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

Coding of Still Pictures

JBIG	JPEG
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE:	**Common Test Conditions for JPEG XE v2.1**
SOURCE:	ISO/IEC SC29 WG1
PROJECT:	-
STATUS:	Final
REQUESTED ACTION:	For information
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

Table of contents

Proposed Common Test Conditions for JPEG XE v2.1

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.

The structure of this document is as follows. 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 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 understood as the event-based equivalent of the Portable PixMap (PPM) format for regular images. However, event data includes a wider range of possible information types and content. For example, the event-based 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 General 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.

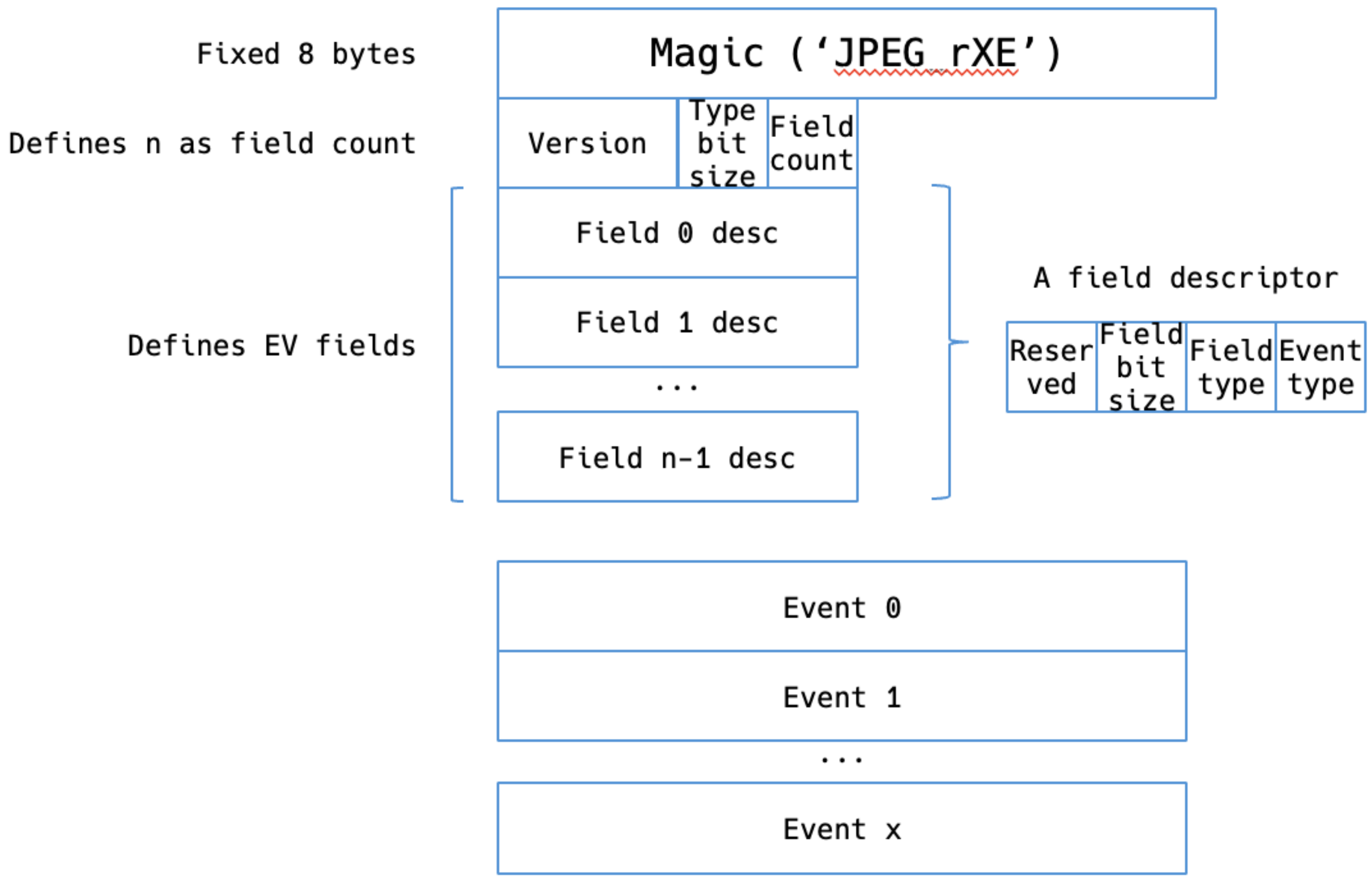


Figure 1 – Canonical event format structure.

