VTGS Satellite Metadata Format
Specification Document

Document ID:	SatMF Specification
Document No:	200-005
Version:	1.0.0-rc2
Date:	2019-06-08
Program:	VT Ground Station
Prepared By:	Zach Leffke
Approved:	Zach Leffke

Distribution:	FOR INTERNAL USE ONLY
Approved:	Zach Leffke
Approval Date:	2019-02-11

Contents

List of Tables

1 General Information

1.1 Purpose

The purpose of this document is to specify the Satellite Metadata Format for the purposes of data transport and aggregation for ground processing systems.

1.2 Scope

The scope of this document is limited to information related to specifying the JSON key/value pairs for the Satellite Metadata Format (SatMF). Additionally, recommended file naming conventions, descriptions of particular use cases, and the logic behind the particular fields are provided. This document does not provide the Application Programming Interface details for how to send, receive, or query for these packets over a network connection. Any system that makes use of the SatMF specification should detail the implementation specifics in a separate system design document or API specification document.

1.3 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.4 Project References

1.4.1 AMSAT STP Specification

The VTGS SatMF specification is not a direct implementation of the AMSAT Satellite Telemetry Protocol (STP) format, but is directly influenced by it. The majority of the "packets" field metadata is taken from the STP specification. SatMF is intended to be a lightweight update of or alternative to the AMSAT STP specification that uses JSON schemas.

For more details concerning STP, Please see:
https://svn.sarpeidon.net/suitsat2/repos/ground_station_software/ARISSatTLM/Documentation/telem.txt

1.4.2 Signal Metadata Format (SigMF)

The SatMF specification is nearly identical to the Signal Metadata Format project (aka 'SigMF'). This is intentional and by design, and the authors of this specification wish to acknowledge that the majority of the 'hard stuff' was figured out by the creators of SigMF. The details concerning the similarities and differences between SatMF and SigMF are discussed in detail in Section 2.3.

For more details concerning SigMF, please se:
https://github.com/gnuradio/SigMF

1.5 Copyright

1.6 Technical Points of Contact

Organization:	VTGS	VTGS
Name:	Zach Leffke	Seth Hitefield
Email:	zleffke@vt.edu	sdh11@vt.edu
Phone:	(540)808-6305	(423)217-9048

2 Introduction

It is acknowledged by the VTGS Team that this specification will evolve and change over time. This current version of the packet format is intended to serve as a starting point to begin a path towards implementation. As time evolves, this format will be updated with possibly significant changes.

2.1 Background

Multiple pieces of information are of interest when a packet of data is exchanged with a spacecraft, not just the content of the packet itself. This information about the packet is referred to as metadata. Metadata is data about data. It is important to retain this information and attach this information to the packet of interest in a defined way. The easiest example of this is to document the time at which a packet was received, referred to as a timestamp. In addition to the packet itself, the time at which the packet was received must be attached to that data in a pre-defined way.

2.2 SatMF Overview

The Virginia Tech Ground Station (VTGS) Satellite Metadata Format (SatMF) specifies a way to describe sets of satellite data with metadata written in JSON. SatMF can be used to describe general information about the satellite data, the characteristics of the system that generated the satellite data, the system communicating with the satellite, and features of the satellite data itself. This description is nearly identical to the SigMF description given in the SigMF specification abstract, and this is by design. It is important to recognize that SatMF only specifies the format for recording the data and metadata. The method of transmission, reception, etc. is left to the user to determine. The intent is to allow end users of this format to develop their own implementations that utilize this format.

2.3 SatMF vs SigMF

Any user of SatMF is encouraged to become familiar with SigMF as SatMF strives to be very similar. The best description of the SigMF specification can be taken directly from their abstract:

"The Signal Metadata Format (SigMF) specifies a way to describe sets of recorded digital signal samples with metadata written in 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."

Generally, SigMF is used to record raw sample streams and therefore an analogy to the Physical Layer (Layer 1) of the OSI Model can be made. The data contained in a SigMF recording does not necessarily contain demodulated information (unless a user elects to define a mechanism to do so via an extension). SatMF on the other hand seeks to encapsulate and record metadata about already demodulated data. Therefore, an analogy to Layer 2 of the OSI Model can be made for SatMF. Perhaps in simpler terms, SatMF is intended to be used with frames of data sent to, from, and/or between spacecraft and to specify a method for recording metadata about that information.

One may ask, why not simply define SatMF as a custom extension under SigMF? Good question, and the answer is not so clean. The SigMF specification is intended to be general, and isn't even necessarily specific to radio frequency (RF) systems. SatMF is not as general as SigMF. It is not intended to be a general data/metadata format for ALL Layer 2 data, and since this is specifically about *satellite* data and metadata (like AMSAT's STP protocol) it was decided to split from the SigMF specification, while maintaining a significant amount of its functionality and conventions. This allows systems to use SatMF that perhaps elect *not* to use SigMF. The overall goal is to provide a general enough, but simple, mechanism for retaining metadata associated with satellite data. Since SatMF can be seen as a hybrid between SigMF and AMSAT's STP, it was decided to create a new specification.

2.3.1 Similarities

SatMF, like SigMF, SHALL use Java Serialized Object Notation (JSON) format. Information concerning the ground station and satellite conducting the communications are stored in the `global` top level JSON object. The top level `packets` JSON object is a JSON Array (ordered list) and contains the per packet data and metadata as entries in the array. One of the major features of SigMF that makes it incredibly flexible is the ability for users to define custom extensions of the specification to meet their specific needs. A similar mechanism for defining custom SatMF extensions is also provided allowing a user to implement custom data and metadata formats that are not part of the canonical SatMF specification. This document will also advise (*not specify*) how file names should be generated when storing the data.

