Binary Timeseries File Format Specification

Jonathan Schilling

November 19, 2019

1 Scope

This is the specification for a really simple binary file format for storing a regularly-spaced sequence of measurement data in an efficiently writeable and readable format. The basic assumption is that the time axis t_i of a series of N measurements can be computed on the fly from the array indices:

$$t_i = t_0 + i \cdot \Delta t \quad \text{for} \quad i = 0, 1, ..., (N-1) \quad , \tag{1}$$

where t_0 is the (reference) timestamp of the first sample and Δt is the sampling interval.

The data values y_i are stored as raw values $\hat{y}_i$, optionally with an offset o and a scaling factor s:

$$\begin{aligned} y_i &= \hat{y}_i \quad \text{for} \quad i = 0, 1, ..., (N-1) \quad \text{without scaling,} \\ y_i &= o + s \cdot \hat{y}_i \quad \text{for} \quad i = 0, 1, ..., (N-1) \quad \text{with scaling.} \end{aligned} \tag{2}$$

The number of samples is limited by the maximum value of the (signed 32-bit) `int` type, which is

$$2\,147\,483\,647 \approx 2.1 \cdot 10^9 \quad .$$

In case of raw `double` values, the corresponding maximum file size is $(64 + 8 \cdot 2\,147\,483\,647)$ bytes ≈ 16 GB, where 64 bytes are reserved for the file header information.

2 Definitions

In Tab. 1 you find an overview of the types for raw data considered in this specification.

type	size in bytes	identifier value	ok for time	ok for data
`byte`	1	0	no	yes
`short`	2	1	no	yes
`int`	4	2	no	yes
`long`	8	3	yes	yes
`float`	4	4	no	yes
`double`	8	5	yes	yes

Table 1: Data types considered relevant for time series data.

The first column lists the Java-style name of the given types. In the second column, the size of the types in bytes is listed. The third column lists the numeric value of the `dtype` bytes identifying the type of data in the file (see below). The last two columns tell you if a given type can be used to specify the time axis (ok for time) and the data values (ok for data).

Throughout this document, the datatypes refer to the signed version of these. Unsigned versions of the datatypes are not considered here, as they are not available in certain programming languages (e.g. Java).

offset	size	type	allowed values	description
0	2	`short`	1	used to verify correct endianess
2	1	`byte`	3 or 5	data type of time: 3 (long) or 5 (double)
3	8	`long` or `double`	*any*	t_0: reference timestamp
11	8	`long` or `double`	*any*	Δt: time interval between two samples
19	1	`byte`	0 ... 5 or -128	data type of scaling for data values highest bit set indicates scaling is disabled
[20]	8	*varying*	*any*	offset o of data values
[28]	8	*varying*	*any*	scaling factor s for data values
36	23	*reserved*	*reserved*	reserved for future use, e.g. units
59	1	`byte`	0 ... 5	dtype of raw data: 0 (byte) to 5 (double)
60	4	`int`	> 0	number N of data values
64	*varying*	*varying*	*any*	data values $\hat{y}_i$ of type given by dtype

Table 2: Structure of the binary timeseries files. The values in the columns "offset" and "size" are in bytes. The rows where the offset is in [ ] are not valid and should contains zeros in the file if no scaling for the data values is used.

3 File Structure

The contents of the files are structured as shown in Tab. 2.

The first field at offset 0 in the file is a `short`, which is always 1. It should be read using a readShort() or similar function, which implicitly assumes the system's endianess. Then it should be checked if the returned value is 1 or -128. In the latter case, the endianess of the reading method is wrong and needs to be flipped in order to proceed.

The next field at offset 2 defines the datatype of the time axis definition values t_0 and Δt. If it is equal to 3, the time definition is given as `long`. If it is equal to 5, the time definition is given as `double`. No assumption should be made on the unit of these values, although it is recommended to reserve `double` for seconds and `long` for nanoseconds.

