downlink** (145.890 MHz): FSK modulation with AX.25 G3RUH framing

CHOMPTT NORAD ID: 43855

Transmitters:

- **9k6 FSK downlink** (437.560 MHz): FSK modulation with AX.25 G3RUH framing
- **1k2 AFSK downlink** (437.560 MHz): AFSK modulation with AX.25 framing

COLUMBIA Alternative names: US04, ON04US

NORAD ID: 42702

Transmitters:

- **9k6 FSK downlink** (437.055 MHz): FSK modulation with AX.25 G3RUH framing

CSIM-FD NORAD ID: 43793

Transmitters:

- **9k6 FSK downlink** (437.250 MHz): FSK modulation with AX.25 G3RUH framing

CubeBel-1 Alternative names: BSUSat-1

NORAD ID: 43666

Transmitters:

- **9k6 FSK downlink** (436.990 MHz): FSK modulation with AX.25 G3RUH framing

CUBEBUG-2 Alternative names: LO-74

NORAD ID: 39440

Transmitters:

- **9k6 FSK downlink** (437.445 MHz): FSK modulation with AX.25 G3RUH framing

CZ02 Alternative names: QB50 CZ0, VZLUSAT-1

NORAD ID: 42790

Transmitters:

- **4k8 AFSK downlink** (437.240 MHz): AFSK modulation with U482C framing

D-SAT NORAD ID: 42794

Transmitters:

- **4k8 AFSK downlink** (437.505 MHz): AFSK modulation with U482C framing

Delphini-1 NORAD ID: 44030

Transmitters:

- **4k8 FSK downlink** (437.500 MHz): FSK modulation with AX100 ASM+Golay framing
- **9k6 FSK downlink** (437.500 MHz): FSK modulation with AX100 ASM+Golay framing

DUCHIFAT-3 NORAD ID: 44854

Transmitters:

- **9k6 BPSK downlink** (436.400 MHz): BPSK modulation with AX.25 G3RUH framing

E-ST@R-II NORAD ID: 41459

Transmitters:

- **1k2 AFSK downlink** (437.485 MHz): AFSK modulation with AX.25 framing

Eaglet-I NORAD ID: 43790

Transmitters:

- **9k6 FSK downlink** (435.800 MHz): FSK modulation with AX.25 G3RUH framing

ECAMSAT NORAD ID: 43019

Transmitters:

- **1k2 AFSK downlink** (437.095 MHz): AFSK modulation with AX.25 framing

ELFIN-A Alternative names: WJ2XNX

NORAD ID: 43617

Transmitters:

- **19k2 FSK downlink** (437.450 MHz): FSK modulation with AX.25 G3RUH framing
- **9k6 FSK downlink** (437.450 MHz): FSK modulation with AX.25 G3RUH framing

ELFIN-B Alternative names: ELFIN-STAR, WJ2XOX

NORAD ID: 43616

Transmitters:

- **19k2 FSK downlink** (437.475 MHz): FSK modulation with AX.25 G3RUH framing
- **9k6 FSK downlink** (437.475 MHz): FSK modulation with AX.25 G3RUH framing

ENDUROSAT ONE Alternative names: ENDUROSAT AD

NORAD ID: 43551

Transmitters:

- **9k6 FSK downlink** (437.050 MHz): FSK modulation with AX.25 G3RUH framing

EntrySat NORAD ID: 44429

Transmitters:

- **9k6 BPSK downlink** (436.950 MHz): BPSK modulation with AX.25 G3RUH framing

ESEO Alternative names: FUNcube-4

NORAD ID: 43792

Transmitters:

- **9k6 FSK downlink** (437.000 MHz): FSK modulation with ESEO framing
- **4k8 FSK downlink** (437.000 MHz): FSK modulation with ESEO framing

FACSAT-1 NORAD ID: 43721

Transmitters:

- **9k6 FSK downlink** (437.350 MHz): FSK modulation with AX100 ASM+Golay framing

FALCONSAT-3 NORAD ID: 30776

Transmitters:

- **9k6 FSK downlink** (435.103 MHz): FSK modulation with AX.25 G3RUH framing

FIREBIRD 3 NORAD ID: 40377

Transmitters:

- **19k2 FSK downlink** (437.397 MHz): FSK modulation with AX.25 G3RUH framing

FIREBIRD 4 NORAD ID: 40378

Transmitters:

- **19k2 FSK downlink** (437.220 MHz): FSK modulation with AX.25 G3RUH framing

FloripaSat-1 NORAD ID: 44885

Transmitters:

- **1k2 FSK beacon** (145.900 MHz): FSK modulation with NGHAm no Reed Solomon framing
- **2k4 FSK downlink** (436.100 MHz): FSK modulation with NGHAm no Reed Solomon framing

FMN-1 Alternative names: FengMaNiu-1

NORAD ID: 43192

Transmitters:

- **9k6 BPSK downlink** (435.350 MHz): BPSK modulation with AX.25 G3RUH framing

GALASSIA NORAD ID: 41170

Transmitters:

- **4k8 AFSK downlink** (436.400 MHz): AFSK modulation with U482C framing

GO-32 Alternative names: TECHSAT-1B

NORAD ID: 25397

Transmitters:

- **9k6 FSK downlink A** (435.325 MHz): FSK modulation with AX.25 G3RUH framing

- **9k6 FSK downlink B** (435.225 MHz): FSK modulation with AX.25 G3RUH framing

GOMX-1 NORAD ID: 39430

Transmitters:

- **4k8 AFSK downlink** (437.250 MHz): AFSK modulation with U482C framing

GOMX-3 NORAD ID: 40949

Transmitters:

- **19k2 FSK downlink** (437.250 MHz): FSK modulation with AX100 Reed Solomon framing

GR01 Alternative names: QB50 GR01, DUTHSat

NORAD ID: 42724

Transmitters:

- **1k2 BPSK downlink** (436.420 MHz): BPSK modulation with AX.25 G3RUH framing
- **9k6 BPSK downlink** (436.420 MHz): BPSK modulation with AX.25 G3RUH framing

GRIFEX NORAD ID: 40379

Transmitters:

- **9k6 FSK downlink** (437.481 MHz): FSK modulation with AX.25 G3RUH framing

IL01 Alternative names: QB50 IL01, DUCHIFAT-2, Hoopoe

NORAD ID: 42718

Transmitters:

- **9k6 BPSK downlink** (437.740 MHz): BPSK modulation with AX.25 G3RUH framing

INNOSAT-2 NORAD ID: 43738

Transmitters:

- **4k8 FSK downlink** (437.450 MHz): FSK modulation with AX100 ASM+Golay framing

INS-1C NORAD ID: 43116

Transmitters:

- **1k2 FSK downlink** (435.080 MHz): FSK modulation with AX.25 framing

IRAZU Alternative names: Irazú

NORAD ID: 43468

Transmitters:

- **9k6 FSK downlink** (436.500 MHz): FSK modulation with AX.25 G3RUH framing

IRVINE-01 NORAD ID: 43693

Transmitters:

- **9k6 FSK downlink** (437.800 MHz): FSK modulation with AX.25 G3RUH framing

ITASAT 1 NORAD ID: 43786

Transmitters:

- **1k2 BPSK downlink** (145.860 MHz): BPSK modulation with AX.25 framing

JY1-Sat Alternative names: FUNcube-6, JO-97

NORAD ID: 43803

Transmitters:

- **1k2 BPSK downlink** (145.840 MHz): DBPSK modulation with AO-40 FEC framing

KR01 Alternative names: QB50 KR01, LINK

NORAD ID: 42714

Transmitters:

- **1k2 BPSK downlink** (436.030 MHz): BPSK modulation with AX.25 G3RUH framing
- **9k6 BPSK downlink** (436.030 MHz): BPSK modulation with AX.25 G3RUH framing

KrakSat Alternative names: SR9KRA

NORAD ID: 44427

Transmitters:

- **9k6 FSK downlink** (435.500 MHz): FSK modulation with AX.25 G3RUH framing

KS-1Q NORAD ID: 41845

Transmitters:

- **20k FSK downlink** (436.500 MHz): FSK modulation with CCSDS Concatenated dual framing

LightSail-2 Alternative names: WM9XPA, LightSail-B

NORAD ID: 44420

Transmitters:

- **9k6 FSK downlink** (437.025 MHz): FSK modulation with AX.25 G3RUH framing

LilacSat-1 Alternative names: CN02, QB50 CN02, LO-90

NORAD ID: 42725

Transmitters:

- **9k6 BPSK downlink** (436.510 MHz): BPSK modulation with LilacSat-1 framing

LilacSat-2 NORAD ID: 40908

Transmitters:

- **9k6 BPSK downlink** (437.200 MHz): BPSK modulation with CCSDS Concatenated differential framing
- **4k8 FSK downlink** (437.225 MHz): FSK modulation with CCSDS Concatenated framing
- **300baud subaudio downlink** (437.200 MHz): FSK subaudio modulation with CCSDS Reed-Solomon framing

LITUANICASAT-2 NORAD ID: 42768

Transmitters:

- **9k6 FSK downlink** (437.265 MHz): FSK modulation with AX.25 G3RUH framing

Lucky-7 NORAD ID: 44406

Transmitters:

- **4k8 FSK downlink** (437.525 MHz): FSK modulation with Lucky-7 framing

LUME-1 NORAD ID: 43908

Transmitters:

- **4k8 FSK downlink** (437.060 MHz): FSK modulation with AX100 ASM+Golay framing

Luojia-1 NORAD ID: 43485

Transmitters:

- **4k8 FSK downlink** (437.250 MHz): FSK modulation with AX100 ASM+Golay framing

M6P NORAD ID: 44109

Transmitters:

- **9k6 FSK downlink** (437.265 MHz): FSK modulation with AX.25 G3RUH framing

MCUBED-2 NORAD ID: 39469

Transmitters:

- **9k6 FSK downlink** (437.480 MHz): FSK modulation with AX.25 G3RUH framing

MINXSS NORAD ID: 41474

Transmitters:

- **9k6 FSK downlink** (437.345 MHz): FSK modulation with AX.25 G3RUH framing

MinXSS 2 NORAD ID: 43758

Transmitters:

- **9k6 FSK downlink** (437.250 MHz): FSK modulation with AX.25 G3RUH framing
- **19k2 FSK downlink** (437.250 MHz): FSK modulation with AX.25 G3RUH framing

MYSAT 1 NORAD ID: 44045

Transmitters:

- **1k2 BPSK downlink** (435.775 MHz): BPSK modulation with AX.25 G3RUH framing
- **9k6 BPSK downlink** (435.775 MHz): BPSK modulation with AX.25 G3RUH framing

Nayif-1 Alternative names: FUNcube-5, EO-88

NORAD ID: 42017

Transmitters:

- **1k2 BPSK downlink** (145.940 MHz): DBPSK modulation with AO-40 FEC framing

NEXUS Alternative names: JS1WAV, FO-99, Fuji-OSCAR 99

NORAD ID: 43937

Transmitters:

- **1k2 AFSK downlink** (435.900 MHz): AFSK modulation with AX.25 framing
- **9k6 FSK downlink** (435.900 MHz): FSK modulation with AX.25 G3RUH framing

NO-84 Alternative names: PSAT, ParkinsonSAT

NORAD ID: 40654

Transmitters:

- **1k2 AFSK downlink** (145.825 MHz): AFSK modulation with AX.25 framing

NODES 1 NORAD ID: 41478

Transmitters:

- **1k2 AFSK downlink** (437.100 MHz): AFSK modulation with AX.25 framing
- **19k2 FSK downlink** (2401.200 MHz): FSK modulation with AX.25 G3RUH framing

NODES 2 NORAD ID: 41477

Transmitters:

- **1k2 AFSK downlink** (437.100 MHz): AFSK modulation with AX.25 framing
- **19k2 FSK downlink** (2401.200 MHz): FSK modulation with AX.25 G3RUH framing

NSIGHT-1 Alternative names: AZ02 ON02AZ

NORAD ID: 42726

Transmitters:

- **9k6 FSK downlink** (435.900 MHz): FSK modulation with AX.25 G3RUH framing

NuSat 1 Alternative names: ÑuSat 1

NORAD ID: 41557

Transmitters:

- **40k FSK downlink** (436.445 MHz): FSK modulation with NuSat framing

O/OREOS Alternative names: USA 219

NORAD ID: 37224

Transmitters:

- **1k2 AFSK downlink** (437.305 MHz): AFSK modulation with AX.25 framing

OPS-SAT NORAD ID: 44878

Transmitters:

- **9k6 FSK downlink** (437.200 MHz): FSK modulation with OPS-SAT framing

PAINANI-1 NORAD ID: 44365

Transmitters:

- **9k6 FSK downlink** (437.475 MHz): FSK modulation with AX.25 G3RUH framing

PHOENIX Alternative names: TW01, ON01TW

NORAD ID: 42706

Transmitters:

- **9k6 FSK downlink** (436.915 MHz): FSK modulation with AX.25 G3RUH framing

PHONESAT 2.4 NORAD ID: 39381

Transmitters:

- **1k2 AFSK downlink** (437.425 MHz): AFSK modulation with AX.25 framing

PicSat NORAD ID: 43132

Transmitters:

- **1k2 BPSK downlink** (435.525 MHz): BPSK modulation with AX.25 G3RUH framing

- **9k6 BPSK downlink** (435.525 MHz): BPSK modulation with AX.25 G3RUH framing

POLYITAN-1 NORAD ID: 40042

Transmitters:

- **1k2 AFSK downlink** (437.675 MHz): AFSK modulation with AX.25 framing
- **9k6 FSK downlink** (437.676 MHz): FSK modulation with AX.25 G3RUH framing

PW-Sat2 NORAD ID: 43814

Transmitters:

- **1k2 BPSK downlink** (435.275 MHz): BPSK modulation with AX.25 G3RUH framing
- **9k6 BPSK downlink** (435.275 MHz): BPSK modulation with AX.25 G3RUH framing

QARMAN NORAD ID: 45257

Transmitters:

- **9k6 FSK downlink** (437.350 MHz): FSK modulation with AX.25 G3RUH framing

QBEE Alternative names: SE01, ON01SE

NORAD ID: 42708

Transmitters:

- **9k6 FSK downlink** (435.800 MHz): FSK modulation with AX.25 G3RUH framing

QO-100 Alternative names: Es'hail 2

NORAD ID: 43700

Transmitters:

- **400baud uncoded BPSK beacon** (10489.800 MHz): DBPSK Manchester modulation with AO-40 uncoded framing
- **400baud FEC BPSK beacon** (10489.800 MHz): DBPSK Manchester modulation with AO-40 FEC framing

Quetzal-1 NORAD ID: 45598

Transmitters:

- **4k8 FSK downlink** (437.200 MHz): FSK modulation with AX.25 G3RUH framing

Reaktor Hello World NORAD ID: 43743

Transmitters:

- **9k6 FSK downlink** (437.775 MHz): FSK modulation with Reaktor Hello World framing

ROBUSTA-1B NORAD ID: 42792

Transmitters:

- **1k2 AFSK downlink** (437.325 MHz): AFSK modulation with AX.25 framing

S-NET A Alternative names: DP0TB

NORAD ID: 43186

Transmitters:

- **1k2 AFSK downlink** (435.950 MHz): AFSK modulation with S-NET framing

Shaonian Xing Alternative names: MXSat-1

NORAD ID: 43199

Transmitters:

- **9k6 BPSK downlink** (436.375 MHz): BPSK modulation with AX.25 G3RUH framing

SiriusSat-1 Alternative names: RS13S

NORAD ID: 43595

Transmitters:

- **4k8 FSK downlink** (435.570 MHz): FSK modulation with AX.25 G3RUH framing

SiriusSat-2 Alternative names: RS14S

NORAD ID: 43596

Transmitters:

- **4k8 FSK downlink** (435.670 MHz): FSK modulation with AX.25 G3RUH framing

SKCUBE NORAD ID: 42789

Transmitters:

- **9k6 FSK downlink** (437.100 MHz): FSK modulation with AX.25 G3RUH framing

SMOG-P Alternative names: MO-105

NORAD ID: 44832

Transmitters:

- **1k25 FSK long concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC framing
- **1k25 FSK short concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC short framing
- **1k25 FSK long RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **1k25 FSK short RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **1k25 FSK signalling** (437.150 MHz): FSK modulation with SMOG-P Signalling framing
- **2k5 FSK long concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC framing
- **2k5 FSK short concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC short framing
- **2k5 FSK long RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **2k5 FSK short RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **5k FSK long concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC framing
- **5k FSK short concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC short framing
- **5k FSK long RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **5k FSK short RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **12k5 FSK long concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC framing
- **12k5 FSK short concatenated FEC** (437.150 MHz): FSK modulation with AO-40 FEC short framing
- **12k5 FSK long RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing
- **12k5 FSK short RA FEC** (437.150 MHz): FSK modulation with SMOG-P RA framing

SNUGLITE Alternative names: DS0DH

NORAD ID: 43784

Transmitters:

- **9k6 FSK downlink** (437.275 MHz): FSK modulation with AX.25 G3RUH framing

SpooQy-1 NORAD ID: 44332

Transmitters:

- **9k6 FSK downlink** (436.200 MHz): FSK modulation with AX100 ASM+Golay framing
- **4k8 FSK downlink** (436.200 MHz): FSK modulation with AX100 ASM+Golay framing

STRAND-1 Alternative names: STRaND-1

NORAD ID: 39090

Transmitters:

- **9k6 FSK downlink** (437.568 MHz): FSK modulation with AX.25 G3RUH framing

Suomi 100 NORAD ID: 43804

Transmitters:

- **9k6 FSK downlink** (437.775 MHz): FSK modulation with AX100 ASM+Golay framing

SwampSat-2 NORAD ID: 45115

Transmitters:

- **9k6 FSK downlink** (436.350 MHz): FSK modulation with AX.25 G3RUH framing

Swiatowid NORAD ID: 44426

Transmitters:

- **1k2 AFSK telemetry downlink** (435.500 MHz): AFSK modulation with AX.25 framing
- **9k6 FSK image downlink** (435.500 MHz): FSK modulation with Swiatowid framing

Tanusha-3 Alternative names: Tanusha-SWSU-3 (RS-8), RS8S

NORAD ID: 43597

Transmitters:

- **9k6 FSK downlink** (437.050 MHz): FSK modulation with AX.25 G3RUH framing
- **1k2 AFSK downlink** (437.050 MHz): AFSK modulation with AX.25 framing

Taurus-1 NORAD ID: 44530

Transmitters:

- **9k6 BPSK downlink** (435.840 MHz): BPSK modulation with LilacSat-1 framing

TBEX-A NORAD ID: 44356

Transmitters:

- **9k6 FSK downlink** (437.485 MHz): FSK modulation with AX.25 G3RUH framing

TBEX-B NORAD ID: 44359

Transmitters:

- **9k6 FSK downlink** (437.535 MHz): FSK modulation with AX.25 G3RUH framing

- **9k6 FSK downlink 2** (437.485 MHz): FSK modulation with AX.25 G3RUH framing

TIGRISAT NORAD ID: 40043

Transmitters:

- **9k6 FSK downlink** (435.000 MHz): FSK modulation with AX.25 G3RUH framing

TRISAT NORAD ID: 46280

Transmitters:

- **9766 baud FSK downlink** (435.612 MHz): FSK modulation with NASA-DSN Concatenated dual framing

TW-1A NORAD ID: 40928

Transmitters:

- **4k8 FSK downlink** (435.645 MHz): FSK modulation with AX100 Reed Solomon framing

TW-1B NORAD ID: 40927

Transmitters:

- **4k8 FSK downlink** (437.645 MHz): FSK modulation with AX100 Reed Solomon framing

TW-1C NORAD ID: 40926

Transmitters:

- **4k8 FSK downlink** (435.645 MHz): FSK modulation with AX100 Reed Solomon framing

TY 4-01 NORAD ID: 43669

Transmitters:

- **9k6 FSK downlink** (435.925 MHz): FSK modulation with AX100 ASM+Golay framing

TY-2 NORAD ID: 43155

Transmitters:

- **9k6 FSK downlink** (435.350 MHz): FSK modulation with AX100 ASM+Golay framing

