ISO/IEC JTC1/SC29/WG1
(ITU-T SG16)

Coding of Still Pictures

**JBIG**
Joint Bi-level Image Experts Group

**JPEG**
Joint Photographic Experts Group

**TITLE:**  
Common Test Conditions for JPEG XE v1.0

**EDITORS:**  
Nicolas Bourdis – nbourdis@prophesee.ai  
Tim Bruylants – tbr@intopix.com  
Walt Husak – wjh@dolby.com

**PROJECT:**  
-

**STATUS:**  
Final

**REQUESTED ACTION:** Distribution

**DISTRIBUTION:** Public

**Contact:**  
ISO/IEC JTC 1/SC 29/WG 1 Convener – Prof. Touradj Ebrahimi  
EPFL/STI/IEL/GR-EB, Station 11, CH-1015 Lausanne, Switzerland  
Tel: +41 21 693 2606, Fax: +41 21 693 7600, E-mail: Touradj.Ebrahimi@epfl.ch
Common Test Conditions for JPEG XE v1.0

1 Scope 2
2 Canonical raw event format 2
  2.1 Generic overview and principles 2
  2.2 Header 3
  2.3 Event field types 3
  2.4 Event data 4
  2.5 Some considerations 5
    State 5
      Absolute and relative timestamps 5
      Conservative approach to sizing the event data 5
      Flexibility vs complexity 5
  2.6 Example of an instance of the canonical raw event format 5
3 Reference dataset 6
4 Constrained and unconstrained operation scenarios 7
  4.1 Constrained operation scenario 7
  4.2 Unconstrained operation scenario 8
5 Evaluation methodology 8
  5.1 High-level methodology 8
  5.2 Key performance metrics 9
    5.2.1 Compression ratio 9
    5.2.2 Average event size in bits 9
    5.2.3 Normalized decoding time 9
    5.2.4 Decoding processing complexity 10
    5.2.5 Decoding memory complexity 10
    5.2.6 Normalized encoding time 10
    5.2.7 Encoding processing complexity 10
    5.2.8 Encoding memory complexity 10
6 Anchors 10
  6.1 EVT2 format 10
  6.2 Standard data compressors 11
1 Scope

This document describes the Common Test Conditions (CTC) for JPEG XE to evaluate lossless event-based coding solutions. Note that lossy event-based coding will be addressed at a later point in the standardization track.

The main objectives of this document are:

- Define the reference format and test dataset used for evaluation.
- Establish the evaluation methodology for comparison of the proposed coding solutions.
- Establish a formal definition of the key performance metrics used for evaluation.

Because there is not a complete overlap of methodologies and key performance metrics between live processing and file storage cases, this document addresses each of these cases separately. First, the reference format and dataset are defined in section 2. Then, section 3 defines the evaluation methodology and key performance metrics for live processing lossless formats. Finally, section 4 defines the evaluation methodology and key performance metrics for file storage lossless formats.

2 Canonical raw event format

To allow exchange and comparison of raw event data between different event coding solutions a common and generic event format is needed. This format, called the canonical raw event format, allows to represent the input data for encoders and the output data for decoders. It is designed primarily for easy minimal effort reading and writing and to be memory layout efficient. The canonical format does not apply any form of compression and can be thought of as the event-based equivalent to the Portable PixMap (PPM) format of regular images. However, “event data” as a common terminology covers a broad range of possible information types and contents. For example, the format needs to support many different types of events, with some event types potentially unknown at this point. Moreover, the bit sizes of various elements of the event data vary between manufacturers and sensor models. Thus, this canonical format also requires enough flexibility to support all the potential variations and differences.

2.1 Generic overview and principles

The canonical raw event format is designed with the following concepts:

- Binary, byte-aligned format.
- Starts with a single binary header to describe the layout of the event data that follows.
- Support for any type of event, identified by an event type and a collection of event fields.
- Event fields contain the real information of an event, and interpretation depends on the event field type. The format has predefined event field types, but also allows for user-defined event field types (whose interpretation is also user-defined).
- Event data follows immediately after the header. All events, regardless of their respective event type) use the same fixed number of bytes.