The next field at offset 3 defines the reference timestamp t_0 and has to be read as a `long` or `double`, depending on the value read at offset 2. The next field at offset 11 defines the sampling interval Δt and has to be read as a `long` or `double`, depending on the value read at offset 2. The time axis definition values t_0 and Δt always have to be of the same datatype and should use the same unit.

The field at offset 19 defines the datatype of the scaling values to come. If the highest bit is set in the datatype of the scaling values, no scaling is used. If the highest bit is zero in the datatype of the scaling values, scaling is used. The scaling values o and s always have to be of the same datatype and should use the same unit. At offset 20, the constant offset o of the raw data is stored. Its size can range from one byte to at most eight bytes and it is always stored from offset 20 on; the remaining (unused) bytes in case of a type smaller than 8 bytes are ignored and should be written as zeros. Right after the data offset value, at an offset of 28, the scaling factor s is stored. Its size can range from one byte to at most eight bytes and it is always stored from offset 28 on; the remaining (unused) bytes in case of a type smaller than 8 bytes are ignored and should be written as zeros.

The bytes at offsets 36 to 58 are reserved for future use.

The next field at offset 59 defines the datatype of the raw data. Right after the raw data type, at an offset of 60, the number of data values N to come is stored. From offset 64 on, the raw data values $\hat{y}_i$ are stored, which have to be read as the data type defined by the value at offset 59.

4 Subset Reading

The main goal of this file format is to allow easy and fast reading of subsets of the whole time series data. Having an equally spaced time axis allows to compute the data indices inside a given time interval and using the definitions in Sec. 3, the offsets in the file can be computed for seeking to the computed position in the file and reading only from there on.

Suppose you have read t_0 and Δt from the binary timeseries file header and now want to read all available samples inside the interval $[t_l, t_u]$ where $t_l < t_u$ and the subscript l (u) stand for *lower* (*upper*). This situation is illustrated in Fig. 1 (SVG, EPS).

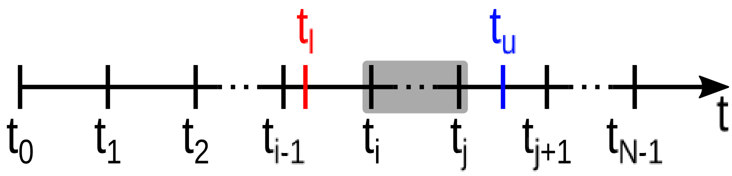

Figure 1: Time axis of a time series, where all data between t_l and t_u should be read. The resulting interval to be read includes indices from i to j and is indicated by the grey rectangle.

It is evident that the timestamps t_l and t_u are not necessarily aligned with the time axis of the available data. Therefore, rounding has to be used to compute the indices of data:

- `double` timestamps: $t_0, \Delta t \in \mathbb{R}$

$$\begin{aligned} i &= \left\lceil \frac{t_l - t_0}{\Delta t} \right\rceil \in \mathbb{N} \\ j &= \left\lfloor \frac{t_u - t_0}{\Delta t} \right\rfloor \in \mathbb{N} \end{aligned}$$

- `long` timestamps: $t_0, \Delta t \in \mathbb{N}$

$$\begin{aligned} i &= \frac{t_l - t_0 + \Delta t - 1}{\Delta t} \in \mathbb{N} \\ j &= \frac{t_u - t_0}{\Delta t} \in \mathbb{N} \end{aligned}$$

In case of the `long` timestamps, implicit computation of the floor of the division result is implied. Using these formulas, the relevant part of the file to be read starts at index i and ends at index j.

5 Finalizing Remarks

An implementation of this file format in Java and Python is available on GitHub:

https://github.com/jonathanschilling/BinaryTimeseries

and the Java version is also on Maven Central:

```
<dependency>
  <groupId>de.labathome</groupId>
  <artifactId>BinaryTimeseries</artifactId>
  <version>1.0.0</version>
</dependency>
```

In case of questions or bug reports, please contact the author via an eMail to

jonathan.schilling@mail.de