|  |  |
| --- | --- |
| Project | **Specification of Sensor Interface for Cyber and Physical World**  <<https://sagroups.ieee.org/2888/> **>** |
| Title | **Syntax and semantics of audio-video sensor data** |
| DCN | **2888-20-0023-00-0001** |
| Date Submitted | **July 20, 2020** |
| Source(s) | Sang-Kyun Kim, [goldmunt@gmail.com](mailto:goldmunt@gmail.com) (Myongji University)  Min Hyuk Jeong, [jmh8900@gmail.com](mailto:jmh8900@gmail.com) (Myongji University)  Kyoungro Yoon, [yoonk@konkuk.ac.kr](mailto:yoonk@konkuk.ac.kr) (Konkuk University)  Sangkwon Jeong, [ceo@joyfun.kr](mailto:ceo@joyfun.kr) (Joyfun)  HyeonWoo Nam, [hwnam@dongduk.ac.kr](mailto:hwnam@dongduk.ac.kr) (Dongduk Women’s University)  Dong Soo Choi, [soochoi@dau.ac.kr](mailto:soochoi@dau.ac.kr) (Dong-A University)  Jeonghwoan Choi [jordhanchoi@skonec](mailto:ceo@joyfun.kr).com (Skonec Entertainment) |
| Re: |  |
| Abstract | This contribution illustrates the basic JSON schema structure for representing audio-video sensor information in the physical world in a standardized data format. The semantics and examples of the environmental sensor information are presented. |
| Purpose | To start discussion on purpose of the standard |
| Notice | This document has been prepared to assist the IEEE 2888 Working Group. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein. |
| Release | The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE’s name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE’s sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that IEEE 2888 may make this contribution public. |
| Patent Policy | The contributor is familiar with IEEE patent policy, as stated in [Section 6 of the IEEE-SA Standards Board bylaws](http://standards.ieee.org/guides/opman/sect6.html#6.3) <[http://standards.ieee.org/guides/bylaws/sect6-7.html#6](http://127.0.0.1:4664/cache?event_id=757737&schema_id=1&s=5X0vID10lu_E6yrIkWkNd4Wz2H8&q=hancock)> and in *Understanding Patent Issues During IEEE Standards Development* <http://standards.ieee.org/board/pat/faq.pdf> |

# Data formats for audio-video sensors

## Microphone sensor

### General

This sub-clause specifies a sensor data type, which describes an audio information captured from a microphone sensor.

### Syntax

|  |
| --- |
| "microphoneSensorType": {  "type": "object",  "properties": {  "microphoneOrientation": {  "$ref": "#/definitions/orientationSensorType"  },  "microphoneLocation": {  "$ref": "#/definitions/globalPositionSensorType "  },  "microphoneAltitude": {  "$ref": "#/definitions/altitudeSensorType"  },  "audioData": {  "$ref": "#/definitions/rawAudioType"  }  },  "additionalProperties": false  },  "rawAudioType": {  "type": "object",  "properties": {  "sample\_rate": {  "$ref": "#/definitions/sampleRateType"  },  "byte\_order": {  "$ref": "#/definitions/byteOrderType"  },  "sign": {  "$ref": "#/definitions/signType"  },  "resolution": {  "$ref": "#/definitions/resolutionType"  }  },  "additionalProperties": false  },  "sampleRateType": {  "type": "number",  "enum": [  8.0,  11.025,  16.0,  22.05,  32.0,  44.056,  44.1,  47.25,  48.0,  50.0,  50.4,  88.2,  96.0,  176.4,  192.0,  352.8,  2822.4,  5644.8  ]  },  "byteOrderType": {  "type": "string",  "enum": [  "littleEndian",  "bigEndian"  ]  },  "signType": {  "type": "string",  "enum": [  "signed",  "unsigned"  ]  },  "resolutionType": {  "type": "integer",  "enum": [  4,  8,  12,  16,  20,  24,  32,  48,  64  ]  }, |

