JEventViewer 2.0
User's Guide

Carl Timmer

Jefferson Lab Experimental Physics Software
and Computing Infrastructure group

26-May-2021

12000 Jefferson Ave
Newport News, VA 23606
Phone 757.269.7100

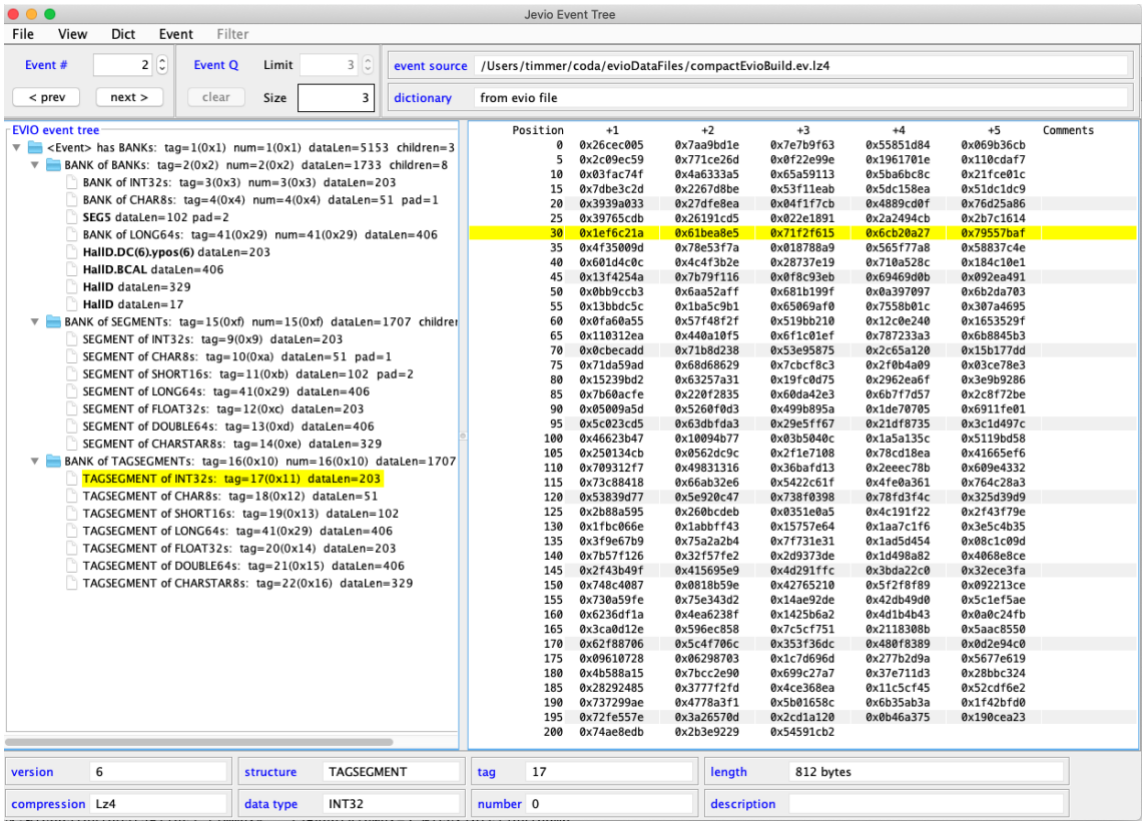
Table of Contents

Chapter 1

1. Evio Event Viewing

This manual describes a graphical user interface for looking at EVIO format files event-by-event, although it can also look at any file as a list of 32 bit integer (words).

Figure 1.1: Event-viewing gui



This version of the JEventViewer is compatible with all evio formats, including the latest, evio 6.0. To run it, using Java 8 or later, simply execute:

```
java org.jlab.coda.eventViewer.EventTreeFrame
```

Make sure that the jar files, JEventViewer-2.0.jar and all the other jars in java/jars directory, are your CLASSPATH environment variable. The alternative to that is executing the provided script:

```
scripts/jeviodump
```

Note that the script is for CODA users and sets the classpath to:

```
$CODA/common/jar
```

In other words, make sure that your environmental variable CODA is defined and all the jar files in java/jars are in that directory as well.

