

# Norwegian National Seismic Network

## Technical Report No. 20



A White Paper about  
MiniSEED for LISS  
and data compression using  
Steim1 and Steim2

version 01.00

Prepared by

**Mauro Mariotti**  
SARA snc – Perugia – Italy  
[www.sara.pg.it](http://www.sara.pg.it)  
[mariotti@infoeq.it](mailto:mariotti@infoeq.it)

**Terje Utheim**  
University of Bergen – Bergen – Norway  
[www.geo.uib.no](http://www.geo.uib.no)  
[terje.utheim@geo.uib.no](mailto:terje.utheim@geo.uib.no)

**Dept. of Earth Science, University of Bergen**  
*Allégt.41, N-5007 Bergen, Norway*  
*Tel: +47-55-583600 Fax: +47-55-583660 E-mail: [seismo@geo.uib.no](mailto:seismo@geo.uib.no)*

**February 2006**

## Introduction

The Live Internet Seismic Server (LISS) is an internet-based distribution mechanism allowing near-real-time data flow from seismic stations around the world to an essentially unlimited number of clients. The clients can be any software capable to read miniSEED packet using TCP/IP sockets. The LISS is claimed to be full functional on Seiscomp, EarthWorm and Antelope. Considering the importance to have a good interchange standard for seismic data miniSEED appeared to be a good choice considering it is widely known and used in the world. The authors of this document Mauro Mariotti and Terje Utheim experienced many problems in coding mSEED starting from the SEED official manual (Standard for the Exchange of Earthquake Data Reference Manual SEED format Version 2.4 August 2004).

This document is prepared with the intent of improve the understanding of the mSEED standard and their popular data compression formats. According to the SEED Reference manual: The Standard for the Exchange of Earthquake Data (SEED) is an international standard format for the exchange of digital seismological data. SEED was designed for use by the earthquake research community, primarily for the exchange between institutions of unprocessed earth motion data. It is a format for digital data measured at one point in space and at equal intervals of time.

We hope it will be useful for the LISS data distribution mechanism where a lot of confusion is present at the moment of the preparation of this document.

### The essential information about miniSEED

Many internet service are widely used to transmit analogue information such as radio channels, audio streaming and so on. Seismic data are slightly different being oriented to lower frequency, higher dynamic range and higher time precision. Audio streaming (like radios or voice over IP application) can tolerate a certain grade of data-loss, this is not allowed in seismic processing but the capabilities of the IP protocol are so wide that good and reliable connection can be established without big problems. While SEED is a complex standard with a large set of data packets, called Blockettes designed to distribute in an harmonized format and protocol all the needed information used to process seismic data, miniSEED would meet the limited requirements of near real time data exchange using TCP/IP.

In SEED blockettes are numbered with codes going from 1 to around 2000 (even if this doesn't mean there are two thousands of blockettes types). In miniSEED basicly only blockette number 1000 and 1001 are used.

Before proceed we recomend the reader to examine the box in this page.

### How Binary Data Fields are Described in This Manual

(direct extract from the SEED Reference Manual pages 33-34)

Throughout this manual, we use some conventions to describe the sizes of fields in the SEED format. Here are the binary data types used in fixed headers and in data blockettes:

| <i>Field type</i> | <i>Number of bits</i> | <i>Field description</i>  |
|-------------------|-----------------------|---|
| UBYTE             | 8                     | Unsigned quantity   |
| IBYTE             | 8                     | Two's complement signed quantity  |
| UWORD             | 16                    | Unsigned quantity   |
| WORD              | 16                    | Two's complement signed quantity  |
| ULONG             | 32                    | Unsigned quantity   |
| LONG              | 32                    | Two's complement signed quantity  |
| CHAR * n          | n * 8                 | n chars, each 8 bits long, valid only ASCII 7 bit (7 <sup>th</sup> bit=0) |
| FLOAT             | 32                    | IEEE Floating point number (single precision)                             |

The IEEE floating point format consists of three stored components: a sign (+ or -), an exponent, and a fraction. In the following description of the storage format these notations will be used:

s=sign (bit 31)

e=biased exponent (bit 30-23)

f=fraction(bit 22-0)

The sign is the sign of the fraction. Rather than storing the sign of the exponent a bias is added to the exponent, and the biased exponent is stored. IEEE single precision values occupy one 32 bit word as shown above in 68000 byte order. Bits 0:22 store the 23 bit fraction, bits 23:30 store the 8 bit exponent, and the high order bit 31 stores the sign bit. The 23 bit fraction combined with the implicit leading bit provide 24 bits of precision in normalized numbers. The value of an IEEE single precision floating point number is calculated as:

$$-1s \times 2^{(e-127)} \times 1.f$$

The byte order of a FLOAT is specified in the station identifier blockette 50.

