SigMF: The Signal Metadata Format

Ben Hilburn BHILBURN@DEEPSIG.IO
Nathan West NWEST@DEEPSIG.IO
Tim O'Shea TIM@DEEPSIG.IO
Tamoghna Roy TROY@DEEPSIG.IO

Abstract

The Signal Metadata Format (SigMF) specifies a way to describe sets of recorded digital signal samples with metadata written in plaintext, structured with JSON. SigMF can be used to describe general information about a collection of samples, the characteristics of the system that generated the samples, and features of the signal itself. It is designed to be a simple and portable format that is easily used by memory-limited applications with real-time requirements and minimal dependencies.

1. Introduction

Sharing sets of recorded signal data is an important part of science and engineering. It enables multiple parties to collaborate, is often a necessary part of reproducing scientific results (a requirement of scientific rigor), and enables sharing data with those who do not have direct access to the equipment required to capture it. Unfortunately, these datasets have historically not been very portable, and there is not currently a widely accepted format for the sharing of arbitrary metadata attached to recordings of streaming digital samples. This is the problem that SigMF solves (sig, 2018).

By providing a standard way to describe data recordings, SigMF facilitates the sharing of data, prevents the "bitrot" of datasets wherein details of the capture are lost over time, and makes it possible for different tools to operate on the same dataset, thus enabling data portability between tools and workflows.

Furthermore, SigMF provides a format that is readily usable in machine learning. The nature of its design and annotation capability make it very simple to use for creating, labeling, and training machine learning models on recordings, which has become a critical area of research in the field of wireless communications.

Proceedings of the 3^{rd} GNU Radio Conference, Copyright 2018 by the author(s).

The SigMF specification is readily available, provided under a Creative Commons license, at `sigmf.org`, and this paper is not meant to be a re-hash of the details in that document. Rather, this paper will discuss the design goals of SigMF, expand on the major design decisions of the specification, introduce some of the other options available, and discuss the future direction of SigMF development.

2. Motivation

While SigMF is generic enough to be used for any recording of a digitized time-series signal, the original motivation was improving the accessibility of wireless RF signal recordings. The explosive success of Software-Defined Radios (SDR) in the early & mid 2000s made it relatively easy to create digital recordings of RF signals and process them with software. This gave rise to a large and vibrant industry of both hardware and software vendors, including numerous open-source software and hardware projects, enabling entirely new approaches for research and engineering (Akeela & Dezfouli, 2018).

Unfortunately, there never emerged a dominant standard for these recordings. As a result, recordings made with a specific software tool or framework have not been easily portable to others and it is often difficult for many users to post-process the same dataset and merge the generated metadata, even using the same software workflow.

The advancing state of SDR technology creates difficulties as well. For example, the Analog-to-Digital Converters (ADC) available in commodity SDRs are now so fast that many systems struggle to keep up with the required write-speed without "dropping" data (i.e., data is being generated too quickly to write to disk and the system buffers are full, so data is lost before it can be stored). The ADCs also produce data at such rates that creating a recording of even a few seconds worth of a signal capture is many gigabytes in size, complicating the storage and sharing of the files.

The topics of storage, portability, and sharing of RF datasets was a problem posed at the 2017 DARPA Brussels Hackfest (DARPA, 2017). A working group formed around the discussion, and over the course of the hackfest laid out the fundamental design principles and an early prototype of the SigMF specification.

3. Design Requirements

The original goal of the SigMF effort was to create a storage format for digital recordings. This is different from a packet format, for example, which is designed for optimal transmission over computer networks, or a data format that might describe how something is represented while it's being processed by a program. A storage format is specifically designed to describe how data is stored and represented when it has been written to disk and is "at rest".

The working group drafted requirements for an ideal solution to the RF dataset problem, as posed, which became the design principles behind SigMF. Those principles are:

1. Describes a storage format that maximizes the speed at which computer programs can both read & write the recording to/from storage in a streaming fashion.
2. Defines a standard that guarantees the ability to parse & process a dataset by any program that meets the specification's requirements for compliance.
3. Minimizes the requirements (e.g., complexity of code logic, software dependencies) to read or write a recording by computer programs that are compliant with the specification.
4. Provides a method for linking many time-variant fields to a recording (e.g., recording the location of a moving receiver or the changing azimuth of a rotating antenna aperture).
5. Enables the sharing of data without sharing the entirety of the recording (e.g., sharing only a portion of the metadata without the signal samples).
6. Allows for arbitrary metadata not defined by the specification.
7. Facilitates easy integration with, and does not preclude use by, existing software tools, frameworks, and workflows.
8. Represents the metadata in a way that is easily parseable, searchable, and indexable, including within a computer database, and that can be easily rendered in a human-readable & editable form.
9. Can be specified by a standards document that is simple and easily understandable.
10. Is governed and maintained by an open-source software development process.