1.1 Installation

The code can be downloaded from its github site:

git clone https://github.com/JeffersonLab/JEventViewer.git

The default branch is "2.0" but one can insure that by calling:

```
git checkout 2.0
```

There's the jar file **JEventViewer-2.0.jar** in the **java/jars/java8** directory, already pre-built with Java 8, so one does not need to build it. There's another one in **java/jars/java15** directory built with Java 15.

However, to build it simply do:

```
ant jar
```

Other options can be seen by calling:

```
ant help
```

The output of this command is:

```
help:
     [echo] Usage: ant [ant options] <target1> [target2 | target3 | ...]

     [echo]        targets:
     [echo]        help       - print out usage
     [echo]        env        - print out build file variables' values
     [echo]        compile    - compile java files
     [echo]        clean      - remove class files
     [echo]        cleanall   - remove all generated files
     [echo]        jar        - compile and create jar file
     [echo]        install    - create jar file and install into 'prefix'
     [echo]                     if given on command line by -Dprefix=dir',
     [echo]                     else install into CODA if defined
     [echo]        uninstall  - remove jar file previously installed into 'prefix'
     [echo]                     if given on command line by -Dprefix=dir',
     [echo]                     else installed into CODA if defined
     [echo]        all        - clean, compile and create jar file
     [echo]        javadoc    - create javadoc documentation
     [echo]        developdoc - create javadoc documentation for developer
```

```
[echo]        undoc      - remove all javadoc documentation
[echo]        prepare    - create necessary directories
```

Although this is fairly self-explanatory, executing ant is the same as ant compile.
That will compile all the java. All compiled code is placed in the generated ./build directory.
If the user wants a jar file, execute ant jar to place the resulting file in the ./build/lib directory.
The java command in the user's path will be the one used to do the compilation.

1.2 Prerequisites

The other jar files necessary to compile JEventViewer-2.0.jar are in the java/jars directory.
They are compiled with Java 8. In addition, there are 2 subdirectories:

```
1) java/jars/java8, which contains all such jars compiled with Java 8, and
2) java/jars/java15 which contains all jars compiled with Java 15.
```

If a jar file is not available in Java 15 use the Java 8 version.

To generate these jar files, go to their respective github sites and follow the directions there:

https://github.com/JeffersonLab/disruptor

https://github.com/lz4/lz4-java

https://github.com/JeffersonLab/evio

https://github.com/JeffersonLab/et

https://github.com/JeffersonLab/cMsg

1.3 Documentation

Basically, you are now reading the only user documentation in either a pdf or word doc. In the repository, it's located in the **doc/users_guide** directory. There is javadoc that can be generated (ant javadoc or ant developdoc) but would only be useful to a developer or one trying to modify the source code.

1.4 Features

Here's a quick list of the main features:

- Valid event sources are files, cMsg messages, and ET buffers
- Fast compare ability for data from different events
- When receiving events through cMsg or ET, they can be filtered based on their CODA event type (physics, control, etc.) and trigger type if physics event

- View integer data as hex or decimal
- Select dictionary from event source or from separate file containing dictionary
- View the dictionary being used
- Export any evio file in xml format
- View the contents of any file as 32 bit hex integers
- Search for values, positions, evio records/blocks, evio events, or evio errors

In the figure above, starting with the middle of the gui first, the left side shows a tree structure diagram of the whole, single evio event being viewed. Notice that the type of each evio structure is given (bank, segment, tagsegment), along with the type of data it contains, tag, num, size, and # of children. Tag and num are shown in decimal and hex. If a dictionary is being used, the dictionary name is displayed instead of the corresponding structure type, data type, tag, and num values.

The right side, on the other hand, shows the data of any selected bank, segment, or tagsegment that contains a data type and not another container type. Integers can be displayed in hex or decimal.

A fast compare feature is able to compare data from different events. If the current event is changed while viewing the data of its selected structure, and if the new event has a structure with the same hierarchy of tags that the previous selection had, it too is automatically selected. This facilitates comparing the same structure in each successive event by simply hitting the “next” event button.

A dictionary can be loaded from a separate xml format file, or it can come embedded in an evio format file or buffer (cMsg, ET). The viewer allows the user to switch, in the “Dict” menu, between the different dictionaries if more than one is available. Any dictionary being used can be displayed instead of the data.