Binary data types are used in the BTIME structure:

| <i>Field type</i> | <i>Number of bits</i> | <i>Field description</i>                   |
|-------------------|-----------------------|--|
| UWORD             | 16                    | Year (e.g., 1987)                          |
| UWORD             | 16                    | Day of Year (Jan 1 is 1)                   |
| UBYTE             | 8                     | Hours of day (0—23)                        |
| UBYTE             | 8                     | Minutes of day (0—59)                      |
| UBYTE             | 8                     | Seconds of day (0—59, 60 for leap seconds) |
| UBYTE             | 8                     | Unused for data (required for alignment)   |
| UWORD             | 16                    | .0001 seconds (0—9999)                     |

NOTE: The BTIME structure differs from the ASCII variable length TIME structure used in the control headers. All binary 32 bit words begin on long-word boundaries, 16 bit words begin on word boundaries, and all bytes on byte boundaries. The fixed portion of the header always ends at the end of a long-word boundary, and each blockette is an integer number of long-words in length. Pack data in either VAX or 68000 swapping order. This swapping key is in the Station Identifier Blockette [50], and may differ for each station. Decoding programs will automatically adapt to specified swapping orders. Negative numbers utilize standard two's complement representation. The data description language in the data dictionary, referred to by the Channel Identifier Blockette [52] for the represented channel, governs byte swapping within the data. Mantissas and exponents for floating point numbers are expressed as binary two's complement integers. The most signi.cant bit of the number (bit 15, or the left most bit) is always set to zero for a positive number, and the most signi.cant bit of the mantissa is in bit 14. For negative numbers, the most signi.cant bit is always set to one, and the integer is in two's complement format.

According to the SEED Reference Manual the mSEED data-packet is composed of the following main fields:

- 1) A fixed header of 48 bytes
- 2) One or two blockettes; always blockette coded as number 1000, optional the blockette coded as number 1001
- 3) Data field

Now we are going to examine each of these fields, each of the following description is composed of 2 paragraphs, first a description of the main field and second a description of the interpretation problems.

It will be demonstrated that major misunderstanding and the inconsistency are present and strange interpretation of the blockette data by EarthWorm and Seiscomp are present too. Furthermore many lacks come from the highly fragmented documentation of the miniSEED and mistakes in the interpretation of the little-endian big-endian notation of data longer than 1 byte. Furthermore all these problems are reflected negatively to the Steim1 and Steim2 code compression, thus is not so complicated, but special attention must be payed to it when related to all the following byte swapping that influence not only the data by themselves but also the time order of the data time series to be represented.

#### **A basic problem, the packet length**

According to the SEED Reference Manual, the SEED format supports virtually any packet length. Data retrieve would be assured by pointers that can point to the next blockettes. For LISS no information and no indications shows that (as we understood after many pains) the packet must be 512 bytes long. Furthermore the packet length is not specified in the fixed header but only in the blockette type 1000.

Anyway be awarned that in miniSEED packet for the LISS operation the length is fixed to 512 bytes.

**Fixed data header fields:**

|    | Field name                             | Type  | Byte position | Length (bytes) |
|----|--|-------|---------------|----------------|
| 1  | Sequence number                        | ASCII | 1-6           | 6              |
| 2  | Data Header quality indicator          | ASCII | 7             | 1              |
| 3  | Reserved byte                          | ASCII | 8             | 1              |
| 4  | Station code                           | ASCII | 9-13          | 5              |
| 5  | Location identifier                    | ASCII | 14-15         | 2              |
| 6  | Channel identifier                     | ASCII | 16-18         | 3              |
| 7  | Network code                           | ASCII | 19-20         | 2              |
| 8  | Record start time                      | BTIME | 21-30         | 10             |
| 9  | Number of samples                      | UWORD | 31-32         | 2              |
| 10 | Sample rate factor                     | WORD  | 33-34         | 2              |
| 11 | Sample rate multiplier                 | WORD  | 35-36         | 2              |
| 12 | Activity flags                         | UBYTE | 37            | 1              |
| 13 | I/O flags                              | UBYTE | 38            | 1              |
| 14 | Data quality flags                     | UBYTE | 39            | 1              |
| 15 | Number of blockettes that follow       | UBYTE | 40            | 1              |
| 16 | Time correction                        | LONG  | 41-44         | 4              |
| 17 | Offset to beginning of data            | UWORD | 45-46         | 2              |
| 18 | Offset to beginning of first blockette | UWORD | 47-48         | 2              |

