Unisens 2.0

How to create your own data files

Malte Kirst

21. Oktober 2010

Introduction

This document explains the usage of the Unisens data format by the help of some examples. The binary files in this examples are all rendered in hexadecimal notation and in little endian format. The Java code in the listings represent a minimal example for writing the data file. The header file (unisens.xml) will be not complete if it is only created by these minimal examples.

Tabelle 1: Data types specified by Unisens

data type	size (byte)	values margin
`double`	8	$4.9 \cdot 10^{-324} \ldots 1.7976931348623157 \cdot 10^{308}$
`float`	4	$1.4 \cdot 10^{-45} \ldots 3.4028235 \cdot 10^{38}$
`int32`	4	$-2147483648 \ldots 2147483647$
`int16`	2	$-32768 \ldots 32767$
`int8`	1	$-128 \ldots 127$
`uint32`	4	$0 \ldots 4294967295$
`uint16`	2	$0 \ldots 65535$
`uint8`	1	$0 \ldots 255$

Signals

All continuous sampled data with a fix sample rate and one or more channels are called signals. They are stored as a `signalEntry`.

We recommend using the binary format for signal entries. In this format, the samples are stored as little or big endian in the following order: sample1.channel1, sample1.channel2, ... sample1.channelN, sample2.channel1, sample2.channel2, ... sample2.channelN, ... The library can provide a random access to the data with the information about the data type, the number of channels and the sample number.

We use for all examples in this document the same signal with two channels (A und B) each with three 16 bit samples and a sampling rate of 250 Hz.

$$\text{signal} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}. \tag{1}$$

BIN

Many programming languages support the direct writing of binary files. Unisens supports different data types for binary files (see table 1) as well as little and big endian. The data type (`dataType`) and the endianess (`endianess`) have to be assigned in the header file.

```
01 00 04 00 02 00 05 00  03 00 06 00
```

Listing 1: Java code for the creation of a binary signal entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
SignalEntry se = u.createSignalEntry("signal.bin", new String[]{"A",
    "B"}, DataType.INT16, 250);
short[][] A = new short[][]{{1, 4}, {2, 5}, {3, 6}};
se.append(A);
u.save();
u.closeAll();
```

XML

Saving signal entries as an XML file is supported by the Unisens library, but we suggest this only in exceptional cases. The structure of the XML file is defined in its schema definition file signal.xsd.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<signal>
        <sample>
                <data>1</data>
                <data>4</data>
        </sample>
        <sample>
                <data>2</data>
                <data>5</data>
        </sample>
        <sample>
                <data>3</data>
                <data>6</data>
        </sample>
</signal>
```

Listing 2: Java code for the creation of an XML signal entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
SignalEntry se = u.createSignalEntry("signal.xml", new String[]{"A",
    "B"}, DataType.INT16, 250);
short[][] A = new short[][]{{1, 4}, {2, 5}, {3, 6}};
se.setFileFormat(se.createXmlFileFormat());
se.append(A);
u.save();
u.closeAll();
```

CSV

Sometimes it is useful to save the signals in a CSV file (manual editability of the data, little amount of data or due to compatibility reasons to other software). In this case, the first channel's data is written in the first column, the second channel in the second column respectively. The column separator (`separator`) and the decimal separator (`deciamlSeparator`) are given in the header file.

```
1;4
2;5
3;6
```

Listing 3: Java code for the creation of a CSV signal entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
SignalEntry se = u.createSignalEntry("signal.csv", new String[]{"A",
    "B"}, DataType.INT16, 250);
short[][] A = new short[][]{{1, 4}, {2, 5}, {3, 6}};
CsvFileFormat cff = se.createCsvFileFormat();
cff.setSeparator(";");
cff.setDecimalSeparator(".");
se.setFileFormat(cff);
se.append(A);
u.save();
u.closeAll();
```

Values

All measurement data that is not sampled with a constant sample rate but at different, flexible points of time should be saved as valuesEntry. Every valuesEntry consists of a time stamp and one or more values at this time. The number of measured values at each time is the number of channels.

The time stamp is always an integer containing the sample number in the sample rate given for this entry. By this method the time stamps can be adjusted to any accuracy. For example, for a millisecond-exact time stamp you use a sample rate of 1000 Hz. The following examples use the two channel measurement values from (1) at the time 1320, 22968 and 30232.

