FDSN miniSEED 3

Release 2023-01-18

FDSN

Jul 16, 2023

CONTENTS

The International Federation of Digital Seismograph Networks (FDSN) defines **miniSEED** as a format for digital data and related information. The primary intended uses are data collection, archiving and exchange of seismological data. The format is also appropriate for time series data from other geophysical measurements such as pressure, temperature, tilt, etc. In addition to the time series, storage of related state-of-health and parameters documenting the state of the recording system are supported. The FDSN metadata counterpart of miniSEED is StationXML which is used to describe characteristics needed to interpret the data such as location, instrument response, etc.

CHAPTER

ONE

VERSIONING

Starting with version 3 of **miniSEED**, the specification version is a single integer. Any non-backward compatible change to the structure of the header or record results in an increment to this number. The specification version corresponds to *field 2, Format version*, in the header of records written to conform with this version of the specification. There are no minor revisions. However, the addition of new data encodings or new reserved headers is considered backwards compatible. The FDSN may add encodings and extra headers within the FDSN key or make editorial changes to the specification without change to the major version, resulting only in a new revision of the specification document. Note that all parsing software MUST allow for the potential of unknown, new data encodings and new extra headers as these are subject to extension. This requirement allows older software to parse future records successfully within the same specification major version, even though they may be unable to decompress the data payload for new encodings.

This document is miniSEED v3 rev. 2023-01-18.

Note: This specification defines miniSEED 3. See *Background* for information on earlier versions.

1.1 Record definition

The fundamental unit of the format is a data record. A time series is commonly stored and exchanged as a sequence of these records. There is no interdependence of records, each is independent. There are data encodings for integers, floats, text or compressed data samples. To limit problems with timing system drift and resolution in addition to practical issues of subsetting and resource limitation for readers of the data, typical record lengths for raw data generation and archiving are recommended to be in the range of 256 and 4096 bytes.

1.1.1 Record layout and fields

A record is composed of a header followed by a data payload. The byte order of binary fields in the header must be least significant byte first (little endian).

The total length of a record is variable and is the sum of 40 (length of fixed section of header), field 10 (length of identifier), field 11 (length of extra headers), field 12 (length of payload).

Field	Description	Type	Length	Offset	Content
1	Record header indicator	CHAR	2	0	ASCII ‘MS’
2	Format version	UINT8	1	2	Value of 3
3	Flags	UINT8	1	3	
	Record start time				
4a	Nanosecond (0 - 999999999)	UINT32	4	4	
4b	Year (0-65535)	UINT16	2	8	
4c	Day-of-year (1 - 366)	UINT16	2	10	
4d	Hour (0 - 23)	UINT8	1	12	
4e	Minute (0 - 59)	UINT8	1	13	
4f	Second (0 - 60)	UINT8	1	14	
5	Data payload encoding	UINT8	1	15	*Data Encodings*
6	Sample rate/period	FLOAT64	8	16	
7	Number of samples	UINT32	4	24	
8	CRC of the record	UINT32	4	28	
9	Data publication version	UINT8	1	32	
10	Length of identifier	UINT8	1	33	
11	Length of extra headers	UINT16	2	34	
12	Length of data payload	UINT32	4	36	
13	Source identifier	CHAR	V	40	URI identifier
14	Extra header fields	JSON	V	40 + field 10	
15	Data payload	encoded	V	40 + field 10 + field 11	

All length values are specified in bytes, which are assumed to be 8-bits in length. Data types for each field are defined as follows:

CHAR ASCII encoded character data.

UINT8 Unsigned 8-bit integer.

UINT16 Unsigned 16-bit integer (little endian byte order).

UINT32 Unsigned 32-bit integer (little endian byte order).

FLOAT64 IEEE-754 64-bit floating point number (little endian byte order).

JSON JSON Data Interchange Standard.

1.1.2 Description of record fields

1 CHAR: **Record header indicator**. Literal, 2-character sequence “MS”, ASCII 77 and 83, designating the start of a record.