2.3.2 Differences

There are a number of major differences between SatMF and SigMF. First,SigMF keeps the data and metadata separate in different files with different filename extensions. The 'dataset' (.sigmf-data filename extension) file is a digital binary file of samples and the 'metadata' (.sigmf-meta) file contains the information that describes the dataset. SatMF completely breaks this convention and everything (data and metadata) is stored in a single JSON file. More specifically in the entries of the JSON Array defined by the "packets" top level JSON object, there is a "raw" key used to encapsulate the data itself.

The second major difference is in the top level JSON objects. SigMF uses three fields: "global," "captures," and "annotations." SatMF will only contain two top level fields: "global," and "packets." Like SigMF, in SatMF the "global" field is intended to contain information that applies to all of the dataset which are all the packets in the "packets" field. The "packets" field of a SatMF object is a bit of a hybrid of the "captures" and "annontations" fields of SigMF. Stated more simply, the SatMF "global" object SHALL contain static (non-changing) information that applies to all the packets in the "packets" object, while the "packets" object shall be an ordered JSON Array that contains time varying information that pertains to the individual packet (such as a datetime timestamp, the raw binary, etc.).

A third difference that is worth mentioning is that SatMF will contain multiple layers of nested data. While this is not explicitly prohibited in SigMF, the canonical representations only show two layers (with the exception of defined extensions); the top level objects (layer 1) and the fields

defined within them (layer 2). This is worth mentioning if for no other reason than it will make specification of each key/value pair a bit more difficult in SatMF in terms of a simple table format and document organization.

Finally, the concept of 'namespaces' and extensions should be discussed. SigMF makes use of namespaces to define the core fields defined in the canonical specification. When users define a custom extension, the extension namespace is used to differentiate from the core namespace in the top level JSON objects. For this version of SatMF, the namespace conventions are *not* being used in the interest of expediency. It is envisioned the near term future versions of the SatMF specification will make full use of the namespace conventions.

3 SatMF Conventions

3.1 SatMF Object Format Example

For the purposes of this document the term 'SatMF object' shall be used to refer to the pair of top level JSON objects in a SatMF File, the `global` and `packets` objects. SatMF objects SHALL be in JSON format. A fully populated example SatMF object is shown below. Not all fields depicted below are required fields.

```
{
  "global":{
    "version":"1.0.0",
    "ground_station":{
      "latitude":37.229980,
      "longitude":-80.439628,
      "altitude":610,
      "callsign":"WJ2XMS-2",
      "common_name": "VT Ground Station, VTGS",
      "description":"M2 400CP30x2, ARR P390-420VDG, Ettus N210 w/ UBX",
      "operator_id":"Zach Leffke, zleffke@vt.edu"
    },
    "spacecraft":{
      "norad_id":99999,
      "callsign":"WJ2XMS-1",
      "common_name":"VT-Ceres"
    },
    "extensions":""
  },
  "packets":[
  {
    "index":0,
    "datetime":"2019-02-13T05:43:02.595874164Z",
    "time_source":"uhd",
    "time_quality":"stratum_1",
    "decode_type":"live",
    "link_type":"downlink",
    "snr":25.1,
    "center_frequency":401120000.0,
    "frequency_offset":8726.0,
    "raw":"82a09a92606860969468a69ca860968868849ca2e6ae92888a64406303f03a4b4a34534e5420
           20203a554e49542e566f6c742c506b742c506b742c50636e742c506b742c4f6e2c4f6e2c4f6e
           2c4f6e2c48692c48692c48692c4869"
  },
  { "index":1, ...},
  { "index":2, ...}
  ]
}
```

3.2 Data Types

Data types within SatMF are defined the same as data types in SigMF:

Table 1: SatMF Data Types.

Type	Long Form Name	Description
int	integer	Signed 64-bit integer.
uint	unsigned long	Unsigned 64-bit integer.
double	double-precision floating-point number	A 64-bit float as defined by IEEE 754.
string	string	A string of characters, as defined by the JSON standard.
boolean	boolean	Either `true` or `false`, as defined by the JSON standard.
null	null	`null`, as defined by the JSON standard.
array	JSON Array	an `array` of other values, as defined by the JSON standard.
object	JSON Object	an `object` of other values, as defined by the JSON standard.

For key fields that are not required, the field itself may either be omitted entirely from the SatMF file or may contain a `null` entry for the value field. It is RECOMMENDED to omit any optional field rather than populating it with a `null` value. For required fields where the value is unknown, the key MUST be present in the object and the value MUST be set to `null`.

3.3 Time Standard

All time data SHALL use the **Universal Time Coordinated (UTC)** time standard.

3.4 SatMF Files

A SatMF file SHALL consist of a single SatMF object. Given this requirement, the SatMF Object example in Section 3.1 could also be called a SatMF File example. This convention may change in future versions of SatMF when considering more complex combinations of

It is anticipated that packets will be received at a ground station in batches associated with each pass of a spacecraft over the ground station. It is RECOMMENDED that the generated SatMF packets (in JSON format as described above) associated with the pass be stored in a single file, referred to as a 'pass file'. While this may be common, it is NOT a specific requirement. Those pass files can then be submitted to a SatMF processor (for example a data warehouse) via any standard file transfer mechanism as defined by the SatMF processor ICD/API. For example, a pass file may be submitted via FTP, ownCloud, File upload to a webserver, etc. The specific transfer mechanism(s) is left to the implementer of the SatMF processor to define.

3.4.1 SatMF File Naming Convention

The following SatMF File naming convention is RECOMMENDED. This format, while not required, will facilitate storage and organization of the files in the data warehouse. The convention described below will allow rapid identification of the ground station, spacecraft, date, and time (or orbit number) associated with the SatMF Pass File. If using this naming convention, the fields in the file name SHALL be underscore delimited. When storing a SatMF file, the filename extension `.satmf` SHALL be used.