CSV

The CSV file format is the best choice for most measurement values, because the format's clear structure. The first column is for the time stamps (`sampleStamps`), the following columns are for the data – one column for each channel. The column separator (`separator`) and the decimal separator (`deciamlSeparator`) are given in the header file. For providing a fast data access, the file has to be sorted ascending by the time stamp.

```
1320;1;4
22968;2;5
30232;3;6
```

Listing 4: Java code for the creation of a CSV values entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
ValuesEntry ve = u.createValuesEntry("values.csv", new String[]{"A",
    "B"}, DataType.INT16, 1000);
CsvFileFormat cff = ve.createCsvFileFormat();
cff.setSeparator(";");
cff.setDecimalSeparator(".");
ve.setFileFormat(cff);
ve.append(new Value(1320, new short[]{1, 4}));
ve.append(new Value(22968, new short[]{2, 5}));
ve.append(new Value(30232, new short[]{3, 6}));
u.save();
u.closeAll();
```

BIN

The storage of a huge amount of values, e. g. for a tachogram of a long ECG data set, can be more efficient in a binary file. In this case, the data is stored in the following order: sample stamp as INT64 followed by the corresponding values, channel by channel, in the given format. The next combinations of sample stamps are following subsequently. In our example we have two values of the type INT16 after each sample stamp.

The biggest possible sample stamp is $2^{63} - 1$. The data type (`dataType`) and the endianess (`endianess`) have to be assigned in the header file.

```
28 05 00 00 00 00 00 00   01 00 04 00 B8 59 00 00
00 00 00 00 02 00 05 00   18 76 00 00 00 00 00 00
03 00 06 00
```

Listing 5: Java code for the creation of a binary values entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
ValuesEntry ve = u.createValuesEntry("values.bin", new String[]{"A",
    "B"}, DataType.INT16, 1000);
ve.setFileFormat(ve.createBinFileFormat());
ve.append(new Value(1320, new short[]{1, 4}));
ve.append(new Value(22968, new short[]{2, 5}));
ve.append(new Value(30232, new short[]{3, 6}));
u.save();
u.closeAll();
```

XML

Values can also be stored as an XML file according to its XML schema values.xsd.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<values>
   <value sampleStamp="1320">
      <data>1</data>
      <data>4</data>
   </value>
```

```
   <value sampleStamp="22968">
      <data>2</data>
      <data>5</data>
   </value>
   <value sampleStamp="30232">
      <data>3</data>
      <data>6</data>
   </value>
</values>
```

Listing 6: Java code for the creation of an XML values entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
ValuesEntry ve = u.createValuesEntry("values.xml", new String[]{"A",
     "B"}, DataType.INT16, 1000);
ve.setFileFormat(ve.createXmlFileFormat());
ve.append(new Value(1320, new short[]{1, 4}));
ve.append(new Value(22968, new short[]{2, 5}));
ve.append(new Value(30232, new short[]{3, 6}));
u.save();
u.closeAll();
```

Annotations

Annotations should be stored as `eventEntry`, because this entry class has no interpretation as physical values. An event entry stores the point of time of the event (according the same procedure of the values), an annotation code of the event and a comment for each event. We are using the sample stamps 124, 346 and 523 in this example, at a time base of 250 Hz. The annotation corresponds to the ECG annotations of the WFDB Annotation Codes.

CSV

The CSV format is the best choice for most event entries. Because of its structure is no declaration of the length of the annotation or the comment necessary. Only the column separator (`separator`) has to be assigned in the header file. The first column of the CSV file contains the sample stamp, the second column the corresponding annotation code and the third column the comment, respectively. For many annotations is the third column optional. For a fast data access, the CSV file as to be sorted ascending by time.

```
124;N;NORMAL
346;N;NORMAL
523;V;PVC
```

Listing 7: Java code for the creation of a CSV event entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
EventEntry ee = u.createEventEntry("event.csv", 250);
CsvFileFormat cff = ee.createCsvFileFormat();
cff.setSeparator(";");
```