Selecting an ET system or a cMsg server as an event source, in the “Event” menu, brings up other menus to allow the proper connections to be created and maintained. The only assumptions made are that in a cMsg message, the evio data is contained in the byteArray field. Any dictionary is first looked for in the evio data and if none is found, it is looked for in a String payload item called “dictionary”.

The box in the upper left (under the row of menu buttons), “Event #”, shows the event currently selected (in this case 2) and allows the user to navigate to the desired event.

The box to its right, “Event Q”, shows different things depending on if the data source is a file, cMsg message, or ET event. For files, it shows the total number of events (in this case 3). For cMsg messages and ET events, on the other hand, events are continually arriving. In this case, “Size” shows the number of events currently in an internal queue. “Limit” allows the user to set the size of this internal queue, while “Clear” will remove all events currently in the queue. Once this queue is full, nothing else is added. The “Event #” controls can be used to switch between events in the queue.

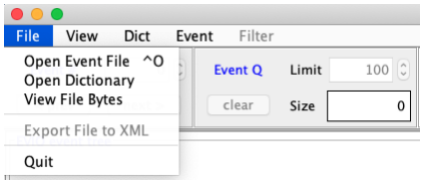
Switching between the different event sources can be done in the “Event” menu item. When selecting a cMsg or ET source, the “Filter” menu is enabled. With this menu, the user can choose to look at control, partially-built physics, physics events, or any combination as well as the selecting the run type of interest.

Notice that above the data, there are boxes containing the event and dictionary sources. Beneath the data are boxes containing information about the selected data structure such as its structure type, data type, tag, num, length in bytes, description, evio version, and the type of data compression if any.

Warning about performance: for large files, make sure they are local to the machine that’s running this program since it uses memory mapping to look at file data. You do not want the performance hit you’ll take for viewing files which are served over the network!