2 UINT8: **Format version**. Set to 3 for this version. When a non-backwards compatible change is introduced the version will be incremented.

3 UINT8: **Flags**. Bit field flags, with bits 0-7 defined as:

0. Calibration signals present. [same as SEED 2.4 FSDH, field 12, bit 0]
1. Time tag is questionable. [same as SEED 2.4 FSDH, field 14, bit 7]
2. Clock locked. [same as SEED 2.4 FSDH, field 13, bit 5]
3. Reserved for future use.
4. Reserved for future use.

5. Reserved for future use.
6. Reserved for future use.
7. Reserved for future use.

4 Record start time, time of the first data sample. A representation of UTC using individual fields for:

a. nanosecond
b. year
c. day-of-year
d. hour
e. minute
f. second

A 60 second value is used to represent a time value during a positive leap second. If no time series data are included in this record, the time should be relevant for whatever headers or flags are included.

5 UINT8: **Data payload encoding**. A code indicating the encoding format, see Section 4: Data encoding codes for a list of valid codes. If no data payload is included set this value to 0.

6 FLOAT64: **Sample rate/period**. Sample rate encoded in 64-bit IEEE-754 floating point format. When the value is positive it represents the rate in samples per second, when it is negative it represents the sample period in seconds. Creators should use the negative value sample period notation for rates less than 1 samples per second to retain resolution. Set to 0.0 if no time series data are included in the record.

7 UINT32: **Number of samples**. Total number of data samples in the data payload. Set to 0 if no samples (header-only records) or unknown number of samples (e.g. for opaque payload encoding).

8 UINT32: **CRC of the record**. CRC-32C (Castagnoli) value of the complete record with the 4-byte CRC field set to zeros. The CRC-32C (Castagnoli) algorithm with polynomial 0x1EDC6F41 (reversed 0x82F63B78) to be used is defined in RFC 3309, which further includes references to the relevant background material.

9 UINT8: **Data publication version**. Values should only be considered relative to each other for data from the same data center. Semantics may vary between data centers but generally larger values denote later and more preferred data. Recommended values: 1 for raw data, 2+ for revisions produced later, incremented for each revision. A value of 0 indicates unknown version such as when data are converted to miniSEED from another format. Changes to this value for user-versioning are not recommended, instead an extra header should be used to allow for user-versioning of different derivatives of the data.

10 UINT8: **Length of identifier**. Length, in bytes, of source identifier in field 13.

11 UINT16: **Length of extra headers**. Length, in bytes, of extra headers in field 14. If no extra headers, set this value to 0.

12 UINT32: **Length of data payload**. Length, in bytes, of data payload starting in field 15. If no data payload is present, set this value to 0. Note that no padding is permitted in the data record itself, although padding may exist within the payload depending on the type of encoding used.

13 CHAR: **Source identifier**. A unique identifier of the source of the data contained in the record. Recommended to use URI-based identfiers. Commonly an FDSN Source Identifier.

14 JSON: **Extra header fields**. Extra fields of variable length encoded in JavaScript Object Notation (JSON) Data Interchange Standard as defined by ECMA-404. It is strongly recommended to store compact JSON, containing no non-data white space, in this field to avoid wasted space.

A reserved set of headers fields is defined by the FDSN, see *Extra Headers*. Other header fields may be present and should be defined by the organization that created them.

15 encoded: **Data payload**. Length indicated in field 12, encoding indicated in field 5.

1.2 Data Encodings

Data payload encodings in the format are identified by a code (number). These codes are assigned by the FDSN. A list of valid codes are as follows:

Code	Description
0	Text, UTF-8 allowed, use ASCII for maximum portability, no structure defined
1	16-bit integer (two's complement), little endian byte order
3	32-bit integer (two's complement), little endian byte order
4	32-bit floats (IEEE float), little endian byte order
5	64-bit floats (IEEE double), little endian byte order
10	Steim-1 integer compression, big endian byte order
11	Steim-2 integer compression, big endian byte order
19	Steim-3 integer compression, big endian (not in common use in archives)
100	Opaque data - only for use in special scenarios, not intended for archiving

Overview and description of the Steim-1 and Steim-2 compression encodings may be found in the SEED 2.4 manual, Appendix B.