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_rXE`' (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 define the number of bits that is used 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. The number of field descriptors are specified by the "field descriptor count". Each field descriptor consists of 4 bytes, being:
    - One reserved byte: always set to `0x00`.
    - Field bit size: Number of bits that this field spans when included 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.

- Event type: Specifies which event type the field belongs to.

Every field descriptor will specify one field for exactly one event type. When an event (of some type) is described, the fields will appear by the order they were defined (from least significant bit positions to most significant bit positions).

2.3 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 type matches. 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.

2.4 Concepts and Principles

State

The event data that follows the header encodes most importantly the event type which allows to obtain the fields of the respective event data instance (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 output is generated using all information available and relevant – being the information encoded in the event data instance plus possibly the information that is contained in the state variables (e.g. most recent absolute timestamp or last-seen Y-coordinate). If the event type contains at least one generating-event field, then state is updated, and an event output is generated. If the event type does not contain any generating-event fields, then only the state is updated, and no event output is generated.

Currently, only polarity (`0x01`) is an event-generating field.

Absolute and relative timestamps

An absolute timestamp is kept and managed as a "state" variable, initialized at 0. The event field type of `0x10` (absolute timestamp LSB) updates the lower bit portion of the state variable. The event field `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.

Flexibility vs complexity

The canonical format is rather flexible since it allows the design of a specific raw event layout, as one instantiation of this format. 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.

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

2.5 Event field types

The canonical format comes with a set of predefined field types that were identified as relevant and common for typical event data sets across vendors and sensor types. Pre-defined field types normatively define the interpretation of their respective content and have a value between `0x00` and `0x7f`. User-defined field types have a range between `0x80` and `0xff`. Each field type descriptor can only be assigned once to an event type (via a field descriptor), but it can be used for more than one event type. Event field types have specific functionality, as specified in the description column of Table 1, when parsed and always update the internal state of the parser (see State in Concepts and Principles), while some also cause the generation of an actual event output.

Table 1 – Canonical field types.

Field type	**Name**	**Description**
`0x00`	*Reserved for future use*	Not to be used.
`0x01`	Polarity	Specifies an event polarity. Considered "abstract" and typically just 1 bit. Generates an event output.
`0x02`	X coordinate	Specifies the originating X coordinate.
`0x03`	Y coordinate	Specifies the originating Y coordinate.
`0x04`	External trigger address event	Specifies the originating trigger address or ID.
`0x05`	Sensor ID	Specifies a sensor ID where the event was generated. This allows for systems with 2 or more sensors.
`0x06`	Padding	Allows for the definition of custom padding fields. Bits inside padding fields are set to zero. This field type is useful to provide explicit padding between other defined fields in an event.
`0x10`	Absolute timestamp (LSB)	Sets the lower bits of the absolute timestamp base (other bits are untouched).
`0x11`	Absolute timestamp (MSB)	Sets the high bits of the absolute timestamp base (other bits are untouched). As such it combines with the LSB of absolute timestamp base.
`0x12`	Relative timestamp	Offsets to the absolute timestamp base.
`Any other value < 0x80`	*Reserved for future use*	Remaining event field types, not to be used.
`0x80 to 0xff`	User-defined (custom)	Custom event field types. Interpretation of the content bits is out of scope.

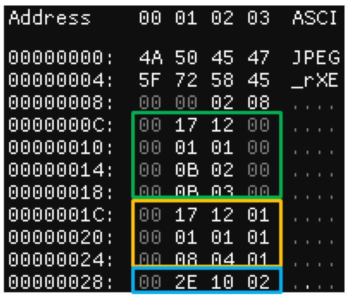
2.6 The JPEG XE canonical raw event format

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.

Event type 0 – CD event
- 00 17 12 00 (Relative timestamp, 23 bits)
- 00 01 01 00 (Polarity, 1 bit)
- 00 0b 02 00 (X coordinate, 11 bits)
- 00 0b 03 00 (Y coordinate, 11 bits)

Event type 1 – Trigger event
- 00 17 12 01 (Relative timestamp, 23 bits)
- 00 01 01 01 (Polarity, 1 bit)
- 00 08 04 01 (Trigger address 8, bits)

Event type 2 – ABS Timestamp
- 00 2e 10 02 (Absolute timestamp LSB, 46 bits)



```
Address    00 01 02 03  ASCI

00000000:  4A 50 45 47  JPEG
00000004:  5F 72 58 45  _rXE
00000008:  00 00 02 08  ....
0000000C:  00 17 12 00  ....
00000010:  00 01 01 00  ....
00000014:  00 0B 02 00  ....
00000018:  00 0B 03 00  ....
0000001C:  00 17 12 01  ....
00000020:  00 01 01 01  ....
00000024:  00 08 04 01  ....
00000028:  00 2E 10 02  ....
```

Figure 2 – Example of a canonical event format header.

