Next generation miniSEED

Background:

Many FDSN members recognize that the current two-character network code needs to expand. The miniSEED format is a fixed-length field format and expanding the network code would render the format incompatible with the current release. Such a small, but disruptive change affords the opportunity to consider other changes to the format, allowing the FDSN to address historical issues and create a new foundation for current and future use.

2016-4-1

Next generation miniSEED

Overview:

We have developed a list of changes to be considered for the next generation of miniSEED. The rationale is provided for each change. **This is intended to serve as the start of a format development process.**

As a starting point, a SEED-style straw man miniSEED definition has been created that includes most of the changes contained in this document.

Guiding principles

- Fix accumulation of historical issues, for example the byte-order flag Catch-22.
- Features that are not used by the vast majority of existing records should *probably* not be included.
- Look for new features or changes that support broad use but not too far, focus on solving FDSN challenges. Ask “Why?” more than “Why not?”.
- Multi - decade horizon, i.e. create a data format that is sensible in 2050 and beyond.

1. Expand the network code

Change: Expand the FDSN network code field to accommodate more networks.

Suggestion: Expand to 6 characters, with convention for temporary network codes as follows:

“xxxxYY”, where ‘x’ are unique identifiers and YY are the last two digits of the start year of the deployment, e.g. 16 for 2016. Temporary network codes will still begin with the letters X, Y, Z, or a numeral from 0-9. Examples: ‘XY2018’, ‘2A2017’.

Rationale: Space enough for identifying a vast number of networks and improved identification of temporary networks.

2. Add a miniSEED version field

Change: Include a version identifier in the format.

Suggestion: Use an 8-bit unsigned integer to identify the format version. This allows 255 versions. The straw man definition sets this value to 3.

Rationale: Explicitly identifies the format version and helps identify, along with other fields, a miniSEED record. Rather obvious.

3. Add a data version field

Change: Include a data version identifier in the format.

Suggestion: Use an 8-bit unsigned integer to identify the version of the contained data. This allows 255 versions. The straw man suggests starting at 1 with later versions using larger values.

Rationale: Explicitly identifies the data version, which has been missing from SEED. The "quality" indicator has been used as form of versioning, but it has meaning beyond a version indicator.

4. Move important Blockette details into fixed section of the header

Change: Many critical details such as byte order, encoding format, actual sample rate, record length and microsecond time resolution are in Blockettes in SEED 2.4. Move them into the fixed section of the header.

Suggestion: Create fields in the fixed header for the critical details.

Rationale: Puts required, important information where it belongs and reduces the need for blockettes.

5. Simplify & improve the record start time

Change: Simplify the record start time encoding and allow higher resolution.

Suggestion: Use a representation of UTC encoded as microseconds since midnight 1 January 1970 not including leap seconds. Use a bit flag to identify leap seconds and remove ambiguity. This a microsecond version of the common POSIX/Unix time.

Rationale: This representation is smaller, more readily usable and higher resolution than the “BTime” representation with a huge range. The definition of “Seconds Since the Epoch” is defined by the IEEE in POSIX as a reference, scaling to microseconds.

Considerations: This could also be nanoseconds.

6. Combine the 3 bit-flag fields

Change: Combine the 3 bit-flag fields: Activity flags, I/O flags & clock flags, and Data quality flags into a single field and eliminate the least used flags.

Suggestion: Use an 8-bit Flags field to represent:
0: Byte order
1: Start time occurred during a leap second
2: A positive leap second occurred during this record
3: A negative leap second occurred during this record
4: Time tag questionable
5: Clock locked

Rationale: Reduce bit flags to what is needed and has been most commonly used.

6 Continued. Dropped bit-flag fields

Bit flags that would not be present:

: Calibration signals present
: Time correction applied
: Beginning of an event
: End of the event
: Event in progress
: Station volume parity error possibly present
: Long record read (possibly no problem)
: Short record read (record padded)
: Start of time series
: End of time series
: Amplifier saturation detected (station dependent)
: Digitizer clipping detected
: Spikes detected
: Glitches detected
: Missing/padded data present
: Telemetry synchronization error
: A digital filter may be charging

7. Eliminate the time correction field

Change: Eliminate the time correction field and any requirement to check bit flags to determine if the timestamp represents the actual start time.

Suggestion: Remove the field and associated bit flag.

Rationale: The requirement to check the correction field and “applied” bit flag adds complexity and has lead to errors interpreting time in the record.

Consideration: What we lose is a notification and record that a correction that has been applied.

8. Forward compatibility mapping

Description: Forward compatibility from miniSEED 2.4 to miniSEED 3 should be a goal.

Suggestion: Document the mapping between 2.4 -> 3, clarifying how fields move and which are dropped in the SEED manual.

9. General compression and opaque data encodings

Change: Define encodings for a general compression method for fundamental data types. Also define an encoding for opaque data.