1.2.1 Retroactive future encodings

New data encodings may be added to the format in the future without incrementing the format version. There is no default encoding, readers must check the encoding value to determine if the encoding is supported.

1.2.2 Retired encoding values, not allowed in this specification

The following numeric codes were used in earlier miniSEED versions and should *not* be used for encodings defined in the future:

2 24-bit integers

12 GEOSCOPE multiplexed format 24-bit integer

13 GEOSCOPE multiplexed format 16-bit gain ranged, 3-bit exponent

14 GEOSCOPE multiplexed format 16-bit gain ranged, 4-bit exponent

15 US National Network compression

16 CDSN 16-bit gain ranged

17 Graefenberg 16-bit gain ranged

18 IPG-Strasbourg 16-bit gain ranged

30 SRO format

31 HGLP format

32 DWWSSN gain ranged format

33 RSTN 16-bit gain ranged format

1.3 Extra Headers

The extra headers are encoded in the JSON Data Interchange Standard as defined by ECMA-404. All extra headers are optional as far as the format is concerned.

Extra headers must follow these rules:

- All extra headers are contained in an anonymous (unnamed) object
- All entries are key-value pairs where values can be any valid JSON type
- The key value of "FDSN" in the root object is reserved for values defined by the FDSN

1.3.1 Validation

Extra headers are specified and documented in JSON Schema.

1.3.2 FDSN Reserved Headers

The **"FDSN"** key at the root of the extra headers are reserved for definition by the FDSN.

See *FDSN Reserved Headers* for documentation of these headers.

See *A: Extra header examples* for an example of FDSN extra headers.

1.3.3 Guidelines for Extension

Network operators, manufacturers, data centers, users and other agencies may wish to define their own extra headers. The following guidelines should be considered, in particular for data that is expected to reside in a public repository:

- All headers defined by a group or agency should be contained in a JSON object that is the value of a key in the root container with a clearly identifiable name, i.e. at the same level as **"FDSN"**.
- Creation of a JSON Schema document describing the field(s) is strongly recommended. The schema should be submitted to the FDSN to be made publically available.
- Headers that would be generally useful should be submitted to the FDSN for consideration of being added to the reserved headers for general definition and use.

Multiple JSON Schema documents are easily combined for use in validating extra headers that may contain headers defined in multiple schema documents.

See *Non-FDSN extra headers* for an example of non-FDSN extra headers.

1.4 FDSN Reserved Headers

The **"FDSN"** key at the root of the *Extra Headers* are reserved for definition by the FDSN. See *A: Extra header examples* for an example of FDSN extra headers.

The FDSN reserved headers are defined and documented in JSON Schema.

Download the FDSN Extra Header schema v1.0.

The documentation and schema of these headers may be browsed here:

When not present, the boolean values in the FDSN reserved headers should be considered to be *false* unless otherwise documented. Such values do not need to be included when the value is *false*.

1.5 Background

The Standard for the Exchange of Earthquake Data (SEED) was adopted by the FDSN in the 1987 and served as the dominant standard for seismological research data archiving and exchange for many decades.

The previous specification of miniSEED, latest 2.4, is defined as a subset of SEED that contains only data records.

1.5.1 Changes relative to 2.4

An overview of significant changes between miniSEED 2.4 and this specification:

- Adoption of FDSN Source Identifiers, replacing independent SEED codes (network, station, location, channel).
- Incorporate critical details previously in blockettes (actual sample rate, encoding, microseconds) into the fixed section of the data header
- Increase sample rate/period representation to a 64-bit floating point value
- Increase start time resolution to nanoseconds
- Specify fixed byte order (little endian) for the binary portions of the headers and define a byte order for each data encoding
- Drop legacy data encodings and reserve their values so they are not used again in the future
- Add a format version
- Add a data publication version
- Add CRC field for validating integrity of record
- Add a "mass position off scale" flag
- Add "Recenter" (mass, gimbal, etc.) headers
- Add "ProvenanceURI" header to identify provenance documentation
- Replace the blockette structure with flexible extra header construct:
  - Specify a reserved set of extra headers defined by the FDSN, provide schema for validation
  - Previous flags and blockette contents defined in reserved extra headers