4. Design Decisions

Based on the requirements above, the working group made some top-level design decisions.

To maximize the speed at which data can be written to disk, the 'data' portion of a SigMF recording is quite simply the raw data stream - there is no inline metadata or file structure. This enables a "writer" program to record information as fast as possible, without requiring processing steps or jumping around on-disk. For a reader, this enables a program to simply read data off of the disk, in-order, as it is needed. In many cases, this precludes the need to read large chunks of data into memory by software applications, which can be a significant issue on low-SWaP embedded systems; the data stays on-disk until it is directly needed for processing. Since the data can be represented on-disk with the same type and format that it is represented with in-memory, no data formatting is required to load & store - the user can actually directly map the SigMF recording on-disk to memory.

The SigMF specification defines what it means to be "compliant", both for applications and for a SigMF recording. This is done such that there can be a guarantee of comprehension: for a given SigMF release version, if one application writes a recording and another application reads it, and both applications are *compliant*, the user is guaranteed that the data will be understood by the two programs in exactly the same way.

With the goal of making it as simple as possible to integrate support for SigMF into applications, it is designed such that it can be implemented with minimal software dependencies or calls to external programs. In fact, using most modern programming languages, no external software dependencies are required to use and interact with a SigMF recording at all.

Recording metadata about a signal that is continuously changing can be a significant challenge, and causes substantial pollution and bloat within other dataset formats. A great example is a radio receiver that is in a moving vehicle - in addition to the continuous & time-varying signal that is being received by the radio, the location of the radio itself is continuously changing. Attempting to record the geolocation with many existing formats requires either (a) reducing the update rate of the location or (b) complicating the recording of the samples significantly. A major design decision of SigMF was to treat such fields as "just another signal" - e.g., in the example above, the continuous & time-varying geolocation is just another signal that is recorded with SigMF, and then linked to the signal recording sampled by the radio receiver. By doing this, the precision of the geolocation isn't tied to the update rate of the RF signal, recording the geolocation doesn't interfere with reading / writing the recorded signal, and the recordings of the signal and geolocation are separable and can processed or shared without the other. This design decision has enabled users to record many time-varying metadata aspects of signals in very complex experiments while maintaining a simple and easy-to-process recording.

The SigMF specification splits the data and metadata into separate files, linking them by index, thus allowing the two pieces - the raw data and the information describing it - to be processed without needing to read or share the counterpart.

Especially since one of the goals of SigMF was to enable collaboration on datasets for new applications, the working group knew that the standard must allow for the creation of arbitrary metadata. The SigMF specification provides for this directly in recordings, and furthermore has a canonical method to extend the specification with user-specific fields while still maintaining the concept of *compliance* with the SigMF.

The SigMF working group was deeply rooted in the Open-Source Software (OSS) community, and the design requirement of enabling use of existing tools is a major reflection of that. Making it possible to interact with SigMF recordings using existing tools in the OSS ecosystem makes the datasets immensely easier to work with and integrate. Based on this, the working group decided to require that the binary samples were recorded in industry-standard datatypes supported on POSIX systems, and that the metadata must be written in plaintext. This makes it immediately possible to manage, process, and edit the metadata with existing OSS tools, including thousands of popular editors, command-line programs, and software workflows. Put differently, SigMF metadata can be treated in the same way that programmers handle and process software code.

Once the decision was made that the metadata must all be plaintext, the working group decided to use JSON to organize the data. JSON has built-in support in most modern software languages and workflows, creates a structure that is easy to search, is translatable to databases, and is also one of the easier metadata formats to read directly by humans (admittedly, this is subjective).

Many standards documents become so complex that navigating them can be a serious challenge, which acts as a barrier to both users and developers. By keeping the specification short and as simple as possible, the SigMF standard is not only approachable to new users but easy to understand by developers seeking to implement support for SigMF in their applications. Keeping the specification simple has the additional benefit of reducing the possibility of incorrect datasets as there is almost no redundancy of information. For example, fields that can be derived directly from the data (e.g., the 'length' of the data) are not described in the metadata, as that information makes it possible to create conflict; if there was a 'length' field that disagreed with the actual length of the data, specifying which takes precedence requires growing the spec with edge-case handling. By keeping it as simple as possible, many of these tyes of issues can be avoided entirely with minimal impact to implementation.