Recommended SatMF File Naming Convention Format:

`<NORAD ID>_<GS ID>_<DATE>_<TIME>.satmf`
or
`<NORAD ID>_<GS ID>_<ORBIT_NUM>.satmf`

NORAD ID NORAD identification number of the spacecraft. SHOULD be 5 characters long.

GS ID Ground Station Identifier, RECOMMENDED to use the callsign and SSID of the ground station. A ground station common name is also acceptable.

DATE Date the pass file was generated indicated by year (YYYY), month (MM), and day (DD) in the following format: YYYYMMDD. The date field SHALL be 8 characters long, and months and days with less than two characters SHALL be zero padded. The date field timezone SHALL be Universal Time Coordinated (UTC/Zulu).

TIME Time the pass file was generated indicated by hours (HH), minutes (mm), and seconds (SS) in the following format: HHMMSS. The time field SHALL be 6 characters long, and hour, minute, and second fields with less than two characters SHALL be zero padded. The time field timezone SHALL be Universal Time Coordinated (UTC/Zulu).

EXAMPLE Spacecraft 99999 received by Ground Station WJ2XMS-2 (the VTGS) starting on Feb 13th, 2019 at 6 hours, 5 minutes, 20 seconds AM, UTC time:

`99999_WJ2XMS-2_20190213_060520.satmf`

Alternatively, instead of using the callsign of the ground station, an alternative ground station identifier MAY be used, such as a common name:

`99999_VTGS_20190213_060520.satmf`

Before implementation, it is RECOMMENDED that either the callsign or common name convention be selected and then adhered to for consistency. Mixing and matching callsign and common name is NOT RECOMMENDED.

ORBIT NO For a single ground station, for a given orbit number of the spacecraft, only a single pass will occur. Therefore, as a natural time delimiter that may be desired for organization of batches of data, the orbit number of the spacecraft MAY be used instead of the date/time. This implies that all `datetime` timestamps within the file for the listed packets occur after the start of an orbit and before the end of an orbit for the given spacecraft:

`99999_VTGS_1234.satmf`

3.4.2 Future SatMF File Conventions

The SatMF specification is very new and all use cases for file naming have not yet been considered. The above section is a rapid initial description of how data may be stored in a SatMF file, with a common use case identified (such as a satellite pass). The driving logic behind this use case **assumes a single ground station receiving or sending data from/to a single spacecraft** and then submitting that data to a data warehouse.

What if however, multiple ground stations receive the same pass data and someone wishes to retain all of that data in a single location, such as a SatMF file. This may be the case for someone querying data from a data warehouse and searching for ALL packets received from a given spacecraft, which may have come from multiple SatMF sources (ground stations). Unless significant modifications to this specification are made, delivering that data in a single SatMF file is not currently possible since the `global` field only allows for a single ground station and single spacecraft entry.

As a potential solution, the data from the multiple ground stations could be concatenated in the SatMF file (more than one SatMF object per file) or the packets themselves could be interleaved based on time stamp, with multiple ground stations listed in the `ground_station` field (perhaps as a nested JSON Array) and then each packet could be tagged with a reference to the receiving ground station in the list of ground stations in the global object (perhaps referenced by index to the JSON Array).

Other combinations of spacecraft and ground stations may be desirable as well (such as a single ground station receiving multiple data streams from a cluster of spacecraft). In any situation involving more than a single ground station and a single spacecraft, the above file naming conventions no longer make sense.

Future versions of this specification will expand on this topic after more careful consideration.

4 Top Level SatMF Object Definitions

There SHALL be two top-level fields to comprise a SatMF Object. These fields are `global` and `packets`. Both of these fields are REQUIRED to comprise a SatMF object.

4.1 global Objects

The `global` object contains metadata associated with all entries in the `packets` JSON Array. Table 2 below shows the child keys assigned to the `global` object.

Table 2: SatMF `global` Child Key Definitions.

JSON Key	**Required**	**Description**
version	true	SatMF version number
ground_station	true	Contains ground station metadata
spacecraft	true	Contains spacecraft metadata
extensions	false	Contains extensions references

See Section 5 for a detailed breakdown of the `global` field definitions.

4.2 packets Array

The `packets` object is an ordered JSON Array. The `packets` object SHALL contain the raw packet data as well as metadata associated with individual packet entries. A minimum of one packet entry in the `packets` array is REQUIRED. There is no limit to the number of entries in the `packets` array.

See Section 6 for a detailed breakdown of the `packets` field definitions.

5 SatMF `global` Field Definitions

5.1 `version`

Parent key: global
Required key: true
Data type: string

The SatMF version specifies the version of the SatMF implemented by each packet. The version of this document SHALL be consistent with the SatMF version number (See note below on release candidate documents). The version number field is a REQUIRED field of the SatMF packet. SatMF versioning SHALL follow Semantic Versioning 2.0.0 format. This format follows the general rule of MAJOR.MINOR.PATCH format. Per the above reference for Semantic Versioning, given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

Example: `"version":"1.0.0"`

Note On Documentation Release Candidates
Release Candidate nomenclature will be used exclusively in documentation only and not in the actual version field of SatMF objects. At no time should a SatMF object contain "-rc1" in the version value field. In this way versions of documents can be incremented and shared for the same SatMF version being defined.

5.2 `ground_station`

Parent key: global
Required key: true
Data type: JSON object

The `ground_station` field describes parameters of the ground station.

Table 3: SatMF `ground_station` Child Key Definitions.