  - Allow arbitrary headers to be included in a record
- Remove the restriction on record length to be powers of 2, allow variable length

Near complete preservation of miniSEED 2.4 data. Information that is not retained is limited to: clock model specification per timing exception (current specification only allows a single clock model specification per record), Blockettes 400 (Beam) & 405 (Beam Delay) and Blockette 2000 (Opaque Data).

1.6 Appendix

1.6.1 A: Extra header examples

The example extra header structures below include additional white space and formatting for readability, which should not be used in an actual record.

FDSN reserved extra headers

A relatively simple example of reserved FDSN extra headers that contains a timing quality value and an event detection is:

```
{
  "FDSN": {
    "Time": {
      "Quality": 100,
      "Correction": 1.234
    },
    "Event": {
      "Begin": true,
      "End": true,
      "InProgress": true,
      "Detection": [
        {
          "Type": "MURDOCK",
          "SignalAmplitude": 80,
          "SignalPeriod": 0.4,
          "BackgroundEstimate": 18,
          "Wave": "DILATATION",
          "Units": "COUNTS",
          "OnsetTime": "2022-06-05T20:32:39.120000Z",
          "MEDSNR": [ 1, 3, 2, 1, 4, 0 ],
          "MEDLookback": 2,
          "MEDPickAlgorithm": 0,
          "Detector": "Z_SPWWSS"
        }
      ]
    }
  }
}
```

An example of the reserved FDSN headers defined by the FDSN is provided below. In this (pathalogical) illustration, all reserved fields are represented.

```
{
  "FDSN": {
    "Time": {
      "Quality": 100,
      "Correction": 1.234,
      "MaxEstimatedError": 1e-06,
      "LeapSecond": 1,
      "Exception": [
        {
          "Time": "2022-05-06T20:32:41.12Z",
          "VCOCorrection": 50.7812,
          "ReceptionQuality": 80,
          "Count": 23,
          "Type": "Valid Timemark",
          "ClockStatus": "SNR=48,51,51,50,50,48,46,48,48,45,45"
        },
        {
          "Time": "2022-05-06T20:32:42.185Z",
          "VCOCorrection": 44.1313,
          "ReceptionQuality": 55,
          "Count": 19690,
          "Type": "Missing timemarks",
          "ClockStatus": "SNR=50,48,46,48,48,45,45"
        }
      ]
    },
    "Event": {
      "Begin": true,
      "End": true,
      "InProgress": true,
      "Detection": [
        {
          "Type": "GENERIC",
          "SignalAmplitude": 80,
          "SignalPeriod": 0.4,
          "BackgroundEstimate": 18,
          "OnsetTime": "2022-05-06T20:32:39.120000Z",
          "Detector": "Dalek STA/LTA"
        },
        {
          "Type": "MURDOCK",
          "SignalAmplitude": 80,
          "SignalPeriod": 0.4,
          "BackgroundEstimate": 18,
          "Wave": "DILATATION",
          "Units": "COUNTS",
          "OnsetTime": "2022-05-06T20:32:39.185000Z",
          "MEDSNR": [ 1, 3, 2, 1, 4, 0 ],
          "MEDLookback": 2,
          "MEDPickAlgorithm": 0,
          "Detector": "Z_SPWWSS"
        }
      ]
```

(continues on next page)