Lastly, many standards are drafted and maintained in a way that can be detrimental to the standard itself. Some examples are closed-door discussions, power over the specification granted to paying committee members, or even specifications that are only available to customers that pay for them. The SigMF effort was born out of the OSS community, and its governance and development reflects any other open-source project - everyone is welcome to contribute, and no one person's input or feedback automatically trumps anyone else's - the success of SigMF is always prioritized. This is made easier by the 'extension namespace' mechanism, mentioned before, whereby if something is not accepted in the specification it can still be used (but won't be required for compliance).

5. Other Solutions

There are many different formats and standards for data storage, and even more specifically RF sample storage. We will briefly discuss some of those most prominently used for working with RF data.

5.1. VITA-49

VITA-49 is one of the most widely used formats, and is supported by many SDR hardware and processing IP vendors. The VITA-49 standard (VITA) is a packet format for RF samples and associated metadata, and is most commonly used for sending samples over a data transport (e.g., Ethernet).

While VITA-49 is popular and enjoys widespread support with existing hardware and tools, it solves a different problem than what the working group set out to address. It's fundamentally designed to be a transport format rather than a storage format, and so doesn't meet many of the requirements described above. For example, as a packet format, metadata is stored in-line with the data in a packet header - while this is perfectly sensible for VITA-49's goals, it breaks many of the requirements for SigMF. A full radio system needs both a transport format and a storage format, and the working group envisions standards like VITA-49 being used in conjunction with SigMF; the former for data "in-motion", and the latter for data "at-rest".

5.2. HDF5

HDF5, or ”Hierarchical Data Format version 5” (Group), is a popular dataset format in the scientific community, and has widespread support in many scientific and engineering tools. It is used for many applications (not just signals), and is supported by The HDF Group, which provides commercial products and services.

HDF5 is a generic dataset format, though, and does not provide the concept of ”compliance” as defined by the SigMF specification. Put differently, it doesn’t define what a recording of signal data must look like, and thus cannot guarantee that two programs, both with HDF5 support, will be able to read and operate on an RF dataset with the same understanding of the information in the recording.

5.3. Digital RF

The Digital RF project (Haystack) is based on HDF5, and solves the problem of ”compliance” by effectively creating a schema for RF datasets written in the HDF5 format. Digital RF is authored and maintained by the MIT Haystack Observatory, and is used for many of their radio science experiments.

Of the alternatives, Digital RF is probably the closest to SigMF in terms of its goals, but its design requirements are markedly different from SigMF and thus differs significantly in its design decisions. Which is best in a particular scenario will depend on the requirements of the application and the goals of the user - in some cases, ’Digital RF’ will be better, and in others, SigMF; the decision of which is a technical design decision to be made by the user. As a goal of SigMF is interoperability between tools and workflows, though, SigMF does need translators to & from ’Digital RF’ to make it simple for users to leverage both where appropriate.

5.4. pickle

’pickle’ (Foundation) provides serialization and deserialization of data for storage on-disk. It is specific to Python, and is only supported within the Python language ecosystem. It also doesn’t support many of the mechanisms for metadata needed by SigMF, and so is generally not appropriate for all but the most simple of recordings that will only be used within Python programs.

5.5. Midas BLUE

The Midas BLUE format (Research) is a data storage format written for use with the Midas Framework and is used in a variety of high-performance radio applications, mostly by the U.S. government. Its design goals are dramatically different from those of SigMF’s, as it was originally created specifically to be used with the Midas software workflow. It now has support in other software workflows that aren’t related to Midas, but it has not seen widespread adoption outside of it’s existing community.

6. Usage Patterns

A major advantage of SigMF is the ease of tooling since metadata is detached in human-readable form and datasets can be organized by directory structures. Using filesystem symbolic links allows raw data to be stored independent of metadata which enables tracking metadata changes without parsing large binary captures. Using a Version Control System (e.g., ’git’) with pre-commit hooks and JSON pretty printers (Hodges) that are widely available enables meaningful change tracking of annotations. Storing raw files separately from annotations and using symbolic links to form compliant SigMF recordings also allows flexibility with recorded datasets that will be annotated - this reduces drive usage and eases synchronizing SigMF captures across distributed storage. This queuing system, relying on minimal data movement and standard filesystem tools, has proven to be a valuable dataset generation approach.

At minimum, a simple JSON structure must be parsed to understand the datatype of the stored samples. This can be automated by tools parsing the datatype field in the metadata or in a one-off fashion with a human reading the structure and running type-specific tools. Using standardized POSIX interfaces such as ’memmap’ (POSIX.1-2017) allows lazy reading of sample structures which significantly reduces the load time of samples since only the indexed samples need to be loaded. Memory-mapping functionality is available from all common operating systems and most programming languages. For example, NumPy has a ’memmap’ call that exposes a memmapped file as a NumPy array with the correct datatype, and C/C++ exposes the file as a pointer which is typically typecast to whatever the underlying sample type is.