- 1) The sequence number is a number that progressively increase while each packet is transmitted over the media it usually proceed from zero up to an indefinite number.
- 2) Data Header Quality indicator. From version 2.4 this field represent: D=Data (indetermined quality), R=Raw data with no quality control, Q=Quality Controlled Data, some processes have been applied to the data
- 3) Reserved (set as ASCII = 32, blank)
- 4) Station identifier. Left justify, pad with blanks
- 5) Location identifier. Left justify, pad with blanks
- 6) Channel identifier. Left justify, pad with blanks
- 7) Two characters alphanumeric used to identify the network operator responsible for the data logger. The identifier is assigned by IRIS Data Management Center in consultation with FDSN working group on the SEED format.
- 8) BTIME struct stamping the time of the packet
- 9) UWORD specifying the number of samples in the record
- 10) WORD Sample rate factor
  - If it is > 0 means the factor is in Sample Per Second
  - If it is < 0 means the factor is in Seconds Per Sample
  - If it is = 0 means the factor is in Seconds Per sample (Used for ASCII and OPAQUE data records)
- 11) Sample rate multiplier
- 12) UBYTE: Activity flag (see SEED Reference Manual page 99)
- 13) UBYTE: I/O Flags and clock flags (see SEED Reference Manual page 99)
- 14) UBYTE: Data Quality Flags (see SEED Reference Manual page 99)
- 15) UBYTE: Total number of blockette that follows
- 16) LONG: Time correction: (see SEED Reference Manual page 99)
- 17) UWORD: Offset in bytes of the beginning of data.
- 18) UWORD: Offset in bytes to the first data blockette in this data record. Enter 0 if there are no data blockettes. The first byte in the data record is byte offset 0.

A note in the reference manual explains that all unused flags must be set to zero.

## Problems with the Fixed Header interpretation

- 1) First headache problem comes from the sequence number, there is no explanation of what should happen when the number reaches its maximum representation allowed by the 6 chars ASCII decimal number. In other words when it arrives to 999999 it have to return to 0? When it returns to zero, what will happen to the dataloggers like Seiscomp or EarthWorm? No answer to this point, we can only be confident that from time to time the data server will be restarted closing the TCP/IP socket connection and allowing the client to correctly treat the new data coming with a starting number of 000000.
- 2) The D R Q Field. According to the Reference Manual from version 2.4 the data can be represented as D or R or Q. EarthWorm and Seiscomp (at least for now) only accept D, be aware of it!
- 3) BTIME structure. In the documentation of minSEED there is no explanation on what is the order to be used in storing the YEAR information and the DAY information neither about the fraction of seconds. We still to do not know if this struct is coded in little-endian or big-endian. It should be defined by the blockette 50 but there is no room for the blockette 50 in the miniSEED packet (usually long 512 bytes)
- 4) Beginning of data. Here we have the same problem of the BTIME structure. There is still no way to know how this pointer is represented. And what is the exact meaning of beginning of data? It means the beginning of the blockette? Or the beginning of the binary waveform data? Only a look to a hex dump of an already alive station "explained" that it is referred to the beginning of the waveform data.
- 5) First Blockette. We have the same problem of the byte order. Big-Endian or Little-Endian?
- 6) To make a long story short we can say that in this field the same problem affects also the, Sample Rate Factor and the sample rate multiplier, basically all fields longer than 1 byte.

### *Little Endian and Big Endian Soap Story*

Many web pages explains how the definition little-endians or big-endians describes the method by which information is stored in a computer's memory. The definition can be misleading (at least for not english speaking people). To better understand how this definition is created we have to go back to the Gulliver's Travels where the terms "big-endians" and "little-endians" were it relates to the conflict over what end of the boiled egg that had to be cracked first before you eat it.

To have a full understanding of the term you have to relate an egg as a data packet longer than 8 bytes (i.e. a 2 or 4 bytes integer). From what "end" of the INT or LONG integer you will begin to read (crack or eat) the data? If you begin from the little-end (the LSB) you are a little-endian machine, if you begin from the big-end (the MSB) you are a big-endian machine. Of course the pointer to the data is always the lower (as magnitude) of the address space.

So, let's take the example of the blockette 1000 identifier of the SEED protocol. In hexadecimal format we will write 1000 as 0x03E8. In memory this 2 byte integer can be stored with the sequence 03 E8 or E8 03. In the first case the byte containing 03 is read as first, and it is the MSB (the bigger end) so machines that read the integer in this way are the big-endian. The machine that store the number E8 as first also read it as first and being it the LSB (the little) are called the machine that read the integer beginning from the little-end and are called little-endians.

Architectures that follow this rule are called *little-endian* include Rockwell 6502 and Intel x86.

Architectures that follow the *big-endian* rule include Motorola 68000 and PowerPC.

Problems can also be proposed on what **bit** is read at first or considered first. See the Steim code compression section about it.

#### *References*

[http://www.ling.upenn.edu/courses/Spring\\_2003/ling538/Lecnotes/ADfn1.htm](http://www.ling.upenn.edu/courses/Spring_2003/ling538/Lecnotes/ADfn1.htm)

<http://www.networksorcery.com/enp/ien/ien137.txt>

<http://www.sun.com/realitycheck/ndian.pdf>

## The blockette type 1000 specifications

This is a Data Only SEED Blockette. It is 8 bytes long and has the following structure:

|   | Field name                                    | Type  | Length (bytes) |
|---|---|-------|----------------|
| 1 | Blockette code                                | UWORD | 2              |
| 2 | Offset to the beginning of the next blockette | UWORD | 2              |
| 3 | Encoding format                               | UBYTE | 1              |
| 4 | Word order                                    | UBYTE | 1              |
| 5 | Data record length                            | UBYTE | 1              |
| 6 | Reserved                                      | UBYTE | 1              |