(continued from previous page)

```
    },
    "Calibration": {
      "Sequence": [
        {
          "Type": "Step",
          "BeginTime": "2022-05-06T20:32:39.120000Z",
          "Steps": 12,
          "StepFirstPulsePositive": true,
          "StepAlternateSign": true,
          "Trigger": "AUTOMATIC",
          "Continued": false,
          "Amplitude": 1345,
          "InputUnits": "COUNTS",
          "Duration": 603.456,
          "SinePeriod": 5.0,
          "StepBetween": 500.0,
          "InputChannel": "CAL",
          "ReferenceAmplitude": 45.8,
          "Coupling": "RESISTIVE",
          "Rolloff": "Description of rolloff"
        },
        {
          "Type": "Step",
          "EndTime": "2022-05-06T20:32:39.120000Z"
        },
        {
          "Type": "Sine",
          "BeginTime": "2022-05-06T20:32:39.120000Z",
          "EndTime": "2022-05-06T20:32:39.120000Z",
          "Trigger": "MANUAL",
          "Continued": true,
          "Amplitude": 1345,
          "InputUnits": "COUNTS",
          "AmplitudeRange": "PEAKTOPEAK",
          "SinePeriod": 5.0,
          "InputChannel": "CAL",
          "ReferenceAmplitude": 45.8,
          "Coupling": "RESISTIVE",
          "Rolloff": "Description of rolloff"
        },
        {
          "Type": "PseudoRandom",
          "BeginTime": "2022-05-06T20:32:39.120000Z",
          "EndTime": "2022-05-06T20:32:39.120000Z",
          "Trigger": "MANUAL",
          "Amplitude": 0.0001,
          "InputUnits": "M/S",
          "Duration": 300,
          "InputChannel": "CAL",
          "ReferenceAmplitude": 45.8,
          "Coupling": "CAPACITIVE",
          "Rolloff": "Very randomly",
```

(continues on next page)

(continued from previous page)

```
          "Noise": "White"
        },
        {
          "Type": "Generic",
          "BeginTime": "2022-05-06T20:32:39.120000Z",
          "EndTime": "2022-05-06T20:32:39.120000Z",
          "Trigger": "MANUAL",
          "Amplitude": 1345,
          "Duration": 100
        }
      ]
    },
    "Recenter": {
      "Sequence": [
        {
          "Type": "Gimbal",
          "BeginTime": "2022-05-06T20:32:39.120000Z",
          "EndTime": "2022-05-06T20:32:39.120000Z",
          "Trigger": "AUTOMATIC"
        },
        {
          "BeginTime": "2022-05-06T20:32:40.120000Z"
        }
      ]
    },
    "Flags": {
      "MassPositionOffscale": true,
      "AmplifierSaturation": true,
      "DigitizerClipping": true,
      "Spikes": true,
      "Glitches": true,
      "FilterCharging": true,
      "StationVolumeParityError": true,
      "LongRecordRead": true,
      "ShortRecordRead": true,
      "StartOfTimeSeries": true,
      "EndOfTimeSeries": true,
      "MissingData": true,
      "TelemetrySyncError": true
    },
    "Logger": {
      "Model": "DM24",
      "Serial": "A4567"
    },
    "Sensor": {
      "Model": "T240",
      "Serial": "123123"
    },
    "Clock": {
      "Model": "P273T11N16",
      "Serial": "24A00000"
    },
```

(continues on next page)

(continued from previous page)

```
    "ProvenanceURI": "prov:sp001_wf_f84fb9a",
    "DataQuality": "D",
    "Sequence": 123456
  }
}
```

Non-FDSN extra headers

An example of reserved FDSN headers combined with other top-level headers, illustrating how custom headers may be added.

```
{
  "FDSN": {
    "Time": {
      "Quality": 90
    }
  },
  "Manufacturer123": {
    "Metadata": {
      "FilamentCurrent": 16.4,
      "HyperCoordinates": "1.1789:965402:73324@3.14159"
    }
  },
  "OperatorXYZ": {
    "DSP": {
      "PeakRMS": 2067,
      "RMSWindow": 10.5
    }
  }
}
```

1.6.2 B: Reference data

The reference data set is intended as an illustration of properly constructed miniSEED 3 and to be used by software implementors during development and testing.

The set contains multiple examples of data records illustrating different characteristics and data encodings of the from. Each example is available in miniSEED 3, JSON, and human-readable text representations.

The whole data set may be downloaded from documentation repository.