JSON Key	Units	Data Type	Required	Description
ground_station	N/A	JSON Object	true	Top level key for ground station fields.
latitude	degrees [deg]	double	true	Decimal Degree format. WGS-84 (GPS) Reference coordinate system.
longitude	degrees [deg]	double	true	Decimal Degree format. WGS-84 (GPS) Reference coordinate system.
altitude	meters [m]	double	true	Above Mean Sea Level. WGS-84 (GPS) Reference coordinate system.
callsign	N/A	string	false	Callsign associated with FCC license of ground station.
common_name	N/A	string	false	Common use name of a ground station, i.e. "VTGS"
description	N/A	string	false	Shorthand description of GS hardware
operator_id	N/A	string	false	Ientification information of the human operator using the ground station.

5.2.1 `latitude, longitude, altitude`

Parent key: ground_station
Required key: true
Data type: double

The `latitude`, `longitude`, and `altitude` fields are required fields in order to enable more detailed analysis using metadata in the future. For example, in order to perform any kind of Doppler shift analysis, the location of the receiving system must be known. For `latitude` & `longitude`, Decimal Degree format is REQUIRED. Northern Hemisphere is positive, Southern Hemisphere is negative. WGS-84 (GPS) Reference coordinate system. For `altitude` the units shall be in Meters [m].
Examples: `"latitude":37.229980`, `"longitude":-80.439628`, `"altitude":610`

5.2.2 `callsign & common_name`

Parent key: ground_station
Required key: false
Data type: string

The `callsign` and `common_name` fields are not required fields with one exception (see note below). However, it is highly RECOMMENDED that at least one of the fields be used to provide easy identification of the ground station. If an FCC issued callsign is available for the ground station, this should take precedence and be used over a common name. SSIDs may be used with the callsign, i.e. "WJ2XMS-2". If however, no FCC callsign is issued to the ground station, as is common for receive only monitoring stations that wish to submit data, the common name associated with the ground station should be used.

Callsign Example: `"callsign":"WJ2XMS-2"`
Common Name Example: `"common_name":"VT Ground Station, VTGS"`

Note: For uplink packets, it is implied that an FCC Callsign has been issued for the ground station. Otherwise, the station would be illegally transmitting. Therefore, if at least one `link_type` field in the `packets` array is set to `uplink`, then the `callsign` field MUST be included.

5.2.3 `description & operator_id`

Parent key: ground_station
Required key: false
Data type: string

The `description` field should be used to give a brief description of the ground station hardware configuration. If a callsign is specified, this effectively identifies the operator, unless it is a shared callsign (such as an Experimental, Part 5 callsign). Users may elect to identify the operator by name, email address, etc. regardless of the use of the callsign field, using the `operator_id` field.

Description Example: `"description":"M2 400CP30x2, ARR P390-420VDG, Ettus N210 w/ UBX"`
Operator ID Example: `"operator_id":"Zach Leffke, zleffke@vt.edu"`

5.3 `spacecraft`

Parent key: global
Required key: true
Data type: JSON object

The `spacecraft` field describes parameters of the spacecraft.

Table 4: SatMF `spacecraft` Child Key Definitions.

JSON Key	Units	Data Type	Required	Description
spacecraft	N/A	JSON Object	true	Top level key for spacecraft fields.
norad_id	N/A	uint	true	NORAD identification number. Same number used in Two Line Element formats. Assigned by JSPoC once satellite is delivered to orbit.
callsign	N/A	string	false	Callsign associated with FCC license of spacecraft, may include SSID; i.e. “WJ2XMS-1“ (receive only systems, without licenses, may inject data, therefore field is optional).
common_name	N/A	string	false	Common use name of a ground station, i.e. "VTGS"

5.3.1 `norad_id`

Parent key: spacecraft
Required key: true
Data type: uint

North American Aerospace Defense Command Satellite Identification Number. Also known as Satellite Catalog Number, NORAD Catalog Number, NORAD ID, NASA Catalog Number, or USSPACECOM object number. This is a 5 digit identification number assigned by NORAD and found in the Two Line Element Set (TLE). May use unassigned numbers for testing (i.e 99999 for VT, 99998 for UVA, 99997 for ODU). As this is a required field, if it is unknown use `null` (rx only gs that may not know NORAD ID of what they are tracking).
Example: `"norad_id":99999`

5.3.2 `callsign` & `common_name`

Parent key: spacecraft
Required key: false
Data type: string

The `callsign` and `common_name` fields are not required fields. However, it is highly RECOMMENDED that at least one of the fields be used to provide easy identification of the spacecraft. If an FCC issued callsign is available for the spacecraft, this should take precedence and be used over a common name. SSIDs may be used with the callsign, i.e. "WJ2XMS-1". Even though an FCC callsign is issued to the spacecraft as a requirement for launch integration (at least in the US), it may not be known by the receiving ground station and is therefore an optional field.

Callsign Example: `"callsign":"WJ2XMS-1"`
Common Name Example: `"common_name":"VT-Ceres"`

5.4 `extensions`

Parent key: global
Required key: false
Data type: multiple possibilities

The `extensions` field is intended to be used for custom metadata that is not currently covered by the implemented version of the SatMF Specification. The intent of this field is *not* to simply fix issues with SatMF that may require updating and version changes. Instead this field is intended to allow users to implement completely custom SatMF JSON schema extensions that may not be part of the core SatMF specification. One envisioned use case is for higher layer parsing of packet data. For example, a user may elect to decode the `raw` field of the entries in the `packets` top level JSON array and place that information into a custom defined `decoded` field within each packet entry.

Note on Extensions, Namespaces, Nested JSON Fields

The concept of 'namespaces' in SigMF have *not* been included in this version of SatMF but are expected to be included in future versions. SigMF namespaces are intricately linked to definition of extensions. When using namespaces, custom extensions are explict in the JSON. Without the use of namespaces, custom extensions are not explicitly clear to someone looking at a SatMF Object.