Figure 1 shows the overall structure of the canonical event format.
2.2 Header

The header is a variable sized structure that describes the binary layout of the event data that follows the header. It is structured as follows:

- **Magic**: 8 bytes fixed magic word containing ‘JPEG_REv’ (bytes 0 to 7 of the header).
- **Version**: 16-bit version identifier, currently set to 0x0000. The version allows for future updates or improvements to the canonical format.
- **Event type bit size**: 1 byte to specify the number of bits needed in each event to specify its respective event type.
- **Field descriptor count**: 1 byte to specify the number of field definitions, called descriptors, that follow.
- **Field descriptors**: 32-bit (4 bytes) per field descriptor. There are “field descriptor count” of these. Each field descriptor contains:
  - **Reserved byte**: always set to 0x00.
  - **Field bit size**: Number of bits that this field spans in an event.
  - **Field type**: A byte to indicate the type of the field (see below). Values between 0x00 and 0x7f are normatively defined by the canonical format. Values between 0x80 and 0xff are user defined field types that can be freely used. Interpretation of user-defined field types is out of scope.
  - **Field event type**: Specifies which event type the field belongs to. Fields are defined in the order of appearance in the associated events (from least significant bit positions to most significant bit positions).

2.3 Event field types

The canonical format comes with a set of predefined event field types that were identified as relevant and common for typical event data sets across vendors and sensor types. Pre-defined event field types
normatively define the interpretation of their respective content. Pre-defined event field types have a value between 0x00 and 0x7f. User-defined event field types have a range between 0x80 and 0xff. Each field type can only be assigned once to an event type, but it can be used for more than one event type.

### Table 1 – Canonical event field types

<table>
<thead>
<tr>
<th>Event field type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved for future use</td>
<td>Not to be used.</td>
</tr>
<tr>
<td>0x01</td>
<td>Polarity event</td>
<td>Specifies an event polarity. Considered “abstract” and typically just 1 bit. Generates an event.</td>
</tr>
<tr>
<td>0x02</td>
<td>X coordinate</td>
<td>Specifies the originating X coordinate.</td>
</tr>
<tr>
<td>0x03</td>
<td>Y coordinate</td>
<td>Specifies the originating Y coordinate.</td>
</tr>
<tr>
<td>0x04</td>
<td>External trigger address event</td>
<td>Specifies the originating trigger address or ID. Generates an event.</td>
</tr>
<tr>
<td>0x05</td>
<td>Sensor ID</td>
<td>Specifies a sensor ID where the event was generated. This allows for systems with 2 or more sensors.</td>
</tr>
<tr>
<td>0x06</td>
<td>Padding</td>
<td>Allows for the definition of custom padding fields. Bits inside padding fields are set to zero.</td>
</tr>
<tr>
<td>0x10</td>
<td>Absolute timestamp (LSB)</td>
<td>Sets the lower bits of the active timestamp base (other bits are untouched).</td>
</tr>
<tr>
<td>0x11</td>
<td>Absolute timestamp (MSB)</td>
<td>Combines with the LSB of absolute timestamp base.</td>
</tr>
<tr>
<td>0x12</td>
<td>Relative timestamp</td>
<td>Offsets to the absolute timestamp base.</td>
</tr>
<tr>
<td>&lt; 0x80</td>
<td>Reserved for future use</td>
<td>Remaining event field types, not to be used.</td>
</tr>
<tr>
<td>0x80 to 0xff</td>
<td>User-defined (custom)</td>
<td>Custom event field types. Interpretation of the content bits is out of scope.</td>
</tr>
</tbody>
</table>

#### 2.4 Event data

After the header, event data follows. Each event contains the concatenation of the bits of its event fields, in the order as defined in the header (little endian ordering). An event field is part of an event when the respective event types match. A consequence is that the canonical format allows defining different types of events that do not necessarily occupy the same number of bits (due to varying fields and field sizes). However, to facilitate reading and writing and allow easy random access all events are padded up to become byte-aligned and to all occupy an equal number of bytes, regardless of their respective event type. The total byte size of each event is equal to the maximum of the size of all defined event types (sum of all bit sizes of individual fields per event type, rounded up to byte alignment, using padding with 0 bits).
Then, events are just concatenated into one stream of events immediately after the header. Events must be sorted in order of increasing timestamps, then (for same timestamp values) in order of increasing Y-coordinates, then (for same timestamp and Y-coordinate values) in order of increasing X-coordinates, then (for same timestamp, Y-coordinate and X-coordinate values, although unlikely) in order of increasing polarity.