All time series in the reference set that contain series are the same expanding sinusoid signal.

#	Description	Download
1	Text payload	mseed3 JSON Text
2	Event detection headers only, no data payload	mseed3 JSON Text
3	Sinusoid series encoded as 16-bit integers	mseed3 JSON Text
4	Sinusoid series encoded as 32-bit integers	mseed3 JSON Text
5	Sinusoid series encoded as 32-bit IEEE float	mseed3 JSON Text
6	Sinusoid series encoded as 64-bit IEEE float	mseed3 JSON Text
7	Sinusoid series encoded as Steim-1 compressed integers	mseed3 JSON Text
8	Sinusoid series encoded as Steim-2 compressed integers	mseed3 JSON Text
9	Series with time quality, correction, event detections headers	mseed3 JSON Text
10	Series with some FDSN and non-FDSN extra headers	mseed3 JSON Text
11	Series with all FDSN extra headers (unrealistic)	mseed3 JSON Text

1.6.3 C: Mapping from miniSEED 2.4

The following list of miniSEED 2.4 format structures specifies the mapping of all fields to this specification. In this listing the following abbreviations are used:

EH Extra Header (of this specification), with path

SID Source Identifier (of this specification)

FSDH Fixed Section Data Header (in either specification)

miniSEED 2.4 Fixed Section Data Header (FSDH)

Field	Description	This specification
1	Sequence number	EH: *FDSN.Sequence*
2	Data header/quality indicator	EH: *FDSN.DataQuality* A data center may choose to translate miniSEED 2.4 quality values to publication versions with the following mapping: 2.4 quality / Pub. version R / 1 D / 2 Q / 3 M / 4
3	Reserved byte	[no mapping]
4	Station identifier code	Incorporated into SID, FSDH field 13
5	Location identifier	Incorporated into SID, FSDH field 13
6	Channel identifier	Incorporated into SID, FSDH field 13
7	Network code	Incorporated into SID, FSDH field 13
8	Record start time	FSDH field 4
9	Number of samples	FSDH field 7
10	Sample rate factor	Incorporated into FSDH field 6
11	Sample rate multiplier	Incorporated into FSDH field 6
12	**Activity flags:** 0 = calibration signals 1 = time correction applied 2 = begining of event 3 = end of event 4 = + leap second included 5 = - leap second included 6 = event in progress	**FSDH and Extra headers:** FSDH field 3, bit 0 [no mapping] *FDSN.Event.Begin* *FDSN.Event.End* *FDSN.Time.LeapSecond* *FDSN.Time.LeapSecond* *FDSN.Event.InProgress*
13	**I/O flags, bits:** 0 = Sta. volume parity error 1 = Long record read 2 = Short record read 3 = Start of time series 4 = End of time series 5 = Clock locked	**FSDH and Extra headers:** *FDSN.Flags.StationVolumeParityError* *FDSN.Flags.LongRecordRead* *FDSN.Flags.ShortRecordRead* *FDSN.Flags.StartOfTimeSeries* *FDSN.Flags.EndOfTimeSeries*

Blockette 100 (Sample Rate)

Field	Description	This specification
3	Actual sample rate	FSDH field 6
4	Flags (undefined)	[no mapping]
4	Reserved byte	[no mapping]

Blockette 200 (Generic Event Detection)

Field	Description	This specification
3	Signal amplitude	EH: *FDSN.Event.Detection.SignalAmplitude*
4	Signal period	EH: *FDSN.Event.Detection.SignalPeriod*
5	Background estimate	EH: *FDSN.Event.Detection.BackgroundEstimate*
6	**Event detection flag bits:** 0 = dil./comp. wave 1 = units	**Extra headers:** *FDSN.Event.Detection.Wave* *FDSN.Event.Detection.Units*
7	Reserved byte	[no mapping]
8	Signal onset time	EH: *FDSN.Event.Detection.OnsetTime*
9	Detector name	EH: *FDSN.Event.Detection.Detector*

Blockette 201 (Murdock Event Detection)