In the interest of avoiding confusion, this topic will not be discussed further as it would require a rather lengthy discussion. SatMF users should be aware that significant changes to the JSON schema definition if/when incorporating namespaces will likely be needed for handling SatMF JSON objects that include namespaces and extensions. This topic will undergo significant consideration and changes will be detailed in future versions. In the mean time, readers are encouraged to read the SigMF documentation concerning extension definitions and use as well as the use of namespaces.

6 SatMF `packets` Field Definitions

6.1 `packets` JSON Array

Parent key: None, this is a Top-Level SatMF key.
Required key: true
Data Type: JSON array

The `packets` value is an ordered JSON Array. Entries in the array SHALL be of JSON Object type. This is depicted below.

```
"packets":[<JSON Obj>, <JSON Obj>, <JSON Obj>, ..., <JSON Obj>]
```

Individual JSON Object entries in the `packets` array may be thought of as a 'SatMF' packet and SHALL contain the raw packet data as well as metadata associated with individual packet entries. The order of entries SHALL be controlled in ascending time order according to the `datetime` field contained in each object. A minimum of one packet entry in the `packets` array is REQUIRED. There is no limit to the number of entries in the `packets` array.

6.2 `packets` JSON Object Detailed Description

Parent key: None, entry in an ordered JSON array object.
Required entry: A minimum of 1 JSON Object as defined in this Section.
Data Type: JSON object

Table 5: SatMF `packets` JSON Object Child Key Definitions.

JSON Key	Units	Data Type	Required	Description	ref
index	N/A	uint	false	index number of the packet.	6.2.1
datetime	N/A	string	true	CRITICAL FIELD that must be kept accurate. The time that the packet was received (if `link_type` == `downlink`) or transmitted (if `link_type` == `uplink`).	6.2.2.
time_source	N/A	string	true	Indication of the device that is actually producing the datetimes.	6.2.3
time_quality	N/A	string	true	Indication of the quality of the time source.	6.2.4
decode_type	N/A	string	true	`live`: packet was decoded over the air live `post`: packet was decoded in post processing	6.2.5
link_type	N/A	string	true	`uplink`: packet was sent from GS to SC. `downlink`: packet sent from SC to GS `crosslink`: packet sent from SC to SC (Future Use)	6.2.6
snr	decibels [dB]	double	false	*Measured* Signal-To-Noise Ratio of received packet	6.2.7
center_frequency	hertz [Hz]	double	false	Tuned Center Frequency of Transceiver. May or may not be the center frequency of the spacecraft (if receiving an entire band for example).	6.2.8
frequency_offset	hertz [Hz]	double	false	*Measured* frequency offset of the signal relative to the `center_frequency` field of packet.	6.2.8
raw	N/A	string	true	Hexstring containing the complete packet, including the header. Raw hextstring, i.e. no '0x' preceding hex values.	6.2.9

6.2.1 `index`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: false
Data Type: uint

The `index` field is used to indicate the packet count index number. It is NOT REQUIRED that this index value match the `packets` ordered JSON array index number. The minimum acceptable value is 0 (zero indexed), though this is not a strict requirement. For each subsequent decoded packet, this field shall be incremented by 1. The index field shall ascend in order according to time. Thus earlier packets according to the `datetime` field should have lower indices relative to packets with later datetimes. This is an optional field as each packet in the JSON Array should be ordered according to time of reception or transmission.

This field is included as an option for users who wish to be explicit about the index location. For example, it is anticipated that multiple copies of the same passes's worth of data may be submitted to a SatMF processor. Setting this field may be useful as part of the deconfliction of duplicated entries as part of the algorithm that combines multiple JSON Arrays into a single deconflicted Array as part of the per orbit normalization process. Alternatively, it is also possible that only partial submissions for a single pass may be submitted to a SatMF processor. In this case, the `index` field may be used to identify the overall index of the packet for the pass, not just the index in the `packets` ordered JSON array for the submitted SatMF file.

Example: `"index":42`

6.2.2 `datetime`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

The `datetime` field is CRITICAL as it directly influences data organization and storage of received data in SatMF processors (Such as a logging utility or a more complex data warehouse). Every effort should be taken to ensure that this is an accurate timestamp and follows the conventions of this packet specification as dictated in this section.

All datetimes SHALL be in **UNIVERSAL TIME COORDINATED (UTC)** format.

The `datetime` field SHALL be a JSON string data type as JSON does not directly support datetime data types or objects.

The `datetime` timestamp SHOULD be associated as closely as possible to the first indication event or 'leading-edge' of a received packet. For example, if using an SDR application and time-stamping according to sample offsets within a sample stream, the sample that first produces a burst detection event and its associated sample index (offset) within the stream mapped to a UTC time should be used for the datetime. This may not be possible for all types of receivers, such as analog (hardware) radios and TNC decoders for example, but every effort should be made to indicate when the packet *first* arrived (or departed if transmitting) at the station.

The `datetime` field SHALL be in accordance with ISO-8601, as defined by RFC 3339, where the only allowed time-offset for the SatMF implementation is Z, indicating the UTC/Zulu timezone. According to this standard, the datetime SHALL follow this format:

`YYYY-MM-DDThh:mm:ss.fffZ`

Where:

Field	Description	Example
YYYY	Year with century as decimal number	2019, 2020, 2021, etc.
MM	Month as zero-padded decimal number	01, 02, ..., 11, 12
DD	Day as zero-padded decimal number	01, 02, ..., 29, 30, 31
T	Date / Time concatenation delimiter	T
hh	Hour as zero-padded decimal number according to 24 hour clock system	00, 01, ..., 23
mm	Minute as zero-padded decimal number	01, 02, ..., 58, 59
ss	Second, integer as zero-padded decimal number	01, 02, ..., 58, 59
fff	Second, fractional with no limit to the number of decimal places as dictated by the accuracy of the timestamping system.	See below
Z	Time zone indicator, Zulu Time (UTC)	Z

Acceptable `datetime` Examples:
2019-02-13T05:43:02.595Z (millisecond accuracy implied)
2019-02-13T05:43:02.595874Z (microsecond accuracy implied)
2019-02-13T05:43:02.595874164Z (nanosecond accuracy implied)
Note the number of fractional decimal places and the implied accuracy of the timestamping system.