2.5 Some considerations

State

The event data that follows the header encodes foremost its event type. Via the event type, the fields of the respective event data instance are known (field type and field bit size). Each field of such an individual event data instance contains information that updates a set of variables in the parser/writer according to the field types it contains. Every event data instance (a group of bytes) is of one particular event type. If the event type contains a generating-event-field, then an event is generated using all information available and relevant for the field — being the information encoded in the event data instance plus possibly the information that is contained in the state variables (e.g. think most recent absolute timestamp or last-seen Y-coordinate). If the event type contains no generating-event-field, then only the state is updated, and no output event is generated.

Currently only polarity (0x01) and external trigger (0x04) are event-generating fields.

Absolute and relative timestamps

An absolute timestamp is kept and managed as a “state” variable, initialized at 0. Type 0x10 (absolute timestamp LSB) updates the lower bit portion of the state variable. Type 0x11 (absolute timestamp MSB) updates the higher bit portion of the state variable (upshift by field size of the LSB). Events without a timestamp field will implicitly happen at the timestamp present in the state variable. Events with a type 0x12 relative timestamp will have “abs_ts + rel_ts” as final timestamp.

Conservative approach to sizing the event data

The canonical raw event format allows defining content without significantly inflating the bit sizes of fields and while keeping the overall event sizes optimal for the current purpose.

Flexibility vs complexity

It is possible to hard-code one specific instance of this canonical format by designing a specific raw event layout. In such a case, an implementation can choose to perform a binary comparison of the header to verify compliance without actually interpreting the header. This CTC provides such a predefined header for the reference dataset.

This format allows for the addition of new fields in the future, while maintaining backwards compatibility.

2.6 Example of an instance of the canonical raw event format

This section provides an example of an instantiation of the canonical raw event format. This example defines three types of events: a) a contrast detection (CD) event, b) a trigger event, and c) an absolute timestamp event. The binary header is given in figure 2.
This example header defines three event types and 7 event fields in total. With this header, the three event types are as given in figure 3.

The CD event type (type 0x00) uses 4 fields that take up 48 bits in total (which equals to 6 bytes). The trigger event type (type 0x01) uses two fields that only take up 27 bits. The absolute timestamp event type (type 0x03) also uses 48 bits. As stated before, all events are padded up to occupy the same number of bits and additionally to a byte-alignment. The CD event type and the Absolute Timestamp event type are the biggest of the event types with 48 bits each. Thus, the trigger event type is padded with 21 bits to make it equally sized. In this example there is no need to apply additional byte-alignment padding because the 48-bit size is already byte-aligned (by coincidence in this example). Thus, all events in this example will occupy 6 bytes.

### Reference dataset

The reference dataset is a set of raw event sequences acquired on a wide variety of use cases, environment conditions, durations, sensors and sensor configurations. Given the fast-growing number of combinations, the reference dataset is not meant to exhaustively represent all possible situations, but instead focuses on relevant & complementary scenarios. All raw event sequences in this dataset are provided in the canonical input format described above. A JPEG GitLab repository [7] is available with the information to download the reference dataset and the source code for software to read and write the canonical input format.

The reference dataset is encoded using the canonical raw event format, using the same header specification as given in the example above in figure 2. This specification yields event data of 48 bits (6 bytes) per instance.

### Table 2 – Reference dataset

<table>
<thead>
<tr>
<th>Sequence name</th>
<th>Resolution</th>
<th>Duration (s)</th>
<th>Event-rate (Mev/s)</th>
</tr>
</thead>
</table>

4 Constrained and unconstrained operation scenarios

This section describes the two considered scenarios for operation of event-based technology, which are illustrated in Figure 4 below:

Figure 4 – Typical constrained and unconstrained operation scenarios

4.1 Constrained operation scenario

The central scenario for event-based technology involves real-time execution of an application from the live sensor stream, illustrated at the top of Figure 4. Typically, in this scenario, event data is immediately
encoded by the sensor and streamed to the application processor, where it is decoded and further processed in real-time by the application.