Field	Description	This specification
3	Signal amplitude	EH: *FDSN.Event.Detection.SignalAmplitude*
4	Signal period	EH: *FDSN.Event.Detection.SignalPeriod*
5	Background estimate	EH: *FDSN.Event.Detection.BackgroundEstimate*
6	**Event detection flag bits:** 0 = dil./comp. wave	**Extra headers:** *FDSN.Event.Detection.Wave*
7	Reserved byte	[no mapping]
8	Signal onset time	EH: *FDSN.Event.Detection.OnsetTime*
9	Signal-to-noise values	EH: *FDSN.Event.Detection.MEDSNR* (array)
10	Lookback value	EH: *FDSN.Event.Detection.MEDLookback*
11	Pick algorithm	EH: *FDSN.Event.Detection.MEDPickAlgorithm*
12	Detector name	EH: *FDSN.Event.Detection.Detector*

Blockette 300 (Step Calibration)

Field	Description	This specification
3	Start calib	EH: *FDSN.Calibration.Sequence.Begintime*
4	Num of calibs	EH: *FDSN.Calibration.Sequence.Steps*
5	**Calibration flags:** 0 = first pulse 1 = cal alt sign 2 = cal auto 3 = cal cont	**Extra headers:** *FDSN.Calibration.Sequence.StepFirstPulsePositi* *FDSN.Calibration.Sequence.StepAlternateSign* *FDSN.Calibration.Sequence.Trigger* *FDSN.Calibration.Sequence.Continued*
6	Duration of step	EH: *FDSN.Calibration.Sequence.Duration*
7	Time between steps	EH: *FDSN.Calibration.Sequence.StepBetween*
8	Amp. of cal. signal	EH: *FDSN.Calibration.Sequence.Amplitude*
9	Channel cal. signal	EH: *FDSN.Calibration.Sequence.InputChannel*
10	Reserved byte	[no mapping]
11	Reference amplitude	EH: *FDSN.Calibration.Sequence.ReferenceAmplitude*
12	Coupling of signal	EH: *FDSN.Calibration.Sequence.Coupling*
13	Rolloff of filters	EH: *FDSN.Calibration.Sequence.Rolloff*

Blockette 310 (Sine Calibration)

Field	Description	This specification
3	Start calib	EH: *FDSN.Calibration.Sequence.Begintime*
4	Reserved byte	[no mapping]
5	**Calibration flags:** 2 = cal auto 3 = cal cont 4 = PtoP amp 5 = ZtoP amp 6 = RMS amp	**Extra headers:** *FDSN.Calibration.Sequence.Trigger* *FDSN.Calibration.Sequence.Continued* *FDSN.Calibration.Sequence.AmplitudeRange* *FDSN.Calibration.Sequence.AmplitudeRange* *FDSN.Calibration.Sequence.AmplitudeRange*
6	Duration of cal	EH: *FDSN.Calibration.Sequence.Duration*
7	Period of signal	EH: *FDSN.Calibration.Sequence.Period*
8	Amp. of cal. signal	EH: *FDSN.Calibration.Sequence.Amplitude*
9	Channel cal. signal	EH: *FDSN.Calibration.Sequence.InputChannel*
10	Reserved byte	[no mapping]
11	Reference amplitude	EH: *FDSN.Calibration.Sequence.ReferenceAmplitude*
12	Coupling of signal	EH: *FDSN.Calibration.Sequence.Coupling*
13	Rolloff of filters	EH: *FDSN.Calibration.Sequence.Rolloff*

Blockette 320 (Pseudo-random Calibration)

