

SigMF: The Signal Metadata Format

Ben Hilburn




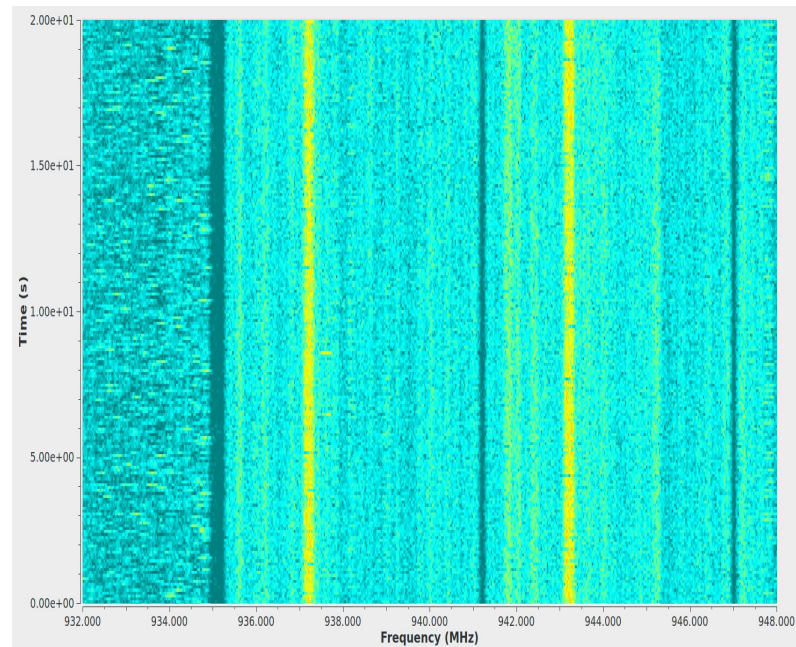
Problem Statement

How do you make large recordings of samples more practically useful?



Signal Metadata Format

- Format for describing recordings of digital samples.
- Why is this useful?
 - Don't need hardware
 - Signals you don't have access to
 -  Reproducibility (for science!)
 - Collaborative processing
 - Basically “code comments” for signal data
 - Create feature / characteristic annotations
 - Moving data between tools/workflows and retaining meta-information



Example Usage Scenario

A receiver will monitor some piece of the spectrum and record the raw data, which will then be processed by some signal detection & classification analysis engine, the results of which will be visualized by a human operator in a GUI.

- Historically, this sort of system would need to be monolithic to guarantee that each block could understand the data from the blocks upstream. This limits the operator's ability to update pieces of the system or analyze the data with different tools. It is effectively system-lock-in, making the data useless outside of that one system.
- SigMF solves this problem by defining an open standard for the recording, including the raw data and metadata.

The SigMF Standard

- Open-Source Standard
 - Itself released under CC-BY-SA license
- Specification format loosely based on IETF RFC formats.
- Metadata is written with JSON
 - Portability!
 - Also, readability.
- This is **not just for GNU Radio**.
- The goal is create something that allows moving datasets between tools and workflows without a loss or corruption of information.

Types of Applications Defined by the Standard

- “Writers” are applications that are writing **metadata**.
 - Must be able to do this in a streaming fashion
 - Must be able to do this without filling up your disk with useless data.

- “Readers” are applications that are reading **metadata and samples**.
 - Must be able to do this in a streaming fashion.
 - Must know, unambiguously, what metadata applies to what samples.

Driving Principles

- Make it as straight-forward as possible for developers to create applications, translation libraries, and integrate the format into existing systems.
 - Define a core namespace, but allow extensions for users to expand the scope of the metadata.
 - e.g., if you are building a detector and want to specify custom fields, you can simply create a `detector` namespace for new fields
- Guarantee that metadata written with a compliant writer application is useful in a compliant reader application.
 - Means we must define what “compliance” means for both applications as well as the data & metadata.
- Keep things simple and easy to understand.

A SigMF Recording

- A SigMF “Recording” of one flat data file and one flat metadata file
 - The data file is just samples
 - The metadata file is just JSON

- Recordings can be stored & distributed in an archive format.
- Archives have a defined directory structure for including multiple Recordings

High-Level Structure of a SigMF Metadata File

- Fundamentally, there are three questions you need to answer to be able to understand the data and metadata:
 - How do I understand these files?
 - How do I understand these samples?
 - How do I understand this metadata?