In this scenario, main properties for a candidate standard format are:

- possibility to implement the encoder block in the sensor (extremely limited complexity),
- low end-to-end algorithmic latency,
- low complexity of the decoder block on the application processor, and
- reasonable compression ratio, to enable high throughput on a variety of transport channels.

More precisely, to be applicable to the constrained operation case, formats need to comply with the following constraints:

- Latency: end-to-end algorithmic latency of the encoding / decoding pipeline needs to be less than one least significant bit of timestamp according to the reference format (in timestamp unit, not in real time unit).
- Complexity: the algorithm shall be defined in such a way to allow for low complexity implementation on hardware or embedded CPU.
  - Encoding in hardware must be possible, with a buffer of up to 64 events.
  - Decoding in real-time at 35 Mev/s, in a compliant manner with the requirements, should be possible using at most 50% of CPU usage of an ARM Cortex-A53 Quad-Core 64-bits (for instance, included in NXP i.MX8M-quad platform) or equivalent.

4.2 Unconstrained operation scenario

Another important scenario for event-based technology involves compressing a sequence of events and storing or transferring it for processing at a later stage, illustrated at the bottom of Figure 4. Typically, in this scenario, data is initially streamed from a live sensor (hence encoded by the sensor), then transcoded and compressed on the recording platform into an encoded stream for storage or transfer. Then in a later stage, the encoded stream is decoded to be further processed, potentially on a different platform.

In this scenario, the main properties for a candidate standard format are:

- high compression ratio,
- relaxed constraints on complexity & latency of encoder and decoder, compared to the constrained scenario, and
- software implementations both for the encoder and decoder blocks.

5 Evaluation methodology

5.1 High-level methodology

To evaluate candidate formats, both in the constrained and unconstrained scenarios above, the encoding and decoding pipeline is defined in Figure 6.
Figure 6 – Encoding and decoding pipeline for evaluation purposes

The encoder block takes as input an event sequence in canonical format (see definition in section 2) and produces the encoded format. The decoder block takes as input the encoded bitstream and produces the canonical format again. For the lossless case, the reference event data at the input and output are expected to be identical (see disambiguation constraints defined in section 2.4).

This process enables the evaluation of decoded event-based sequences and of the encoding and decoding processes using the reference dataset (see definition in section 3). For this purpose, we compare candidate formats based on the following metrics:

- compression ratio
- average encoded event size in bits
- normalized decoding time
- decoding processing complexity
- decoding memory complexity
- normalized encoding time
- encoding processing complexity
- encoding memory complexity

Precise definitions for these metrics are defined in the next section.

5.2 Key performance metrics

This section provides accurate definitions for the key performance metrics mentioned in the previous section. The JPEG GitLab repository [7] will contain an implementation of these key performance metrics, provided as source code.

5.2.1 Compression ratio

The compression ratio quantifies the size gain when encoding the data, compared to the canonical format as defined in section 2.6. It is calculated as the ratio between the canonical input size and the encoded size and expressed in percentages:

\[ C_R = \frac{\text{size(input)}}{\text{size(encoded)}} \]

Because of the limited compression performances of the canonical input format, compression ratios of proposals should be compared to the anchors listed in section 6, by division with the compression ratio for the anchors:

\[ \frac{C_R(\text{proposal})}{C_R(\text{anchor})} \]

5.2.2 Average event size in bits

The average event size in bits quantifies the average number of bits used to represent an event. This is calculated as the ratio between the size of the encoded sequence and the total number of events (CD events plus others) in the input sequence and expressed in bits/event:

\[ S' = \frac{\text{size(encoded)}}{\#(\text{Input})} \]

5.2.3 Normalized decoding time

The decoding time quantifies the time spent at application processor level to decode a sequence using a single processing core, expressed in seconds. To make the decoding time less dependent on the chosen execution platform, the raw decoding time measurements are normalized by the time for a reference task executed on the same platform.
The reference task is defined as the computation of the average event rate for the `psee_structuredlight_fan` sequence in canonical format, consisting in reading the input sequence, counting the total number of events, calculating the total sequence duration (i.e. difference of last by first event timestamp), and finally calculating the average event rate as the ratio of the total number of events by the total sequence duration.

