Binary I/O Format

2015-01-22

Author(s): H. Bräuning
Contributions:

Keywords: Data storage; BDIO, Tagged File Format

Applications using the BDIO library write and read data in binary files. Each file consists of one or more blocks of data. Blocks may be nested in other blocks. Each block is identified by a tag and must contain the total size of the block. Because the blocks are tagged, it is also called a tagged data format (TDF). The files typically have the extension '.tdf'.

Basic Structure

The TDF file starts with a 4 byte magic string. The 4 bytes represent the ASCII string 'TDF1'. It is always written as a string, so that the order of the 4 bytes is the same independent on the endianess of the system.

The 4 byte magic string is immediately followed by a header block. The header block contains the name of the application which wrote the file and the system time when the file was created in milliseconds. As applications in general define their own block structure, the application name is absolutely required.

The header block may be followed by one or more blocks.

Blocks

As explained above, a TDF file consists of a series of blocks. Data blocks can have an arbitrary length. Therefore the length of the block is specified within the block itself. Each data block starts with the 4 byte tag value. The tag is immediately followed by a 8 byte (64bit) field given the size of the data block in bytes. The size of the data block always includes the 4 bytes for the tag and the 8 bytes of the size field.

The basic block structure is as follows:

tag (4bytes)	
size (8bytes)	- n data bytes + 8 bytes size + 4 bytes tag
data (n bytes)	

Tags

Tags are 4 byte integer values. However, the upper 2 bytes are unused so that tags are limited to 16 bit values. They can be considered as 2 byte unsigned short values stored as 4 byte integer. This makes is easier to handle them for example in Java which does not have unsigned integer values.
System tags are indicated by setting bit 31 (0x8000). Their values range from 0x8000 to 0xFFFF. User tags can have values from 0x0000 to 0x7FFF. They are application specific and each application may define their own tags. This also means that the same user tag number may have different meanings in different applications.

The following system tags are predefined:

0xFFFF	– general header block
0xFFFE	– container block
0xFFFD	– beam information block
0xFFFC	– table block

Predefined Blocks

general header block

The general header block is present in each TDF file and is always the first block in the file. It contains general data and identifies the application which has written the file.

tag (4bytes)	– 0xFFFF
size (8bytes)	– 84
application name (64bytes)	– ascii string (0 terminated if less then 64 characters)
time stamp (8 bytes)	– system time in milliseconds

Note: the general header block does not contain the total file size, because this is often not known when the block is written. It is also not required. Reading applications should read blocks (whose size is defined) until end-of-file or may obtain the files size from the operating system.

container block

A container block is a data block which contains other data blocks. It is a means to structure the data within the file. Its usage is recommended but not required

tag (4bytes)	– 0xFFFE
size (8bytes)	– 12 + sum of all contained data block sizes
contained data blocks (arb. bytes)	

beam information block

The beam information block contains the information about the beam (if any) for which the data was taken. This block contains only information which is available on the frontend and applicable to all data. Specific information like amplifier settings etc. must be contained in user specific blocks.

tag (4bytes)	– 0xFFFD
size (8bytes)	– 52 bytes
cycle name (32bytes)	– cycle name, i.e. SIS.USER.VACC_01
cycle stamp (8bytes)	– start time of cycle in nanoseconds UTC

table block

The table block contains single valued data in form of a table together with a key and a unit. Each value is represented as a row. The length of a row is fixed. The rows have the following structure:
key (48 bytes)
value (8 bytes, double)
unitId (4 bytes, int)
unit (16 bytes)

The key string is used to identify the data value. The unit string gives the physical unit of the value and may be empty. The unitId value is intended for computational processing of the unit. Currently the following ids are in use:

```
<item access="RW" symbol="A" value="0"/>
<item access="RW" symbol="cd" value="1"/>
<item access="RW" symbol="K" value="2"/>
<item access="RW" symbol="kg" value="3"/>
<item access="RW" symbol="m" value="4"/>
<item access="RW" symbol="mol" value="5"/>
<item access="RW" symbol="rad" value="6"/>
<item access="RW" symbol="s" value="7"/>
<item access="RW" symbol="V" value="8"/>
<item access="RW" symbol="m_s" value="9"/>
<item access="RW" symbol="Hz" value="10"/>
<item access="RW" symbol="NBCharges" value="11"/>
<item access="RW" symbol="A_s" value="12"/>
<item access="RW" symbol="C" value="90"/>
<item access="RW" symbol="Particles" value="91"/>
<item access="RW" symbol="Counts" value="92"/>
<item access="RW" symbol="ADC_value" value="93"/>
<item access="RW" symbol="arb_units" value="99"/>
```

The total length of a single row is thus 76 bytes. The table block does not store the number of rows as it can easily be calculated from the block size and the fixed row length.

tag (4bytes)	– 0xFFFC
size (8bytes)	– n rows *76 bytes + 12 bytes
row 1 (76 bytes)	– first row
. . .	
row n (76 bytes)	– n-th row

Endianess

For the FESA class it is important that data is written as fast as possible. Therefore the data will be written in the endianess of the host system running the FESA class. Currently it s forseen that all host systems are Intel based and the data is therefore written in Little Endian. Should this change in the future, a backward compatible way to determine the endianess of the TDF file will be introduced.