Unacceptable `datetime` Example:
2013-02-01T12:52:34.00-05:00 <- this format is NOT acceptable and will be silently rejected by SatMF processors as it is indicating a timezone of 5 hours west (EST) of the UTC zero meridian. The Equivalent UTC/Zulu datetime that would be accepted is: 2013-02-01T17:52:34.00Z.

6.2.3 `time_source`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

The `time_source` field is used to indicate what is producing the actual timestamps in the `datetime` field. Generally, the closer to the antenna that the timestamp is produced the better. The RECOMMENDED values for this field are `uhd`, `host`, `other`.

`uhd` should be used if using Ettus Research USRPs and the UHD driver is configured to emit timestamp stream tags. This is considered one of the best timestamping approaches as it is as close to the antenna as possible.

`host` should be used if the timestamps are produced by the host operating system that is running the software radio application or decoding software. If timestamp tags are produced within an SDR application, but not directly by the driver that is running the ADC clock (i.e UHD for USRPs), then `host` should be used.

Example: `"time_source":"uhd"`

6.2.4 `time_quality`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

This field is used in conjunction with the `time_source` field to indicate the quality of the time source used to produce the timestamps. Generally, this field should follow Network Time Protocol (NTP) conventions if the `time_source` field is set to either `uhd` or `host`. Under this convention each layer of the hierarchy is termed a 'stratum.' The top of the hierarchy is a precision time keeping device and as one moves further 'away' from the precision time keeping device, the stratum number is incremented.

Stratum 0 devices are generally high precision devices such as atomic clocks, GPS receivers, or other timekeeping radio receivers (such as WWV).

Stratum 1 devices are computer systems whose system time is synchronized to within a few microseconds of the attached Stratum 0 device. To be termed a Stratum 1 device the device must be directly connected to the timekeeping device, usually via a dedicated connection such as IRIG-B, Pulse-Per-Second, Serial (often in conjunction with PPS), or via some other dedicated connection. The most common implementation of Stratum 1 devices are as 'Primary Time Servers' that are used to provide NTP services to systems via a network connection such as the Internet or via a LAN connection.

Stratum 2 devices are connected via a network connection to one or more Stratum 1 devices. This is generally the situation for computer systems connected to the Internet and using an NTP daemon of some sort. This is often done automatically when 'getting the time from the internet' in Windows/MAC systems. On most Linux Debian/Ubuntu systems NTP daemons and monitoring tools (to determine the current NTP stratum) can be installed via aptitude (or similar package management tools). Depending on the Linux distribution and version, this may or may not be automatically done and all SatMF processors are encouraged to verify that NTP is installed on their system.

Stratum 3 devices are synchronized to Stratum 2 devices. It is possible that systems using NTP daemons may not always be Stratum 2 and may be Stratum 3 or worse, especially during initial startup of NTP the daemon.

When using stratum indicators for time quality, the values should be all lower case with the word 'stratum' delimited from the stratum value by an underscore.

Examples: `stratum_1`, `stratum_2`, `stratum_3`

It is possible that an operator may elect to use a host system that is NOT running an NTP daemon (i.e. should firewall rules for some reason block access to external NTP servers). In this case the system time may be set and updated manually and may drift away from the true time between manual corrections. Therefore, to handle this case, the other acceptable value for the `time_quality` field when using `host` for the `time_source` field but not using an NTP daemon to correct the host machine's system time (freely drifting clock) is `unlocked`.

If the `time_source` field is set to `other`, then the `time_quality` field may be used to describe the time quality and source using something other than the stratum indicators if desired. Alternatively, the stratum convention may still be used if desired even if the `time_source` field is set to `other`. Generally, the use of `other` for the `time_source` field and the use of something other than the stratum indication method for the `time_quality` field is NOT RECOMMENDED, however is provided for cases not covered by the recommended conventions and which may be included in future versions/updates of this specification.
For more details on NTP see: https://en.wikipedia.org/wiki/Network_Time_Protocol#Clock_strata. For common example implementations of `time_source` and `time_quality` combinations please see Section 7.2.

Example: `"time_quality":"stratum_1"`

6.2.5 `decode_type`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

It is anticipated that not all packets will be decoded in real time. Therefore, the `decode_type` field is used to indicate what type of decoder was running. For example, it is expected during the Launch and Early Operations Phase (LEOPs) of a mission that operating parameters of the radios may need to be fine tuned. Therefore, it is possible that initial passes will be recorded as raw IQ data. This data may be replayed multiple times through the decoders (i.e. GNU Radio flowgraphs) to maximize the number of packets collected. This field is used to indicate whether the packets were decoded live or via file based playback in 'post-processing.'

The overall spirit of this field is intended to provide a secondary check on the datetimes and time sources involved in the time-stamping process. The `datetime` field is critical to keeping packets organized within the data warehouse. Therefore if packets are decoded in post-processing long after the pass has completed, this field should be set to `post`. If packets are decoded live, this field should be set to `live`.