The normalized decoding time is then computed as the ratio of the decoding time by the time taken for the reference task as described above, expressed in percentages: 
\[ D_{n_d} = \frac{\text{time}(\text{decoding})}{\text{time}(\text{reference task})} \]

To reduce dependency of this metric on external factors such as system CPU scheduling or disk caching, this metric must be averaged on a minimum of 10 runs.

5.2.4 Decoding processing complexity

The decoding processing complexity quantifies the complexity of the decoding algorithm, expressed in the number of retired CPU instructions \[1\]. This metric is denoted \( C_{p_d} \).

5.2.5 Decoding memory complexity

The decoding memory complexity quantifies the total memory \[2\] allocated by the decoding algorithm, expressed in MB of allocated memory. This metric is denoted \( C_{m_d} \).

5.2.6 Normalized encoding time

The encoding time quantifies the time spent at application processor level to encode a sequence using a single processing core, expressed in seconds. To make the encoding time less dependent on the chosen execution platform, we normalize the raw encoding time measurements by the time for a reference task (see definition in section 5.2.3), expressed in percentages: 
\[ D_{n_e} = \frac{\text{time}(\text{encoding})}{\text{time}(\text{reference task})} \]

To reduce dependency of this metric on external factors such as system CPU scheduling or disk caching, this metric must be averaged on a minimum of 10 runs.

5.2.7 Encoding processing complexity

The encoding processing complexity quantifies the complexity of the encoding algorithm, expressed in the number of retired CPU instructions \[1\]. This metric is denoted \( C_{p_e} \).

5.2.8 Encoding memory complexity

The encoding memory complexity quantifies the total memory \[2\] allocated by the encoding algorithm, expressed in MB of allocated memory. This metric is denoted \( C_{m_e} \).

6 Anchors

6.1 EVT2 format

EVT2 \[3\] is a format used in all recent Prophesee sensors.

For convenience, the reference dataset encoded in the EVT2 format can be downloaded using this link: 
https://kdrive.infomaniak.com/app/share/975517/ff0eb5f8-a55e-4eaa-8c9d-393a8911276a. Stand-alone C++ code to encode and decode EVT2 can be found on the Prophesee OpenEB repository: CSV to EVT2.0
encoder, EVT2.0 to CSV decoder. These stand-alone C++ samples are released under Apache v2 license, can be built using CMake, do not require any particular dependency, and are compatible with recent Ubuntu and Windows versions. Python developers can also install the software from Prophesee website and refer the file to CSV Python sample.

Table 3 below provides KPI measurements, measured on a Lenovo ThinkPad T460p laptop (Windows 10 64-bits, Intel Core i7 6820HQ 2.7 GHz CPU, 16 GB DDR4 RAM memory, SAMSUNG 512GB SSD SATA 6.0GBps drive), using the EVT2 format:

<table>
<thead>
<tr>
<th>Sequence name</th>
<th>Decoding speed (Mev/s)</th>
<th>Encoded event size in bits (bits/ev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>activemarkers_handheld</td>
<td>91.9</td>
<td>36.51</td>
</tr>
<tr>
<td>automotive_urban</td>
<td>81.0</td>
<td>32.10</td>
</tr>
<tr>
<td>deblur_street</td>
<td>79.2</td>
<td>32.05</td>
</tr>
<tr>
<td>depthsensing_highspeedlaser</td>
<td>107.7</td>
<td>32.05</td>
</tr>
<tr>
<td>eyetracking_right</td>
<td>84.0</td>
<td>32.51</td>
</tr>
<tr>
<td>industrial_counting</td>
<td>57.6</td>
<td>34.74</td>
</tr>
<tr>
<td>industrial_fluidflow</td>
<td>85.5</td>
<td>33.73</td>
</tr>
<tr>
<td>industrial_spray</td>
<td>76.7</td>
<td>33.82</td>
</tr>
<tr>
<td>localization_cube</td>
<td>88.9</td>
<td>32.24</td>
</tr>
<tr>
<td>structuredlight_fan</td>
<td>106.1</td>
<td>32.03</td>
</tr>
<tr>
<td>surveillance_startracking</td>
<td>77.6</td>
<td>33.74</td>
</tr>
</tbody>
</table>