- 1) The blockette code that contains always the number 1000.
- 2) The offset to the beginning of the next blockette
- 3) The encoding format according to the following basic table:

| Code | Encoding Format                                |
|------|--|
| 0    | ASCII text, byte order as specified in field 4 |
| 1    | 16 bit integers                                |
| 2    | 24 bit integers                                |
| 3    | 32 bit integers                                |
| 4    | IEEE floating point                            |
| 5    | IEEE double precision floating point           |
| 10   | STEIM (1) Compression                          |
| 11   | STEIM (2) Compression                          |

- 4) The word order. According to the text in the SEED reference manual a zero (0) stands for little-endian and a one (1) for big endian (page 113)
- 5) The exponent of a base of 2 to specify the record length. In LISS miniSEED it is 9 that means 2 raised at the 9<sup>th</sup> power become 512.
- 6) Reserved

## Problems with the blockette type 1000 interpretation

- 1) Reading this blockette code itself fall in the misunderstanding of the big-endian / little-endian code. The field 4 should be read first but we are not sure if we can point the field 4 (word order) because we don't know how to read this packet. Anyway considering that in miniSEED the first blockette field can be only 0x03E8 or 0xE803 the software should test the very first byte: if it is a 0x03 we are reading a packet wrote from a bigendian machine and if we read 0xE8 we are reading a packet wrote by a little-endian machine.
- 2) The encoding format. Anybody would expect that the basic data format would be supported by the popular Seiscomp and EarthWorm. Don't be so sure. EW absolutely refuse (at least the versions we tested) all encoding format except Steim1, Steim2 and Steim3 (not listed in the table above). We would expected to be able to send data at least in 32 bit integers. This is not possible with EW.

## The blockette type 1001 specifications

This blockette is 8 bytes long and has the following structure:

|   | Field name                                    | Type  | length (bytes) |
|---|---|-------|----------------|
| 1 | Blockette code                                | UWORD | 2              |
| 2 | Offset to the beginning of the next blockette | UWORD | 2              |
| 3 | Timing quality                                | UBYTE | 1              |
| 4 | Microseconds                                  | BYTE  | 1              |
| 5 | Reserved                                      | UBYTE | 1              |
| 6 | Frame count                                   | UBYTE | 1              |

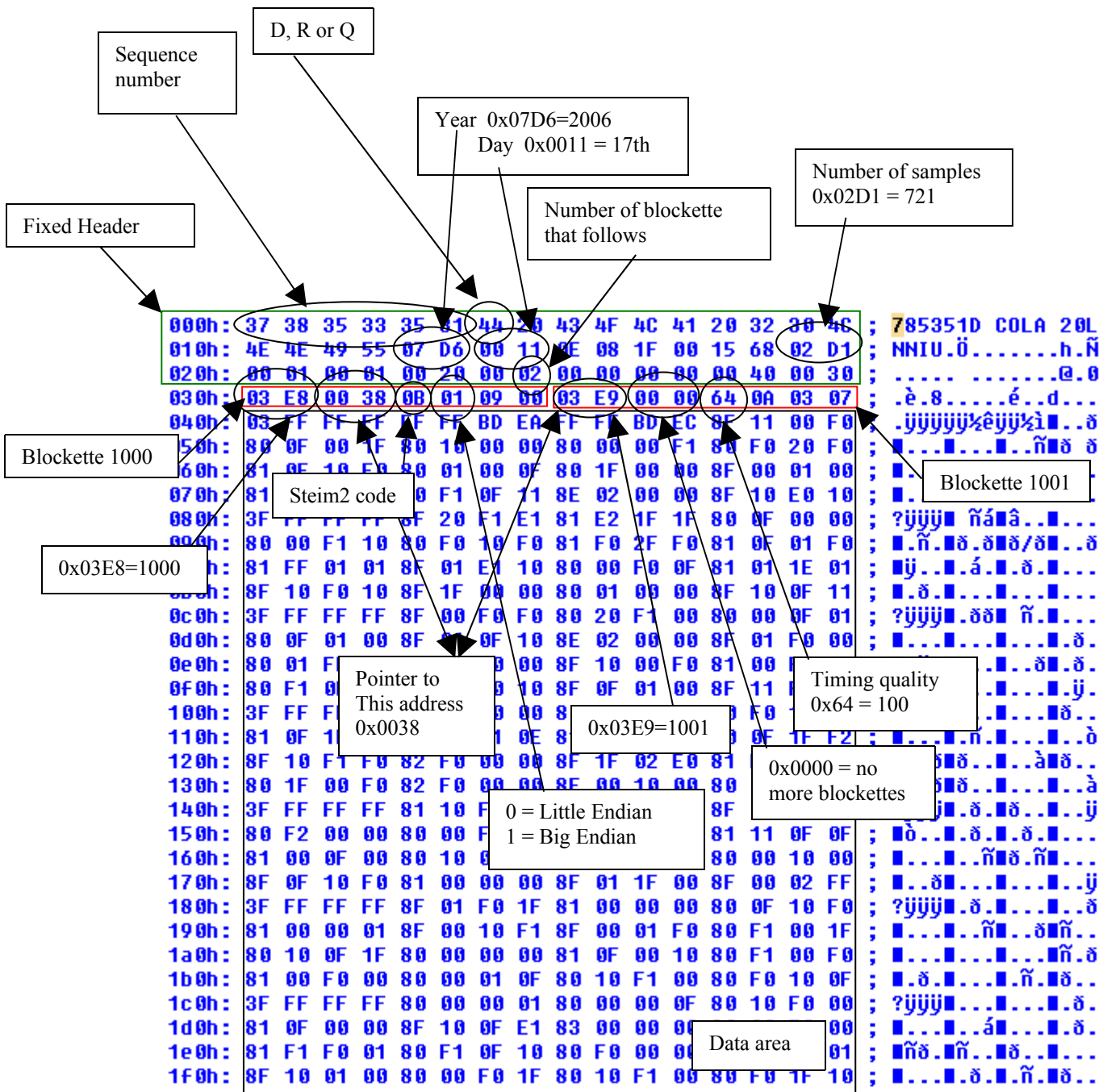
- 1) Blockette code that contains always the number: 1001
- 2) The offset to the beginning of the next blockette
- 3) Timing quality. Can be used by the digitizer manufacturer to estimate the time quality of this data packet from 0 to 100% of accuracy.
- 4) Precision of the start time down to microseconds. This field is present to improve the accuracy of the time stamping given by the fixed header time structure.
- 5) Reserved byte
- 6) Frame count. Is the number of 64 byte compressed data frames in the 4k record. (maximum of 63).