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.

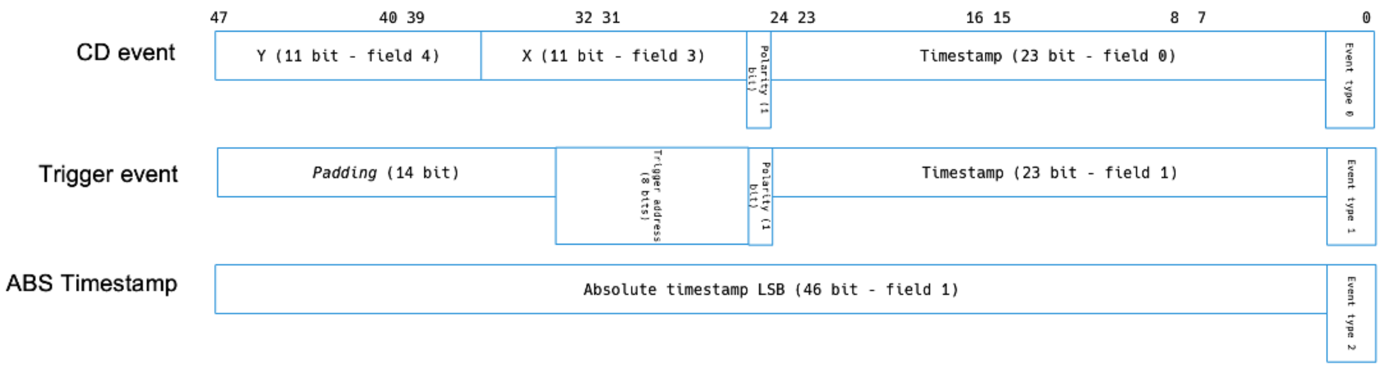


Figure 3 – Three event types as defined by the example header.

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 be byte-aligned. 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.

Table 2 – Overview of the three JPEG XE canonical raw event format event types.

Event type	**Name**	**Description**	**Output**
`0x00`	CD event	A contrast detection event with a relative 23-bit timestamp, 1 polarity bit, and 11-bit X and Y coordinates.	Yes
`0x01`	Trigger event	A trigger event with a relative 23-bit timestamp, 8-bit trigger ID, and 1 polarity bit.	Yes
`0x02`	ABS timestamp	A 46-bit timestamp (LSB).	Updates state

3 Reference dataset

The reference dataset is given in Table 3 and consists of a set of raw event sequences collected over a wide range 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 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. All of the sequences belong to a class that represents the density (event rate) of the sequences.

Table 3 – Reference dataset.

Class	***Sequence name***	***Resolution***	***Duration (s)***	***Event-rate (Mev/s)***
Low	activemarkers_handheld	1280x720	10.1	0.6
Low	industrial_counting	640x480	6.3	0.73
Low	industrial_spray	640x480	4.3	1.3
Low	surveillance_startracking	1280x720	12.3	1.84
Low	industrial_fluidflow	1280x720	5.6	1.88
Low	eyetracking_right	1280x720	9.9	5.75
Low	localization_cube	640x480	10	8.29
Low	m9_hud_flicker	1032x928	0.42	10.84
Low	eyetracking_dark_pupil	320x320	10	15.26
Mid	human_walking	1032x928	0.42	20.1
Mid	automotive_urban	1280x720	10.2	32
Mid	car	1032x928	0.42	34.69
Mid	depthsensing_highspeedlaser	1280x720	3	38.8
Mid	farview	1032x928	0.42	42.6
Mid	deblur_street	1280x720	2.4	64.6
Mid	walking2	1920x1078	0.5	67.8
Mid	driving2	1920x1078	0.5	84.6

Mid	watching1	1920x1078	0.5	86
High	tree	1032x928	0.42	89.1
High	structuredlight_fan	1280x720	10	93.1
High	driving_rouen	1280x720	2	134.96
High	driving1	1920x1078	0.5	138.2
High	opera_panel	1280x720	0.067	195.8
High	walking1	1920x1078	0.5	257.8
High	flicker_train	1280x720	1	336.1
High	aircraft_drone	1032x928	0.42	411.3
High	fan	1032x928	0.42	632.1

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. Because there is not a complete overlap of methodologies between constrained (live processing) and unconstrained (file storage), this section addresses each of these cases separately.