```
cff.setDecimalSeparator(".");
ee.setFileFormat(cff);
ee.append(new Event(124, "N", "NORMAL"));
ee.append(new Event(346, "N", "NORMAL"));
ee.append(new Event(523, "V", "PVC"));
u.save();
u.closeAll();
```

XML

Events can also be stored as an XML file according to its XML schema events.xsd.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<events>
   <event comment="NORMAL" sampleStamp="124" type="N"/>
   <event comment="NORMAL" sampleStamp="346" type="N"/>
   <event comment="PVC" sampleStamp="523" type="V"/>
</events>
```

Listing 8: Java code for the creation of a XML-EventEntry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
EventEntry ee = u.createEventEntry("event.xml", 250);
ee.setFileFormat(ee.createXmlFileFormat());
ee.append(new Event(124, "N", "NORMAL"));
ee.append(new Event(346, "N", "NORMAL"));
ee.append(new Event(523, "V", "PVC"));
u.save();
u.closeAll();
```

BIN

The storage of events as a binary file is possible, but only useful in some scenarios. Similar to the values entry, the sample stamp is stored as a 64 bit integer, followed by the annotation code and the comment. Because of the binary format, the length of the annotation code and the comment has to be set to a fix length. We use the length 1 for the annotation code (`typeLength`) and 6 for the length of the comment (`commentLength`). Please note that all annotation codes and all comments have to be of the same length.

```
7C 00 00 00 00 00 00 00  4E 4E 4F 52 4D 41 4C 5A
01 00 00 00 00 00 00 4E  4E 4F 52 4D 41 4C 0B 02
00 00 00 00 00 00 56 50  56 43 20 20 20
```

Listing 9: Java code for the creation of a binary event entry

```
UnisensFactory uf = UnisensFactoryBuilder.createFactory();
Unisens u = uf.createUnisens("C:\\TestData");
EventEntry ee = u.createEventEntry("event.bin", 250);
ee.setFileFormat(ee.createBinFileFormat());
ee.setCommentLength(6);
ee.setTypeLength(1);
ee.append(new Event(124, "N", "NORMAL"));
```

```
ee.append(new Event(346, "N", "NORMAL"));
ee.append(new Event(523, "V", "PVC   "));
u.save();
u.closeAll();
```

Header File

After running all samples above, the Java library creates automaticly the following header file C:\TestData\unisens.xml. The order of the entries can vary. Please note that this unisens.xml file is not valid according the XML schema definition. Required attributes like timestampStart, lsbValue or contentClass may be missing.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<unisens version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.unisens.org/unisens2.0
    http://www.unisens.org/unisens2.0/unisens.xsd" xmlns="http://www.
    unisens.org/unisens2.0">
  <signalEntry dataType="int16" id="signal.bin" lsbValue="1"
      sampleRate="250">
    <binFileFormat endianess="LITTLE"/>
    <channel name="A"/>
    <channel name="B"/>
  </signalEntry>
  <signalEntry dataType="int16" id="signal.csv" lsbValue="1"
      sampleRate="250">
    <csvFileFormat decimalSeparator="," separator=";"/>
    <channel name="A"/>
    <channel name="B"/>
  </signalEntry>
  <signalEntry dataType="int16" id="signal.xml" lsbValue="1"
      sampleRate="250">
    <xmlFileFormat/>
    <channel name="A"/>
    <channel name="B"/>
  </signalEntry>
  <valuesEntry dataType="int16" id="values.bin" lsbValue="1"
      sampleRate="1000">
    <binFileFormat endianess="LITTLE"/>
    <channel name="A"/>
    <channel name="B"/>
  </valuesEntry>
  <valuesEntry dataType="int16" id="values.csv" lsbValue="1"
      sampleRate="1000">
    <csvFileFormat decimalSeparator="." separator=";"/>
    <channel name="A"/>
    <channel name="B"/>
  </valuesEntry>
  <valuesEntry dataType="int16" id="values.xml" lsbValue="1"
      sampleRate="1000">
    <xmlFileFormat/>
    <channel name="A"/>
    <channel name="B"/>
```

```
  </valuesEntry>
  <eventEntry commentLength="6" id="event.bin" sampleRate="250"
      typeLength="1">
    <binFileFormat endianess="LITTLE"/>
  </eventEntry>
  <eventEntry id="event.csv" sampleRate="250">
    <csvFileFormat decimalSeparator="." separator=";"/>
  </eventEntry>
  <eventEntry id="event.xml" sampleRate="250">
    <xmlFileFormat/>
  </eventEntry>
</unisens>
```