7. Future of SigMF

Since it was first published in early 2017, SigMF has seen rapid adoption by government, industry, and academia. It’s simplicity and flexibility have made it an attractive option for many applications. Just to name a few, The National Telecommunications and Information Administration (NTIA), an entity of the US federal government, is using SigMF for spectrum sensing (NTIA); SkySafe, a commercial counter-drone company, is using it for their RF systems (SkySafe); In-Q-Tel, a technology research and investment firm is using it for highly complex experiments with many sensors (Mohan et al., 2018); and DeepSig (the authors’ employer), a commercial machine learning company, is using it for deep learning on RF data (DeepSig).

7.1. Development

While SigMF is currently shepherded by the GNU Radio Foundation, it is meant to be a format accessible and usable by anyone. Indeed, the specification has received considerable contributions from people developing and using completely different tools and workflows. All development activity, including changes and discussion, occurs on the SigMF GitHub repository (`sigmf.org`), and we welcome involvement from anyone that wants to participate.

Issues are raised and discussed as Github 'Issues' in the repository, and contributions are discussed as 'Pull Requests', thereby making these discussions not only open and accessible but preserved for posterity. Since SigMF is maintained as a git repository, all changes and releases are tracked and can be individually referenced.

7.2. Current State

As of this paper, the SigMF project has made one release, 'Release v0.0.1' (Hilburn, 2018), and is now working towards the second release. It is using Semantic Versioning (SemVer 2.0.0), and since all current releases are below '1.0.0', SigMF is currently considered 'unstable'. This means that there is currently no guarantee of backwards compatibility until the project reaches the v1.0.0 stable release. That said, the SigMF contributors are already working to preserve backwards compatibility where possible, and compatibility is already guaranteed for applications & datasets adhering to the same version (e.g., 'v0.0.1'), per the specification for *compliance*.

Some of the software-related goals of the SigMF project are to provide translators for other common formats (like those listed above in this paper), validators (for assuring that recordings are compliant to the specification), and utilities for managing and working with SigMF recordings. Some code contributions have already been made along these lines, but much more development is needed in this space.

It is not the goal of the SigMF project to dictate how SigMF data is used within applications (when the data is not at-rest), and so framework-specific blocks (e.g., a GNU Radio SigMF Source block) will never be part of the SigMF repository.

7.3. Getting Involved

SigMF welcomes participation from everyone, and works to maintain a friendly and accessibly developer community. If you have questions or feedback, feel free to post an Issue on the repository or contact the maintainers directly!

References

Signal metadata format specification. `http://sigmf.org/`, 2018.

Akeela, Rami and Dezfouli, Behnam. Software-defined radios: Architecture, state-of-the-art, and challenges. *CoRR*, abs/1804.06564, 2018. URL `http://arxiv.org/abs/1804.06564`.

DARPA. Darpa brussels hackfest. `https://darpahackfest.com/past-hackfest#DARPA-Brussels-Hackfest`, 2017.

DeepSig. Datasets for radio deep learning.

Foundation, Python. pickle. `https://docs.python.org/3/library/pickle.html`.

Group, The HDF. Hdf5. `https://support.hdfgroup.org/HDF5/doc/H5.intro.html`.

Haystack, MIT. Digital rf. `https://github.com/MITHaystack/digital_rf`.

Hilburn, Benjamin. Sigmf release v0.0.1. Jul 2018. doi: 10.5281/zenodo.1418396.

Hodges, Jeff. Jsonpp. `https://github.com/jmhodges/jsonpp`.

Mohan, Ankur, Pappu, Ravi, and Shadmand, Sean. D3 - a system for recording complex experiments with an extension of sigmf. *Proceedings of the GNU Radio Conference*, 3, 2018. URL `https://pubs.gnuradio.org`.

NTIA. Scos sigmf extension. `https://github.com/NTIA/sigmf-ns-scos`.

POSIX.1-2017. IEEE Std 1003.1-2017. Standard, The Open Group & IEEE, 2018. URL `http://pubs.opengroup.org/onlinepubs/9699919799/`.

Research, Rincon. Midas blue. `http://nextmidas.techma.com/nm/nxm/sys/docs/MidasBlueFileFormat.pdf`.

SemVer 2.0.0. Semantic versioning 2.0.0. Standard, 2013. URL `https://semver.org/`.

SkySafe. gr-sigmf. `https://github.com/skysafe/gr-sigmf`.

VITA. Vita-49. `https://www.vita.com/VITA-49`.