## Problems with the blockette type 1001 interpretation

- 1) The field of the blockette code has the famous big/little problem as well as the offset field.
- 2) The Reserved field is used in some stations for example *cola.iu.liss.org* server is transmitting a 0x03 number in this field. What is the meaning? It should be 0x00 isn't it? And if it is reserved, is reserved to whom and for what?
- 3) The frame count is not well explained. When you will look at the steim1 and steim2 compression description (that follows in this document) you will easily realize that this field is not needed. First of all if we don't have a compression there is no need to specify more pointers or length descriptors because there is already present a sample number field in the fixed header. Furthermore either the Steim1 and Steim2 code are self explanatory on how many samples are containing.  
Anyway according to what some LISS station issue on this field it have to be set to number 7 that are the frame that can be contained in a 448 bytes of data ( $7 \times 64 = 448$ ). Each SteimX frame is long 64 bytes.

### Schematic interpretation of the LISS miniSEED 512 bytes packet

The following picture represent an hex dump of the station of *cola.iu.liss.org*. The most important fields are specified and linked to a description box.





### The Steim1 encoding

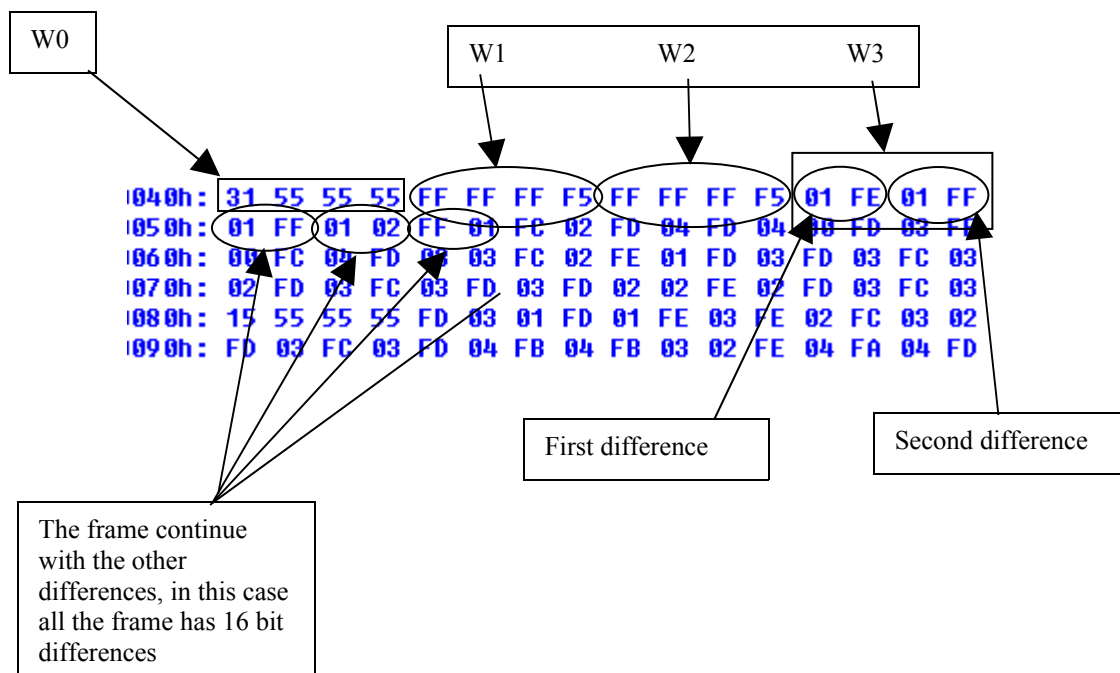
The Steim1 encoding is accomplished basically on the following rules:

- Each frame is composed of 64 bytes
- Each frame is divided in 16 words of 4 bytes each
- The first 4 bytes contain 16 nibbles each 2 bits long that specify the encoding mode of the next 15 words
- Each word can contain 4 samples (8 bit long), 2 samples (16 bit long), 1 sample (32 bit long)
- Each word can contain only differences coded in the same way
- In the nibble coding: 0b00=no data; 0b01=8 bit data, 0b10=16 bit data, 0b11=24 bit data

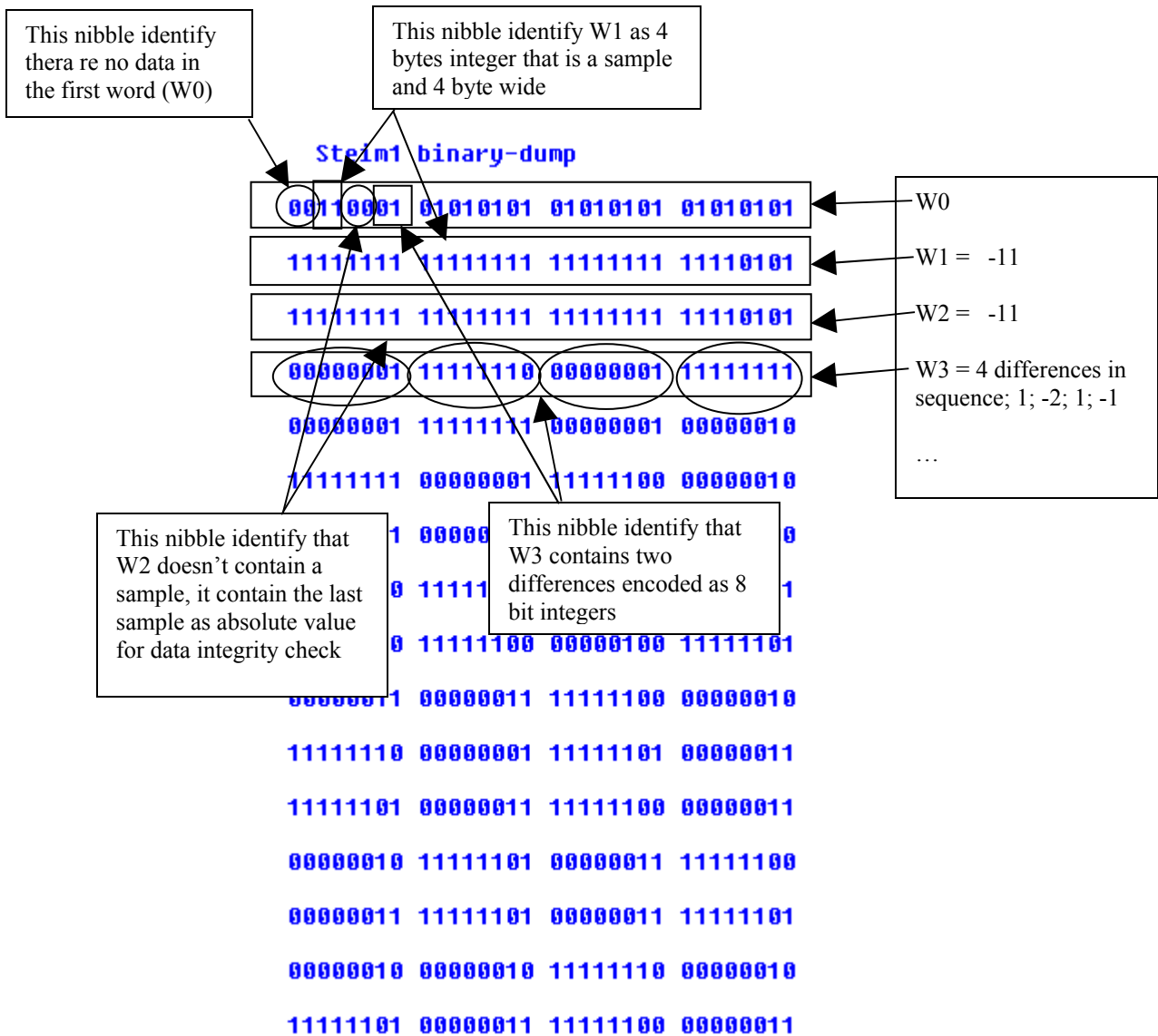
The following table shows the frame structure and the position of the nibble coding table. The binary nibble in the table are only as examples and vary according to the encoding signal.

|     | C0   | C1         | C2         | Cx         | C15         |
|-----|--|------------|------------|------------|-------------|
| W0  | 0b00<br>W0 itself  | 0b01<br>W1 | 0b10<br>W2 | 0bXX<br>Wx | 0b11<br>W15 |
| W1  | Four 8 bit samples: d0, d1, d2 and d3  |            |            |            |             |
| W2  | C1=01 = two 16 bit samples; d4 and d5 (in this example)  |            |            |            |             |
| Wx  | Wx code  |            |            |            |             |
| W15 | 1 sample. The 32 bit integers contain the last difference of the frame long 4 bytes because C15=0b11 |            |            |            |             |

The writing routine must keep in account that the very second long word of the frame contain the very first absolute sample and the third long word contain the last absolute sample of the frame. For both the code is 4 bytes long and follow or the big-endian rule or the little-endian rule.