- SigMF has three top-level JSON objects to answer these three questions.

How do I understand these files?

- **Global** - General information about the file. The minimal information needed to parse the dataset file. Example fields:
 - **Datatype:** How are the samples stored?
 - **Sample Rate:** What is the sample rate at which this data was recorded?
 - **Version:** What version of the SigMF standard is in use?
 - **Author:** Who created these files?
 - **License:** What is the license of this data?
 - **Hash:** A hash of the data to provide proof of integrity.
 - etc.,

How do I understand these samples?

- **Global** - General information about the file.
- **Captures** - An array of segments that describe the parameters of the capture, starting at a certain sample index. Example fields:
 - **Center Frequency:** At what frequency was the radio tuned to during the capture?
 - **Timestamp:** What is the timestamp of a particular sample index?
 - etc.,

How do I understand the metadata?

- **Global** - General information about the file.
- **Captures** - An array of segments that describe the parameters of the capture.
- **Annotations** - An array of segments that describe features or provides comments about the signal.
 - Annotations are meant to be as flexible as possible.
 - “detected interference here”
 - “classified modulation as QAM64”
 - “cat jumped on antenna”
 - etc.,

Mapping Metadata to Samples

- Each piece of metadata is mapped to samples using the most fundamental unit in the recording: **Sample Index**
- Metadata that applies to many samples provides both the starting index at which the metadata applies, as well as the number of samples it applies to.
- Active Development: Annotations in frequency

Continuously Varying Fields

- Dealing with fields that are continuously changing can be a significant challenge for metadata.
 - Example: If your receiver is in a vehicle, how do you record the changing geolocation in a useful way?
 - Example: If your antenna is a spinning RADAR dish, how do you record the changing azimuth of your aperture?
- SigMF makes this easy: these continuously variably fields are just another SigMF recording!
 - (This feature is still under active development, and is not yet reflected in the master spec)
 - (Also under development is using this feature for continually varying sample rates)

```

5 {
6   "global": {
7     "core:datatype": "cf32",           # The datatype of the recording (here, complex 32-bit float)
8     "core:sample_rate": 100000000,    # The sample rate of the recording (10 MHz, here).
9     "core:version": "0.0.1",          # Version of the SigMF spec used.
10    "core:description": "An example metadafile for a SigMF recording.",
11  },
12
13  "capture": [
14    # The `capture` object contains a list of segments, sorted by the `sample_start` value
15    {
16      "core:sample_start": 0,          # The sample index that these parameters take effect.
17      "core:frequency": 900000000,    # The center frequency of the recording (900 MHz, here).
18      "core:time": "2017-02-01T11:33:17,053240428+01:00",
19    },
20    {
21      "core:sample_start": 100000,     # Mandatory
22      "core:frequency": 950000000,    # Now at 950 MHz
23    },
24  ],
25
26  "annotations": [
27    # The `annotations` object contains a list of segments, sorted by the `sample_start` value
28    {
29      "core:sample_start": 1000000,   # The sample index at which this annotation first applies.
30      "core:sample_count": 120000,    # The number of samples that this annotation applies to.
31      "core:comment": "Some text comment about stuff happening",
32    },
33  ],
34 }

```

Why didn't you use <X>?

- VITA49
 - The packet header-based format makes it impossible to separate the metadata from the data for modification, inspection, a streaming without also interacting with the data.
 - Some fields cannot be continuously varying with VITA49 (e.g., sample rate)
 - Everyone seems to have their own implementation where they have cherry-picked the parts of VITA49 they need, and no one agrees on what VITA49 actually is. This breaks portability of datasets.
- HDF5
 - The HDF5 standard is highly generalized, and as a result of the large scope it is much more complex in terms of the standard, readers, writers, and structure of the data.
 - Because of the generality, there is no definition or agreement on what it would mean to create a “compliant” application of metadata for signals. i.e., we could not guarantee that portability.

Progress

- Draft of the standard is live on Github
 - Actively discussing and making changes using Issues and Pull Requests
- Python validator for SigMF files
- Alpha web front-end for the validator
- Already have involvement from numerous other communities, organizations, and authors of non-GNU Radio tools.
- Things to do:
 - Give us feedback, ask questions, and help us improve the standard!
 - GNU Radio Source / Sink blocks (good example implementations)
 - Static Visualization Tool
 - Make a slick logo!
 - Proselytize!

Come get involved!

<https://github.com/gnuradio/SigMF>