HBr-2012-003

Binary I/O Format

2012-04-24

Author(s): H. Bräuning
Contributions:

Keywords: Data storage; FESA

Quite often FESA classes in the beam diagnostic group must write data to disk. While ASCII data may be preferred by those customers, who analyse it with their own tools, it has severe disadvantages when being written on a diskless frontend computer by a FESA class. Compared to direct binary output, ASCII data

- requires at least 2 - 3 times more storage space
  - higher network load
  - using a truly standardized xml format instead of tables incurs an even higher costs
- is cost intensive in terms of CPU time due to the conversion
- may lose precision for floating point numbers

From a performance point of view, writing in binary is the best choice. Therefore all FESA classes which must write data to disk should do so in binary.

It has been noted, that there is no standard format for binary I/O. The same however holds true for ASCII data with the exception of xml. Only xml provides a standardized, flexible (and even verifiable) way of presenting ASCII data. However, xml is not designed and not suited for measurement data, is more cumbersome to handle and very costly in terms of storage space and CPU time.

To facilitate the usage of binary data files, we propose here a flexible, extendable standard to store measurement data in binary form. The basic principle of the proposed format is to present the data in the form of data blocks and tag each data block. The tag specifies the meaning of the data in the block. The length of a data block is arbitrary. The resulting Tagged Data Format (TDF) file consists of a sequence of an arbitrary number of data blocks. To uniquely identify TDF files, the sequence of data block is preceded by a magic number.

Data block can be divided into system blocks and user block. Similarly, tags are divided into system tags and user tags. System block and tags have a predefined structure and value, which is the same for each TDF file. User blocks and tags however can be freely defined by the application.

Rules

magic number
The magic number is a 4 byte value representing the ASCII string TDF1

tags
Tags are 2 byte unsigned short values. System tags are indicated by setting the highest bit (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

blocks
Data block can have an arbitrary length. Therefore the length of the block is specified within the block itself. Each data block starts with the 2 byte tag value. The tag is immediately followed by a 4 byte (32bit) field given the size of the data block in bytes. The size of the data block includes the 2 bytes for the tag and the 4 bytes of the size field. The 32bit of the size field limit the size of a block to 4GByte. However, it is unlikely that larger data blocks are required. If more than 4GByte of data are to be stored, applications may easily break the data into blocks of smaller size.

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 (2bytes)	– 0xFFFF
size (4bytes)	– 70
application name (64bytes)	– ascii string (0 terminated if less then 64 characters)
time stamp (8 bytes)	– unix time stamp (or other?)

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 (2bytes)	– 0xFFFE
size (4bytes)	– 6 + 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 (2bytes)	– 0xFFFD
size (4bytes)	– 46 bytes
cycle name (32bytes)	– cycle name, i.e. SIS.USER.VACC_01
cycle stamp (8bytes)	– start time of cycle in nanoseconds UTC

endianess

Given that most frontend computers are currently Intel based, it is proposed that the data representation is fixed to Little Endian.

Note: It is also possible to leave the endianess free, i.e. data is written in the native endianess of the frontend computer. This requires that reading applications determine the endianess of the data file automatically. In principle this can be done by analysis the representation of the magic number. However, it introduces a further element of complication.

Libraries

To make dealing with the proposed binary format easier, libraries for C and Java will be provided. These libraries will provide helper functions to write to and read from TDF files, predefined structures and constants for system blocks and tags.