The following scheme shows the same frame in binary.



**MSI Version 1.4 Output of the above mSEED file**

Detected record length of 512 bytes  
 NN\_STATI\_LL\_SHZ, 000003, D  
 start time: 2006,017,12:39:50.720000  
 number of samples: 413  
 sample rate factor: 50 (50 samples per second)  
 sample rate multiplier: 1  
 number of blockettes: 2  
 time correction: 0  
 data offset: 64  
 first blockette offset: 48  
 BLOCKETTE 1000: (Data Only SEED)  
 next blockette: 56  
 encoding: STEIM 1 Compression (val:10)  
 byte order: Big endian (val:1)  
 record length: 512 (val:9)  
 BLOCKETTE 1001: (Data Extension)  
 next blockette: 0  
 timing quality: 100%  
 micro second: 0  
 frame count: 7

decimal dump in decoded values:  
 -11 -10 -12 -11 -12 -11 -12 -11 -9 -10 -9 -13  
 -11 -14 -10 -13 -9 -9 -12 -9 -10 etc... etc.... Last value: -11

Problems with the Steim1 encoding algorithm experienced using the SEED Reference Manual (page 129-132)

- 1) Seems there are no reference on where the first absolute sample and the last check sample should be stored inside the frame. The position we explained before is based on conclusion of analysis of packet recorded from miniSEED LISS remote stations. The position seems to be logical and reasonably well made but not well documented
- 2) Data check integrity with the TCP/IP protocol as transport media seems to be an overkill considering the TCP/IP already perform all the needed data integrity check. A backward compatible method to avoid the storing of the integrity check value could be advisable
- 3) Results in coding Steim1 packet has been successfully achieved testing the packets with DumpSeed and with MSI, EarthWorm seems to do not accept the data integrity value, Seiscomp test are under progress until during the editing of this document

## The Steim2 encoding

The Steim2 encoding is accomplished basically on the following rules:

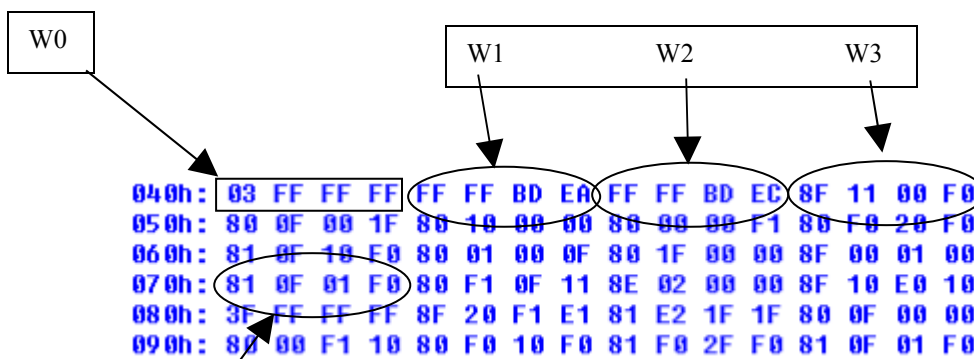
- Each frame is composed of 64 bytes
- Each frame is divided in 16 words of 4 bytes each
- The first 4 bytes contain 16 nibbles each 2 bits long that specify the encoding mode of the next 15 words
- Each word can contain 2 more encoding nibbles in the 30<sup>th</sup> and 31<sup>st</sup> bits
- The words can contain differences of variable length of 30, 15, 10, 8, 6, 5, 4 bits.
- Each word can contain only differences coded in the same way
- In the nibble coding: 0b00=no data; 0b01=8 bit data, 0b10=point to the additional encoding nibble and rules of type A applies to it, 0b11=point to the additional encoding nibble and rules of type B applies to it

Additional encoding nibble type A:  
 0b01=one 30 bit difference in W<sub>x</sub>  
 0b10=two 15 bit differences in W<sub>x</sub>  
 0b11=three 10 bit differences in W<sub>x</sub>

Additional encoding nibble type B:  
 0b01=one 6 bit difference in W<sub>x</sub>  
 0b10=two 5 bit differences in W<sub>x</sub>  
 0b11=three 4 bit differences in W<sub>x</sub>

The following table shows the frame structure and the position of the nibble coding table. The binary nibble in the table are only as examples and vary according to the encoding signal.