Example: `"decode_type":"live"`

6.2.6 `link_type`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

The `link_type` field is used to indicate the type of link this packet was sent over. As shown in Table 5 the acceptable values are: `uplink`, `downlink`, and `crosslink` (future use). While it is expected that the majority of packets sent will be of type `downlink`, the ability for the SatMF packet to handle all types of links is important. For example, recording uplinked packets as well

as downlinked packets in a single SatMF pass file might be useful for building 'ping diagrams' and troubleshooting system performance.

Example: `"link_type":"downlink"`

6.2.7 `snr`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: false
Data Type: double

A *measurement* of the Signal To Noise Ratio of the packet. Units are Decibels [dB].

Tracking this field will be useful for a number of applications. Initially, it is expected that determining the ground stations *actual* local elevation mask is one such use (instead of a blanket 10 or 20 degree assumption). It is also envisioned that this field will be useful during LEOPs in confirming the proper deployment of the antenna systems as noticeable increase in SNR is expected should the antennas properly deploy.

More advanced concepts can be explored in the future such as attempting to confirm/match attitude changes and antenna orientation/pointing of the spacecraft as it tumbles to SNR values. In this way phenomenon such as 'spin fades' can be measured and understood.

Even more advanced future concepts include using SNR information in combination with the `timestamp`, `frequency_offset`, and `center_frequency` fields to apply advanced Time Difference of Arrival (TDOA) and Frequency Difference of Arrival (FDOA) geolocation techniques. In this use case, SNR can be used as part of the weighting functions involved in the algorithms in efforts to determine the spacecraft position over time and which then enables the performance of initial orbit determination (IOD) and orbit estimation techniques.

Example: `"snr":25.1`

6.2.8 `center_frequency` & `frequency_offset`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: false
Data Type: double

The units for both `center_frequency` and `frequency_offset` SHALL BE Hertz [Hz]. The `center_frequency` field should be filled with the actual tuned center frequency of the receiving or transmitting system both for link_types of `uplink` and `downlink`. Ideally, this would align with and be identical to the center frequency of the associated spacecraft radio. However, frequencies can drift, most notably as a function of temperature and oscillator dependency on temperature. Therefore, this field should be filled with the ground station radio's *actual* or *measured* center frequency.

Similarly, the `frequency_offset` field should be filled with the *measured* frequency offset of the signal (packet) relative to the `center_frequency` field. It is envisioned that for most use cases this

field can be thought of as the so called 'Doppler Offset' of the signal.

It is important that both of these fields be populated with real data and not simply the expected values (such as those computed by TLE orbital propagators). The reason for this is that advanced applications using these fields include orbit determination and estimation techniques which can yield results *better* than TLE accuracy. However, that is only true if the receiving / transmitting system on the ground uses a high quality frequency reference oscillator (such as is produced by GPS Disciplined Oscillators). In the absence of a stable GPSDO or other reference oscillator, accurate measurement of true center frequencies by calibrated laboratory equipment is acceptable.

Another important use case for this field concerns initial deployment and the LEOPs phase of the mission. When cubesats are deployed, they are spatially very closely spaced to each other and therefore difficult to resolve from each other. NORAD may be able to resolve each individual spacecraft and generate a TLE for each, however it will not be immediately known to the spacecraft operator or NORAD as to which TLE belongs to which spacecraft. Knowing the center frequencies of the spacecraft, and being able to take highly accurate doppler shift measurements, will allow ground operators to accurately match the NORAD generated TLE to the appropriate spacecraft. Again, this is only feasible if the frequency information contained in these fields are *measured* or *real* values, not simply the TLE based estimate of the offset value.

Finally, it should be noted that for Software Radio based systems, operating on streams or chunks of IQ samples, it is expected that the ground station's center frequency will be tuned to the center frequency of the intended spacecraft and *will not* be retuned during a pass (which will result in instantaneous phase discontinuities that will produce 'pops' in the spectrum). This is true for both the uplink and downlink systems. Therefore, every entry in the `center_frequency` field is expected to be identical. The `frequency_offset` field is then used to indicate the tuning offset applied (uplink) or measured (downlink) *in software* to the sample stream (using a complex multiply operation when actual applying the software tuning).

Center Frequency Example: `"center_frequency":401112000.0`
Frequency Offset Example: `"frequency_offset":8726.0`

6.2.9 `raw`

Parent key: Entry in JSON Array `packets` child JSON object.
Required key: true
Data Type: string

FORMAT
The `raw` field of a SatMF packet SHALL be a hexadecimal string dump of raw binary data.
The `raw` field of a SatMF packet SHALL be byte or octet (8 bit) aligned. If non byte aligned data is encapsulated, the last byte encoded should be zero padded.
The `raw` field of a SatMF packet SHALL NOT contain a leading '0x' as is sometimes common with hexadecimal representations.
The `raw` field of a SatMF packet SHALL NOT contain whitespace or any other characters between the hexadecimal entries.

Note that the `raw` field is not strictly enforced in terms of content, only in terms of format. This is intentional to allow users to use this field however they wish.
See the `raw` entry in Section 3.1 for an example of a properly formatted `raw` field.

CONTENT
SatMF encoders SHOULD only encode *the information exchanged* between the ground station and the spacecraft. Higher layer frames and packets are envisioned to be the content of the `raw` field. Stated differently, this is the actual information transmitted over the air. However, given multiple layers of the OSI model and the manipulation of data at those layers, it is important to clarify. Low level data link and physical layer manipulation of the data is not the envisioned content of the SatMF packet. Therefore data manipulations such as line encoding, forward error correction, modulation of bits into symbols, etc. is not expected to be in the `raw` field. The content of the `raw` field SHOULD be specified in an ICD or API as part of a SatMF processor design documentation, but is not strictly enforced by the SatMF specification itself.