6.2 Standard data compressors

Another anchor is to compress the EVT2 format with lossless entropy coders such as lz4 [4], bzip2 [5] and 7zip [6]. Doing so is an easy approach to further reduce the size of each sequence. Lz4 was selected because it has low complexity and allows real-time compression and decompression on any modern computer system. Bzip2 offers improved compression performance over lz4 at the cost of complexity. 7zip goes even further but uses a lot of CPU time (minutes) to compress each sequence.
<table>
<thead>
<tr>
<th>Name</th>
<th>raw (bytes)</th>
<th>lz4 9 (bytes)</th>
<th>bz2 9 (bytes)</th>
<th>7z mx9 (bytes)</th>
<th>lz4 9</th>
<th>bz2 9</th>
<th>7z mx9</th>
</tr>
</thead>
<tbody>
<tr>
<td>activemarkers_handheld</td>
<td>27,699,917</td>
<td>25,297,061</td>
<td>19,051,531</td>
<td>12,998,194</td>
<td>91%</td>
<td>69%</td>
<td>47%</td>
</tr>
<tr>
<td>automotive_urban</td>
<td>1,305,928,808</td>
<td>1,305,930,071</td>
<td>1,167,338,779</td>
<td>804,193,436</td>
<td>100%</td>
<td>89%</td>
<td>62%</td>
</tr>
<tr>
<td>deblur_street</td>
<td>621,175,968</td>
<td>621,176,579</td>
<td>527,920,367</td>
<td>247,603,329</td>
<td>100%</td>
<td>85%</td>
<td>40%</td>
</tr>
<tr>
<td>depthsensing_highspeedlaser</td>
<td>465,950,756</td>
<td>465,951,219</td>
<td>376,162,119</td>
<td>221,911,221</td>
<td>100%</td>
<td>81%</td>
<td>48%</td>
</tr>
<tr>
<td>eyetracking_right</td>
<td>230,971,667</td>
<td>229,200,210</td>
<td>198,195,710</td>
<td>163,803,321</td>
<td>99%</td>
<td>86%</td>
<td>71%</td>
</tr>
<tr>
<td>industrial_counting</td>
<td>19,929,563</td>
<td>18,940,003</td>
<td>14,298,325</td>
<td>11,707,463</td>
<td>95%</td>
<td>72%</td>
<td>59%</td>
</tr>
<tr>
<td>industrial_fluidflow</td>
<td>44,581,212</td>
<td>43,884,373</td>
<td>38,399,634</td>
<td>36,704,922</td>
<td>98%</td>
<td>86%</td>
<td>82%</td>
</tr>
<tr>
<td>industrial_spray</td>
<td>23,757,856</td>
<td>23,626,667</td>
<td>19,717,351</td>
<td>18,687,463</td>
<td>99%</td>
<td>83%</td>
<td>79%</td>
</tr>
<tr>
<td>localization_cube</td>
<td>332,960,062</td>
<td>331,343,082</td>
<td>263,474,113</td>
<td>176,566,564</td>
<td>100%</td>
<td>79%</td>
<td>53%</td>
</tr>
<tr>
<td>structuredlight_fan</td>
<td>3,728,999,557</td>
<td>3,712,835,660</td>
<td>2,612,736,175</td>
<td>726,072,266</td>
<td>100%</td>
<td>70%</td>
<td>19%</td>
</tr>
<tr>
<td>surveillance_startracking</td>
<td>95,841,092</td>
<td>89,131,283</td>
<td>66,340,210</td>
<td>53,562,167</td>
<td>93%</td>
<td>69%</td>
<td>56%</td>
</tr>
</tbody>
</table>

References