|                | C0                | C1          | C2         | C3          | C4         | C5          | C6         | C <sub>x</sub> | C15         |            |
|----------------|-------------------|-------------|------------|-------------|------------|-------------|------------|----------------|-------------|------------|
| W0             | 0b00<br>W0 itself | 0b01<br>W1  | 0b11<br>W2 | 0b11<br>W3  | 0b11<br>W4 | 0b10<br>W5  | 0b10<br>W6 | 0bXX           | 0b10<br>W15 |            |
| W1             | 0b01 (A)          | 30 bit (d0) |            |             |            |             |            |                |             |            |
| W2             | 0b10 (A)          | 15 bit (d0) |            |             |            | 15 bit (d1) |            |                |             |            |
| W3             | 0b11 (A)          | 10 bit (d0) |            | 10 bit (d1) |            | 10 bit (d2) |            |                |             |            |
| W4             | 8 bit (d0)        |             | 8 bit (d1) |             | 8 bit (d2) |             | 8 bit (d3) |                |             |            |
| W5             | 0b01 (B)          | 6 bit (d0)  |            | 6 bit (d1)  |            | 6 bit (d2)  |            | 6 bit (d3)     |             | 6 bit (d4) |
| W <sub>x</sub> | 0b10 (B)          | 7 bit (d0)  |            | 7 bit (d1)  |            | 7 bit (d2)  |            | 7 bit (d3)     |             | 7 bit (d4) |
| W15            | 0b11 (B)          | Not used    | 4 bit (d0) | 4 bit (d1)  | 4 bit (d2) | 4 bit (d3)  | 4 bit (d4) | 4 bit (d5)     | 4 bit (d6)  |            |



In this example W1 is the first absolute value encoded as integer 32 bit. W2 is the integrity check value and W3 is the first word containing data. Only for further explication, if you count you see that the 4<sup>th</sup> circle is related to W12. This encoding format is better viewed if a binary dump is examined; see the following paragraph.

Nibble indicating the first sample absolute word. This byte is obscure not understood if it should be 00 or what...

Nibble indicating the W2 is a non data (it is anyway a 32 bit word for data integrity check)

Nibble indicating the W0 itself, no data

**Steim2 binary-dump**

00000011 11111111 11111111 11111111

11111111 11111111 11111111 11110111

11111111 11111111 11111111 11110100

10000000 00100001 00101110 01011101

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10000000 00001111 11111110 00001111

10001110 00011101 11010000 000011

10001110 00001100 11101110 001010

10000000 00101110 00001111 111111

10000100 00011111 00110001 001000

10000011 11110010 01000000 000100

10000010 11100011 00000000 110000

W0  
W1 = - 5  
W2 = - 12  
W3 7 differences of 4 bit  
W4 7 differences of 4 bit  
...

First sample absolute word

Integrity check word

Encoding nibble of W3. It tell (being 0b11) to look at the additional encoding nibble and use the table B for its decoding. In the example the nibble 0b11 and the additional nibble 0b10 explain the word contains seven 4 bits differences.

The 32 bit words W3 exploded, to better view the division, is:

|      |                            |            |
|------|----------------------------|------------|
| 10   | additional encoding nibble |            |
| 0    | not used                   |            |
|      |                            | abs values |
| 0000 | = 0                        | -5 *       |
| 0010 | = +2                       | -3         |
| 0001 | = +1                       | -2         |
| 0010 | = +2                       | 0          |
| 1110 | = -2                       | -2         |
| 0101 | = +5                       | 3          |
| 1101 | = -3                       | 0          |

\* This value (the first difference) seems to be not used in the Steim2 with EW we didn't understood why and how. MSI doesn't ignore it.

Additional encoding nibble. Not readable if the data are packed in four 8 bit differences, in this case the nibble contain the bit 6<sup>th</sup> and 7<sup>th</sup> of the related byte.

### MSI Version 1.4 Output of the above mSEED file

Detected record length of 512 bytes

NN\_STATI\_LL\_SHZ, 000024, D

start time: 2006,017,15:37:35.240000

number of samples: 721

sample rate factor: 50 (50 samples per second)

sample rate multiplier: 1

number of blockettes: 2

time correction: 0

data offset: 64

first blockette offset: 48

BLOCKETTE 1000: (Data Only SEED)

next blockette: 56

encoding: STEIM 2 Compression (val:11)

byte order: Big endian (val:1)

record length: 512 (val:9)

BLOCKETTE 1001: (Data Extension)

next blockette: 0

timing quality: 100%

micro second: 0

frame count: 7

|    |    |    |     |     |     |     |     |        |        |                  |    |    |    |
|----|----|----|-----|-----|-----|-----|-----|--------|--------|------------------|----|----|----|
| -5 | -3 | -2 | 0   | -2  | 3   | 0   | 0   | 0      | -1     | -2               | -4 | -4 | -5 |
| -5 | -9 | -9 | -12 | -12 | -14 | -18 | -17 | etc... | etc... | last value = -12 |    |    |    |

### Problems using Steim2 format

- 1) The second and the third encoding nibble in W0 has no explication in any manual
- 2) The very first difference seems to be ignored by EW but not by MSI (not tested on DumpSEED); why it is skipped by EW? EW server send the very first difference (or the location where it should be) with a non sense value.