Suggestion:

Encoding 50: 32-bit integers, general compressor
Encoding 51: 32-bit IEEE floats, general compressor
Encoding 52: 64-bit IEEE floats, general compressor
Encoding 100: Opaque data

A good candidate for the general compressor is Brotli by Google because a) is designed for and efficient for small payload sizes, b) will be an IETF standard, in progress now, c) a general compressor, not dependent on data type, d) an open source reference implementation is available with support in most computing environments.

Rationale: Benefit from broadly-used compression techniques and a much larger set of resources. Opaque data encoding simplifies and replaces the SEED 2.4 Blockette 2000 payload functionality.

10. Add CRC field for validating integrity

Change: Add field for CRC or other checksum to be used for validating data integrity.

Suggestion: Add 32-bit field for a CRC-32 value. For non-opaque data this is the CRC of the decoded data, for opaque data this is the CRC of the opaque payload.

Rationale: Provides data validation for all encodings. Using a CRC of the decoded data allows for additional validation of the decoding process.

Considerations: CRC could also include most of the header. This becomes more complicated to compute if the CRC is for part of the header and the decoded data. A CRC for part of the header and the encoded data would be easier, but losing the advantage of validating the decoding process.

11. Expand the channel codes

Change: Expand the channel identifier/codes field.

Suggestion: Expand the 3 character field to 4, 5 or 6 characters and define additional instrument codes.

Rationale: No further instrument codes are available, they have all been allocated so new instruments cannot be easily defined.

Consideration: Incorporate the common use of location identifiers directly into the channel codes.

12. Expand the location identifier and require a value

Change: Expand the location identifier field and require a value.

Suggestion:
a) Expand the 2 character location identifier field to 3,4,5 or 6 characters.
b) Require a value to make its handling synonymous with the other identifiers (network, station, channel).

Rationale:
a) For future "large N" deployments containing 1000s of sensors the current value is inadequate. Strategies combining the station and location fields become cumbersome.
b) The special case of empty location identifiers in SEED 2.4 requires special handling in a lot of software, leading to errors. Empty strings are challenging identifiers to specify and display.

13. Fixed-point data sample encoding

Change: Include a fixed-point, constant resolution data sample encoding and define a representation

Suggestion: Evaluate the need for this and determine a representation. Two options:

1) Adopt Q for a number format, is binary format defined?
2) Define an encoding as a single resolution value followed by integer data. Fixed-point values are restored by shifting the decimal place according to the resolution value.

Rationale: Increasingly, rational data sample values are stored in miniSEED. The supported IEEE floating point values cannot store the original resolution of the values. For example, a value of "22.1" stored in a float, even if stored exactly, does not include qualification that the value is only valid to a single decimal.

Considerations: There is no widely adopted and supported standard for fixed-point representation.

14. No SEED 2.4 blockettes, include support for opaque headers.

Change: a) Remove support for blockettes. b) Include support for opaque headers.

Suggestion: a) Remove the blockette count field, the blockette offset field and define no blockettes. b) Use a single byte to include a count of opaque headers and define the header values to be UTF-8 text.

Rationale: Simplifies the header.
With
a) core information from blockettes 100, 1000 and 1001 moved into the header and
b) opaque data encoding support and opaque headers, supplanting Blockette 2000 functionality,
the need for blockettes is significantly reduced. The remaining blockettes are rarely used.

A low cost, single byte is used to indicate the count of opaque headers.

15. Eliminate sequence numbers

Change: Eliminate sequence number field.

Suggestion: Replace the field with a simple 2-character identification sequence of “MS”.

Rationale: The sequence numbers take 6 bytes and are part of the full SEED volume design. They are of limited use beyond the data collection center. Except for their use to identify the start of a record they are routinely ignored by many miniSEED readers.

The functionality of determining data record order in a stream independently from the timestamp can be accomplished either as
a) a feature of a transport protocol
or
b) using opaque headers

16. Eliminate the timing quality field

Suggestion: Remove the 32-bit timing quality field.

Rationale: The header includes bit flags for “time tag questionable” and “clock locked”. The addition of a vendor-specific, percent timing quality may not be adding much more information. The definition of what these values mean is not included in station metadata, rendering the future interpretation challenging or impossible.

Network operators could alternatively use a separate channel for timing quality or an opaque header.

Consideration: An improved timing quality designation with a clearer meaning: estimated maximum clock error in a unit of time. For example, 200 microseconds.

17. Variable record lengths

Change: Remove the limitation that record lengths must be 2^factor

Suggestion: Change the record length and sample count fields to 32-bit integers and define the length value to be in bytes.

Rationale: Variable record lengths are good for catering to a wide variety of uses from low latency applications to data processing, avoiding padding in “unfilled” records to save storage and transmission, etc.

Considerations: Very large records could be problematic for a number of reasons including more intensive to create & parse, risk of corruption rendering a larger amount of data unusable, etc. We might consider making the record length and sample count fields 16-bit integers, limiting records to a maximum of 65,535 bytes and the same number of samples.