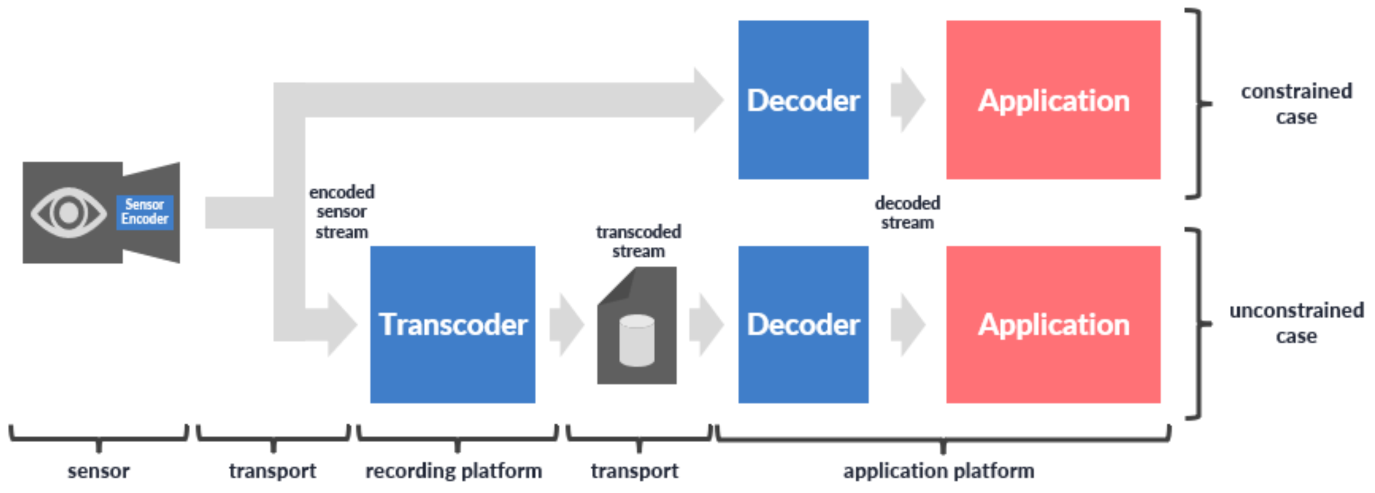


Figure 4 – Typical constrained and unconstrained operation scenarios

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 future 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
- moderate 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).
- 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 future 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 the candidate formats, both in the constrained and unconstrained scenarios above, the encoding and decoding pipeline is defined in figure 5.

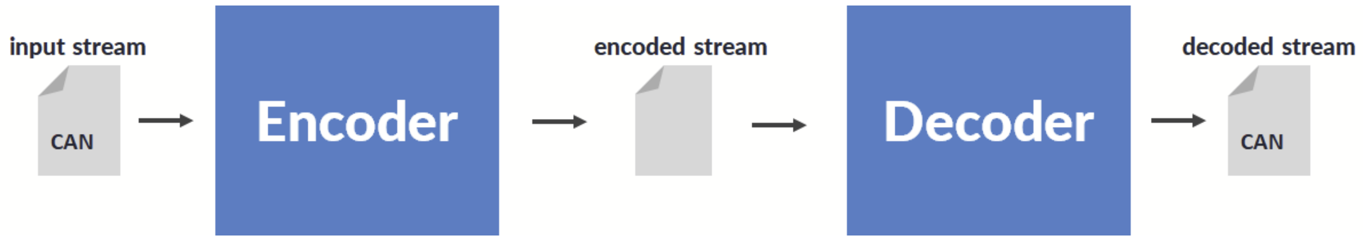


Figure 5 – Encoding and decoding pipeline for evaluation purposes

The encoder block takes as input an event sequence in canonical format (see definition in section 2) and produces the encoded bitstream. The decoder block takes as input the encoded bitstream and produces the canonical format again. For the lossless case, the input event data and the decoded output event data should relate as defined in the JPEG XE Use Cases and Requirements document [8] ("Lossless coding of events").

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] contains guidelines and instructions for the calculation of these key performance metrics. These instructions should be used to evaluate the performance of the candidate event-based encoders and decoders.