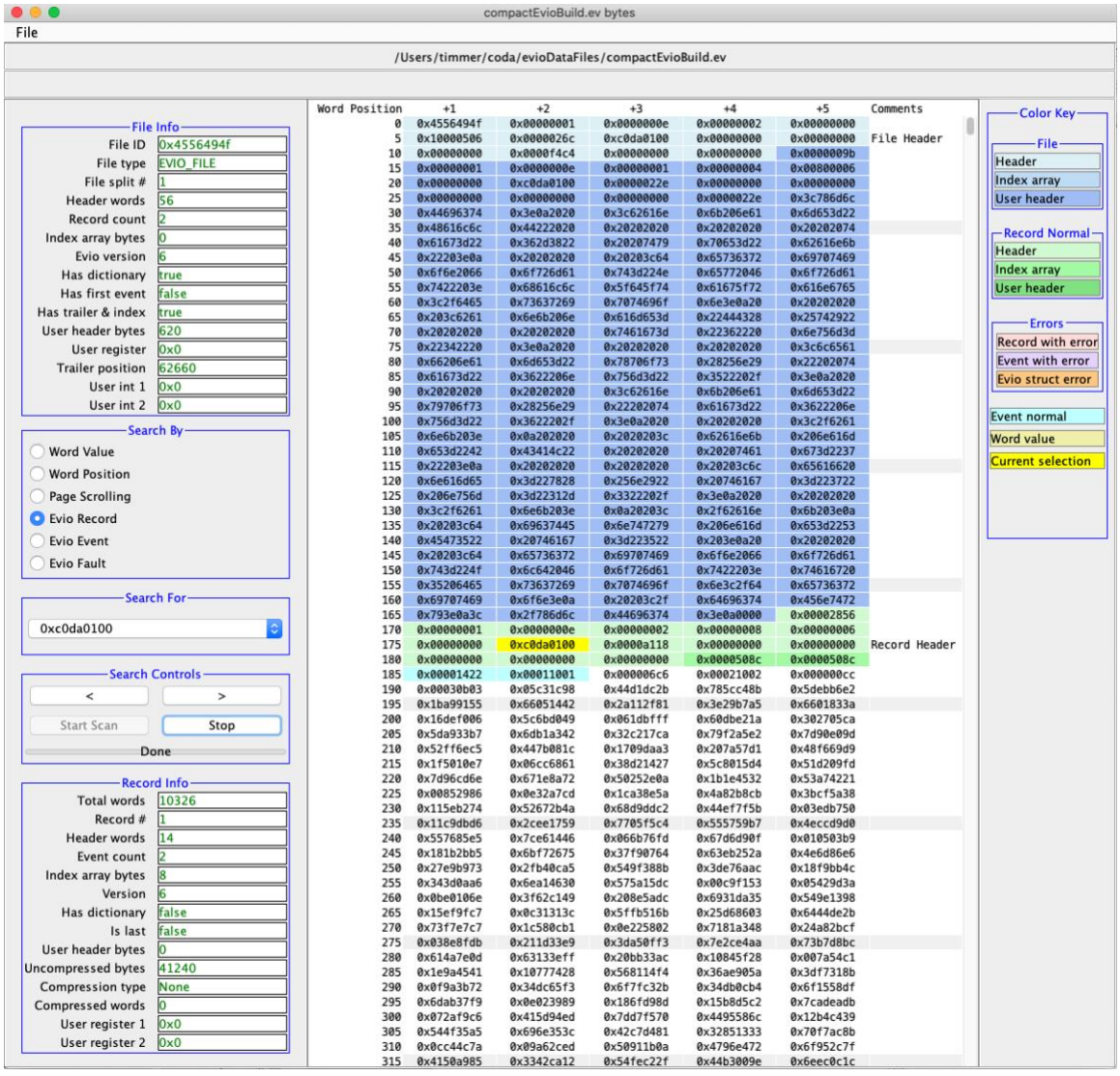
Chapter 2

2. File Data Viewing



The following figure is a screen shot of a file's data obtained by selecting the "View File Bytes" option of the "File" menu of the initial screen shown previously.

Figure 2.1: Data-viewing gui



There are occasions when one wants to examine the raw bytes in a file. This tool will allow one to do just that. It is capable of viewing any file's data, although it's designed specifically to look at evio versions 4 and 6 format data.

Each cell of the table contains 32 bits worth of data displayed in hex. Data can be switched between big and little endian under the "File" menu. The table contains up to 1GB worth of data at one time. For larger files, the next or previous 1GB are loaded when required while scanning through it. On the immediate right of the data is a slider which indicates where the current view is in relation to the part of the file that is currently memory mapped (up to 1GB). On the far right is a color key to highlighted cells.

The figure above is showing an evio 6 format file. All such files have a file header shown in blue. The light blue is the main header of 14 words. Although there is no index in this case, there is a dictionary which is stored in the file header's so-called user header. This is seen highlighted in dark blue. In the "File Info" box on the top left, all values in the file header appear in a table.

When searching for record headers, each one shows up highlighted in green. The light green is the main header of 14 words. The mandatory index of events shows up in medium green. Although not seen above, since it isn't used in evio, any associated user header is shown in dark green. When a record header is found, it's data is shown in the "Record Info" box on the left.

When searching for events, the first 2 words of each are highlighted in cyan. When an event is found, it's data is shown in the "Event Info" box on the left (not seen in the figure above).

2.1 Searching

In order to facilitate finding the data of interest, there are a number of different ways to hunt through it. The control panel on the left has "Search By" radio buttons allowing one to select whether to search by:

1. Looking for a given value
2. Jumping to a given position in the file
3. Scrolling page by page or by blocks of 40 pages
4. Jumping from one evio record/block header to the next
5. Jumping from one evio event to the next
6. Scanning the whole file for evio faults or errors

2.1.1 By Value

Look for a given value by selecting the "Word Value" radio button, typing the value into the "Search For" widget, and then hit the forward or backward search button under "Search Controls". The "Stop" button will be activated since searching a large file (say 20GB) may take extended time. If a search is stopped,

the view position stays where it was when the search was started. If stopped, starting another search starts from the same location. A progress bar is there to estimate how much of the file has been searched.

When a value is found, it is highlighted in gold. Hit the search button again to find the next or previous value. Highlights can be cleared under the “File” menu.

2.1.2 By Location

Look at a given location in the file by selecting the “Word Position” button, typing the position into the “Search For” widget, and then hitting the “Go” button. The view jumps to the given location and the value is selected (but not highlighted). The first position starts at 1, not 0. You can read the position from the table by taking the number in the far left column and adding the number of the heading at the very top of the column.