Example
For clarity by example, consider the reception of an AX.25 UI frame by a ground station from a spacecraft. AX.25 frames are often transported out of the modem using the KISS protocol. KISS manipulates the fields of the AX.25 frame and adds additional 'special character' bytes on the ground for frame delimiting and uses 'escaped' byte sequences. See https://en.wikipedia.org/wiki/KISS_(TNC) for more details on KISS. A SatMF encoder SHOULD only encode the AX.25 frame, *not* the KISS frame in the `raw` field of the SatMF packet as the AX.25 frame is what actually contains information intended for the recipient and the KISS bytes are removed before over the air transmission. KISS might be used to transport an AX.25 frame to a SatMF processor, which then only encodes the AX.25 data after the KISS operations are removed. The HDLC protocol may be considered to be one layer or sub-layer 'lower' than the AX.25 frame (still technically part of the Link Layer or Layer 2). Even though the HDLC Frame Check Sequence (FCS), HDLC flags, and bit stuffing process does add data that is sent over the air to the ground station from the spacecraft this should also *not* be included in the `raw` field. This is because SatMF is intended to be a higher layer specification. Since AX.25 frames are defined at the upper portion of the Link Layer, the operations below this sub-layer (HDLC Flags, FCS, bit stuffing) should not be included in the `raw` field. (Note: We recognize that technically HDLC operations are the lower sublayer of the link layer, but we had to draw a line somewhere for this example. Additionally, these features are typically used for synchronization, frame detection, and frame validation and do not directly carry information intended for the recipient and are thus usually removed from the received packet information.) Again, as the `raw` field content is not strictly enforced by the SatMF specification itself, it is left to the user to decide what exactly they wish to capture in this field and that this should be clearly defined in an ICD/API for the designed SatMF processor (i.e. nothing in SatMF specifically denies encapsulation of KISS framing or HDLC framing operations if that is the intended use of the implementer).

7 The Importance of Accurate Timekeeping

7.1 Post Processed Data & `datetime` Field

The `datetime` field, along with the `decode_type`, and `index` fields will be CRITICAL to maintaining data warehouse organization and data reduction processes. It is anticipated that multiple submissions of a single pass's worth of data will occur. Again the idea is that one submission may be live data, with unprocessed, raw IQ recorded at the same time. In post-processing, the recorded IQ can be played back and potentially more data extracted from the same pass. If additional packets are collected a number of important events MUST occur. First, the `decode_type` field must be set to `post`. Second, the index number of the packets must be updated according to the newly obtained data. Third, the `datetime` fields MUST maintain accuracy according to the originally recorded timestamps. Timestamps produced from post-processed data that are hours/days/weeks after the actual recording time are NOT directly detectable by the `datetime` field.

Finally, as a use case example, consider a student examining metadata from two years after the data was recorded. Assume this student has the historical TLEs from the time that the data was recorded. Let's say they are attempting to plot the measured downlink Doppler frequency offset on the same plot as the TLE based expected downlink Doppler offset. If the `datetime` field is corrupted as part of the post-processing of raw IQ, and does not accurately reflect the true timestamp, then it may show a doppler offset for times when the satellite was not even overhead. The student would have TLE based 'pass windows' for which the AOS and LOS times would not match the timestamps in the SatMF file, leading to significant confusion.

7.2 Exemplar `time_source` & `time_quality` Combinations

7.2.1 UHD, USRPs, & GPSDOs

If using a GPS Disciplined Oscillator (GPSDO) that is internally connected to an Ettus Research USRP, and the UHD driver for the USRP is properly configured to make use of the GPS time, then the `time_source` field should be set to `uhd` and the `time_quality` field should be set to `stratum_1`. The GPS receiver itself is a stratum 0 device, but the system it is connected to (the USRP) can be no better than stratum 1 since it is one step removed from the GPS. This is currently envisioned as the best possible combination in terms of time keeping.

If however, a GPSDO (or other reference oscillator such as a standalone OCXO or rubidium time source) is only used as a *frequency* reference and the UHD driver is configured to synchronize to the PC Clock, then `time_source` field should be set to `uhd` and the `time_quality` field should be set to `stratum_2` or `stratum_3` depending upon the host computers NTP stratum status. In this scenario, the UHD driver itself is still producing the timestamps (closest possible location to the antenna), however it is not synchronized to the Stratum 0 GPSDO, it is the host operating system's clock which is controlled by the NTP daemon and can therefore be no better than Stratum 2. An example of this scenario is when connecting a USRP to an external GPSDO such as the Ettus Research Octoclock *without* connecting the Pulse-Per-Second signal and configuring the UHD driver correctly to use the PPS signal.

7.2.2 GNU Radio Time Stamps *without* UHD

GNU Radio itself does not require the use of Ettus Research USRPs and the UHD driver for them. Alternative SDR hardware exist and can be used with GNU Radio such as RTL-SDRs, HackRF, LimeSDR, Red Pitaya, etc. (to name only a few). Therefore timestamps can be produced inside GNU Radio flowgraphs that do not necessarily originate from UHD drivers or the SDR hardware itself. For example, the burst detection stage of a receive only flowgraph may produce timestamps before decoding is accomplished but after digitization of the signal and pre-processing. In this scenario, the `time_source` field will likely be the Host operating system and therefore should be set to `host` with the `time_quality` field should be set to `stratum_2` or `stratum_3` depending upon the host computers NTP stratum status.

APPENDIX - Revision History

Table 6: Revision History

Version	Date	Author	Description
1.0.0-rc2	2019-06-24	ZJL	Release candidate 2 document version This version posted to Github.
1.0.0-rc1	2019-06-10	ZJL	Release candidate 1 document version
1.0.0	2019-05-08	ZJL	Setting up new version
0.0.1	2019-02-11	ZJL	Initial Document Generation