TY-6 NORAD ID: 43158

Transmitters:

- **9k6 FSK downlink** (436.100 MHz): FSK modulation with AX100 ASM+Golay framing

UA01 Alternative names: PolyITAN 2-SAU, QB50 UA01

NORAD ID: 42732

Transmitters:

- **9k6 FSK downlink** (436.600 MHz): BPSK modulation with UA01 framing

UBAKUSAT NORAD ID: 43467

Transmitters:

- **9k6 FSK downlink** (437.325 MHz): FSK modulation with AX.25 G3RUH framing

UCLSAT NORAD ID: 42765

Transmitters:

- **9k6 FSK downlink** (435.975 MHz): FSK modulation with AX.25 G3RUH framing

UKube-1 Alternative names: FUNcube-2

NORAD ID: 40074

Transmitters:

- **1k2 BPSK downlink** (145.840 MHz): DBPSK modulation with AO-40 FEC framing

UNISAT-6 NORAD ID: 40012

Transmitters:

- **9k6 FSK downlink** (437.421 MHz): FSK modulation with AX.25 G3RUH framing

UPMSat 2 NORAD ID: 46276

Transmitters:

- **1k2 FSK telemetry downlink** (437.405 MHz): FSK modulation with AX.25 framing

URSA MAIOR Alternative names: IT02

NORAD ID: 42776

Transmitters:

- **9k6 FSK downlink** (435.950 MHz): FSK modulation with AX.25 G3RUH framing

US01 Alternative names: Challenger, QB50 US01, QBUS 1

NORAD ID: 42721

Transmitters:

- **9k6 FSK downlink** (437.505 MHz): FSK modulation with AX.25 G3RUH framing

UWE-3 NORAD ID: 39446

Transmitters:

- **1k2 AFSK downlink** (437.385 MHz): AFSK modulation with AX.25 framing
- **9k6 FSK downlink** (437.384 MHz): FSK modulation with AX.25 G3RUH framing

UWE-4 Alternative names: DP0UWH

NORAD ID: 43880

Transmitters:

- **9k6 FSK downlink** (435.600 MHz): FSK modulation with AX.25 G3RUH framing

X-CUBESAT Alternative names: FR01, ON01FR

NORAD ID: 42707

Transmitters:

- **9k6 FSK downlink** (437.020 MHz): FSK modulation with AX.25 G3RUH framing
- **1k2 AFSK downlink** (437.020 MHz): AFSK modulation with AX.25 framing

XW-2A Alternative names: CAS-3A

NORAD ID: 40903

Transmitters:

- **9k6 FSK downlink** (145.640 MHz): FSK modulation with AX.25 G3RUH framing

XW-2B Alternative names: CAS-3B

NORAD ID: 40911

Transmitters:

- **9k6 FSK downlink** (145.705 MHz): FSK modulation with AX.25 G3RUH framing

XW-2C Alternative names: CAS-3C

NORAD ID: 40906

Transmitters:

- **19k2 FSK downlink** (145.770 MHz): FSK modulation with AX.25 G3RUH framing

XW-2D Alternative names: CAS-3D

NORAD ID: 40907

Transmitters:

- **9k6 FSK downlink** (145.835 MHz): FSK modulation with AX.25 G3RUH framing

XW-2E Alternative names: CAS-3E

NORAD ID: 40909

Transmitters:

- **9k6 FSK downlink** (145.890 MHz): FSK modulation with AX.25 G3RUH framing

XW-2F Alternative names: CAS-3F

NORAD ID: 40910

Transmitters:

- **9k6 FSK downlink** (145.955 MHz): FSK modulation with AX.25 G3RUH framing

ZACUBE-1 Alternative names: South Africa CubeSat-1, TshepisoSat, ZA003

NORAD ID: 39417

Transmitters:

- **9k6 FSK downlink** (437.356 MHz): FSK modulation with AX.25 G3RUH framing

Zhou Enlai NORAD ID: 43156

Transmitters:

- **9k6 BPSK downlink** (436.420 MHz): BPSK modulation with AX.25 G3RUH framing

CHAPTER 12

Indices and tables

- `genindex`
- `modindex`
- `search`