Field	Description	This specification
3	Start calib	EH: *FDSN.Calibration.Sequence.Begintime*
10	Reserved byte	[no mapping]
5	**Calibration flags:** 2 = cal auto 3 = cal cont 4 = Random amps	**Extra headers:** *FDSN.Calibration.Sequence.Trigger* *FDSN.Calibration.Sequence.Continued* *FDSN.Calibration.Sequence.AmplitudeRange*
6	Duration of cal	EH: *FDSN.Calibration.Sequence.Duration*
7	PtoP amp. of steps	EH: *FDSN.Calibration.Sequence.Amplitude*
8	Channel cal. signal	EH: *FDSN.Calibration.Sequence.InputChannel*
9	Reserved byte	[no mapping]
10	Reference amplitude	EH: *FDSN.Calibration.Sequence.ReferenceAmplitude*
11	Coupling of signal	EH: *FDSN.Calibration.Sequence.Coupling*
12	Rolloff of filters	EH: *FDSN.Calibration.Sequence.Rolloff*
13	Noise type	EH: *FDSN.Calibration.Sequence.Noise*

Blockette 390 (Generic Calibration)

Field	Description	This specification
3	Start calib	EH: *FDSN.Calibration.Sequence.Begintime*
10	Reserved byte	[no mapping]
5	**Calibration flags:** 2 = cal auto 3 = cal cont	**Extra headers:** *FDSN.Calibration.Sequence.Trigger* *FDSN.Calibration.Sequence.Continued*
6	Duration of cal	EH: *FDSN.Calibration.Sequence.Duration*
7	Amplitude of signal	EH: *FDSN.Calibration.Sequence.Amplitude*
8	Channel cal. signal	EH: *FDSN.Calibration.Sequence.InputChannel*
9	Reserved byte	[no mapping]

Blockette 395 (Calibration Abort)

Field	Description	This specification
3	End calib	EH: *FDSN.Calibration.Sequence.Endtime*
10	Reserved byte	[no mapping]

Blockette 400 (Beam), Blockette 405 (Beam Delay)

No mapping for these blockettes

Blockette 500 (Timing)

Field	Description	This specification
3	VCO correction	EH: *FDSN.Time.Exception.VCOCorrection*
4	Time of exception	EH: *FDSN.Time.Exception.Time*
5	Microsecond offset	[included in record start time]
6	Reception quality	EH: *FDSN.Time.Exception.ReceptionQuality*
7	Exception count	EH: *FDSN.Time.Exception.Count*
8	Exception type	EH: *FDSN.Time.Exception.Type*
9	Clock model	EH: *FDSN.Clock.Model*
10	Clock status	EH: *FDSN.Time.Exception.ClockStatus*

Blockette 1000 (Data Only SEED)

Field	Description	This specification
3	Encoding format	FSDH field 5
4	Word order	[no mapping, no longer needed]
5	Data record length	[no mapping, no longer needed]
6	Reserved byte	[no mapping]

Blockette 1001 (Data Extension)

Field	Description	This specification
3	Timing quality	EH: *FDSN.Time.Quality*
4	Microsecond offset	Incorporated into FSDH field 4
5	Reserved byte	[no mapping]
6	Frame count	[no mapping]

Blockette 2000 (Variable Length Opaque)

No mapping for this blockette. The opaque data encoding may be used to specify an opaque payload for nearly equivalent functionality.

Unsupported miniSEED 2.4 content

The following defined information in miniSEED 2.4 cannot be represented in this specification:

- Clock model specification per timing exception. Current specification only allows a single clock model specification per record.
- Blockettes 400 (Beam) & 405 (Beam Delay)
- Blockette 2000 (Opaque Data)

1.7 Software

A variety of open-source software packages are available to that support the miniSEED format.

1.7.1 Validation

The mseed3-utils project includes the following programs:

- mseed3-validator - a validator for the miniSEED 3 formatted files
- mseed2text - convert miniSEED 3 format to a text representation
- mseed2json - convert miniSEED 3 format to a JSON representation

Download mseed3-utils releases

1.7.2 mseedconvert

The mseedconvert program converts miniSEED between versions and data encodings.

Download mseedconvert releases

1.7.3 libmseed

A general library (C language) for reading and writing miniSEED. Release versions 3.x support both this specification and version 2 of the format.

Download libmseed releases

1.8 Changes

Changes to this specification are listed below.

1.8.1 April 2023

- Specification revision 2023-01-18 adopted by FDSN.
- Reference data were refined following the specification revision.

1.8.2 2021

- Initial specification of miniSEED 3