5.2.1 Compression ratio

The compression ratio quantifies the relative gain of the compressed data stream, 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 data stream size and expressed in percentages: $C_R = \frac{size(input)}{size(encoded)}$.

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 with the compressed ratios achieved for the anchors: $\frac{C_R(proposal)}{C_R(anchor)}$.

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 data stream and the total number of events (CD events plus others) in the input sequence and is expressed in bits/event: $S = \frac{size(encoded)}{\#(input)}$.

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 hardware platform selected for decoding, the raw decoding time measurements are normalized by the time for a reference task executed on the same platform.

Note: A reference task is defined and implemented as part of the CTC Tools repository [7].

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 the last by the 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: $Dn_d = \frac{time(decoding)}{time(reference\ task)}$.

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 Normalized decoding throughput

This corresponds to the number of events in the sequences divided by the normalized decoding time, and is expressed in Mev/s.

5.2.5 Decoding processing complexity

The decoding processing complexity quantifies the complexity of the decoding algorithm, expressed in the number of retired CPU instructions as measured with [1]. This metric is denoted Cp_d.

5.2.6 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 Cm_d.

5.2.7 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 hardware platform selected for encoding, the raw encoding time measurements are normalized by the time for a reference task (see definition in section 5.2.3) executed on the same platform.

The normalized encoding time is then computed as the ratio of the encoding time by the time taken for the reference task (as defined in 5.2.3), expressed in percentages: $Dn_e = \frac{time(encoding)}{time(reference\ task)}$.

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.

5.2.8 Encoding processing complexity

The encoding processing complexity quantifies the complexity of the encoding algorithm, expressed in the number of retired CPU instructions as measured with [1]. This metric is denoted Cp_e.

5.2.9 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 Cm_e.

6 Anchors

6.1 EVT2 format

EVT2 [3] is a streaming format used by several event-based sensors on the market. This format is lossless and leads to a reasonable compression ratio, about 34 bits/event on average, and a high decoding speed, about 334 Mev/s.

For convenience, a download link to the reference dataset encoded in the EVT2 format can be found in the CTC Tools repository [7], along with stand-alone C++ code to encode and decode EVT2, from and to CSV format.

Table 4 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 4 – Anchor KPI evaluation for EVT2 format on the reference dataset.

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

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 5 – Lossless compression anchor results.

Name	raw (bytes)	lz4 9 (bytes)	bz2 9 (bytes)	7z mx9 (bytes)	lz4 9	bz2 9	7z mx9
activemarkers_handheld	27,699,917	25,297,061	19,051,531	12,998,194	91%	69%	47%
automotive_urban	1,305,928,808	1,305,930,071	1,167,338,779	804,193,436	100%	89%	62%
deblur_street	621,175,968	621,176,579	527,920,367	247,603,329	100%	85%	40%
depthsensing_highspeedlaser	465,950,756	465,951,219	376,162,119	221,911,221	100%	81%	48%
eyetracking_right	230,971,667	229,200,210	198,195,710	163,803,321	99%	86%	71%
industrial_counting	19,929,563	18,940,003	14,298,325	11,707,463	95%	72%	59%
industrial_fluidflow	44,581,212	43,884,373	38,399,634	36,704,922	98%	86%	82%
industrial_spray	23,757,856	23,626,667	19,717,351	18,687,463	99%	83%	79%
localization_cube	332,960,062	331,343,082	263,474,113	176,566,564	100%	79%	53%
structuredlight_fan	3,728,999,557	3,712,835,660	2,612,736,175	726,072,266	100%	70%	19%
surveillance_startracking	95,841,092	89,131,283	66,340,210	53,562,167	93%	69%	56%

References

[1] Linux kernel profiling with Perf, https://perf.wiki.kernel.org/index.php/Tutorial.

[2] Heaptrack - a heap memory profiler for Linux, https://github.com/KDE/heaptrack.

[3] Prophesee, “EVT2.0”, https://docs.prophesee.ai/stable/data/encoding_formats/evt2.html.

[4] Lz4, https://lz4.org/.

[5] Bzip2, https://sourceware.org/bzip2/.

[6] 7zip, https://www.7-zip.org/.

[7] ISO/IEC JTC1/SC29/WG1, “GitLab Repository for JPEG XE CTC Tools”, https://gitlab.com/wg1/jpegxe/ctc_tools.

[8] ISO/IEC JTC1/SC29/WG1 WG1N100887, Use Cases and Requirements for JPEG XE v2.0, July 2024.