2.1.3 By Page

The “Page Scrolling” button activates the “<” and “>” buttons which hop through the file page (or view) by page. It also actives the “<<” and “>>” buttons immediately underneath which move through the file in 40 pages at a click.

2.1.4 By Evio Record/Block Header

For evio version 4 files: look for an evio format block header by selecting the “Evio Block” button. The program first looks for the magic # (0xc0da0100) of an evio block header. If found, it checks that the header length is 8 words. If so, it highlights all 8 words in green. All the information contained in that header is also displayed on the left in a panel called “Block Info.

For evio version 6 files: look for an evio format block header by selecting the “Evio Record” button. The program first looks for the magic # (0xc0da0100) of an evio record header. If found, it checks that the header length is 14 words. If so, it highlights all 14 words in light green. It highlights the index part of the header in medium green, and the user header part in the darkest green. All the information contained in that header is also displayed on the left in a panel called “Record Info” which can be seen in the figure above.

2.1.5 By Evio Event

Look for an evio event (top level evio bank) by selecting the “Evio Event” button. This is less straightforward than looking for record/block headers since there is no universal signature to look for. There are two ways to do the search. The first way is start the search immediately upon loading the file’s data or to first select a position before any events. Then hit the forward button. It is smart enough to hop over any file/record/block headers encountered and uses the length found in the event’s header to be able to find the next one when the forward button is clicked again. The first two words (or header) of each event found in this way is highlighted in cyan and the header information is displayed on the left in a panel called “Event Info” (see figure below).

2.2 Event information panel

The second way to search is to select the known first word of an event with the mouse. Hit the forward button to find subsequent events. Remember that the word immediately after a record/block header is the first word of an event. Hint: selecting the first word of any bank structure (top level or not) will display all of its information

A quick note on the bank type. In CODA online, some tags are reserved for specific purposes. If a selected event has such a reserved tag, its purpose will be shown as the "Bank Type".

2.1.6 By Evio Faults

Look for faults or errors in the evio format by selecting the "Evio Fault" button. Simply hit the "Start Scan" button and this program scans the file from beginning to end (or as far as it can parse) and lists all blocks containing errors in a panel on the left called "Evio Errors" (which can be seen in figure 2.3 below).

The algorithm used to find these errors tries to parse as much of the file as possible. For example, if a block header length does not equal the sum of the lengths of all the events it contains, then the block header length is assumed for the moment to be correct and the event lengths in error. It tries to continue by scanning the next block and stops if it encounters an unrecoverable error or makes it to the end of the file.

Errors that are caught include bad/inconsistent values in a block/event header, wrong endianness of the displayed data, length of block header not consistent with length of contained events, and not enough data to read block/event (usually a bad length), and too large of an event count in a header. The search can go into events themselves to find lower level evio errors.

For an evio version 6 file, it will find inconsistencies between compression type and header values of compression word length and uncompressed data length. Any conflict between the index length and the number of events in a record will be flagged. Of course, if a file contains compressed data, evio events will not be scanned.

To print out suspicious record numbers or record header sizes, one must set the debug flag by hand in the scanFileForErrors() method of the EvioScanner(V6).java file.

Each block in which there is a problem is listed as a button. Click one and it hops to the beginning of that block which will be highlighted in red. Within that block, the ">" and "<" buttons move from event to event. If an event has an error, it is the last event to be accessible through the search buttons and will be highlighted in purple. If the event containing the error has an internal bank or structure with an error, it can also be

accessed through the search buttons and will be highlighted in orange. A corresponding error message (or messages) is displayed at the top of the gui in red text.

Below, a small file with evio format errors has been scanned. It reveals errors in 2 records. The first record is selected showing, in red, a header with an uncompressed data length of 0 even though there is no compression. It also shows the header saying it contains 3 events but there are entries in the index for only 2. Finally, it found an error in the first event, signified by its header in purple. The error is in a sub-structure, highlighted in orange. In this case a little investigation shows that the second bank header word shows padding of 2 for a data type of 32 bit unsigned int, when it should be 0.

Figure 2.3: Error Scanning