### Semantics

Semantics of the microphoneSensorType:

| *Name* | *Definition* |
| --- | --- |
| microphoneSensorType | Tool for describing sensed information with respect to a microphone sensor. |
| microphoneLocation | Describes the location of a microphone using the structure defined by globalPositionSensorType. |
| microphoneOrientation | Describes the orientation of a microphone using the structure defined by orientationSensorType. |
| microphoneAltitude | Describes the altitude of a microphone using the structure defined by altitudeSensorType. |
| audioData | Describes audio data refer to rawAudioType. |
| sample\_rate | Sample rate is the number of samples of audio carried per second, measured in Hz or kHz (one kHz being 1 000 Hz). For example, 44 100 samples per second can be expressed as either 44 100 Hz, or 44.1 kHz. Bandwidth is the difference between the highest and lowest frequencies carried in an audio stream. |
| byte\_order | It tells how the data is stored with the most significant byte on one end or the other. When more than one byte is used to represent a PCM sample, the byte order (big endian vs. little endian) must be known. Due to the widespread use of little-endian Intel CPUs, little-endian PCM tends to be the most common byte orientation. |
| sign | It is not enough to know that a PCM sample is, for example, 8 bits wide. Whether the sample is signed or unsigned is needed to understand the range. If the sample is unsigned, the sample range is 0...255 with a center point of 128. If the sample is signed, the sample range is -128...127 with a center point of 0. If a PCM type is signed, the sign encoding is almost always 2's complement. In very rare cases, signed PCM audio is represented as a series of sign/magnitude coded numbers |
| resolution | This parameter specifies the amount of data used to represent each discrete amplitude sample. The most common values are 8 bits (1 byte), which gives a range of 256 amplitude steps, or 16 bits (2 bytes), which gives a range of 65536 amplitude steps. Other sizes, such as 12, 20, and 24 bits, are occasionally seen. Some king-sized formats even opt for 32 and 64 bits per sample |
| Signed | Specifies that the raw audio data coming from the microphone sensor is stored as signed numbers. |
| Unsigned | Specifies that the raw audio data coming from the microphone sensor is stored as unsigned numbers. |
| bigEndian | It specifies that the audio data is stored in the Big Endian format: the most significant byte of a word in the smallest address and the least significant byte is stored in the largest address |
| littleEndian | It specifies that the audio data is stored in the Little Endian format: the least significant byte in the smallest address. |

### Examples

In this example, the orientation has (60, 30, 120), the location is (23.215, 33.971), and altitude is 122.2 meter. The sampling rate is 44.1kHz and the byte order follows the little-endian method. It is signed data with a resolution 8 bits.

|  |
| --- |
| {  “sensedInfoBaseAttributes”: {},  “microphoneSensorType”: {  “microphoneOrientation”: {  “orientation”: [60, 30, 120]  },  “microphoneLocation”: {  “longitude”: 23.215,  “latitude”: 33.971  },  “microphoneAltitude”: {  “altitude”: 122.2,  “unit”: “meter”  }  “audioData”: {  “sample\_rate”: 44.1,  “byte\_order”: “littleEndian”,  “sign”: “signed”,  “resolution”: 8  }  }  } |

## Color camera sensor

### General

This sub-clause specifies a sensor data type, which describes the color of the captured objects in the environment, encoded as a sequence of pixel values.

### Syntax

|  |
| --- |
| "colorCameraSensorType": {  "type": "object",  "properties": {  "cameraSensor": {  "$ref": "#/definitions/cameraSensorType"  },  "rawVideo": {  "$ref": "#/definitions/rawVideoType"  }  }  },  "rawVideoType": {  "type": "object",  "properties": {  "videoData16": {  "type": "string"  },  "videoData64": {  "type": "string"  },  "width": {  "type": "integer"  },  "height": {  "type": "integer"  },  "bit\_depth": {  "type": "integer"  },  "stride": {  "type": "integer"  },  "coding4CC": {  "type": "integer"  },  "fps": {  "type": "integer"  },  "use\_frame\_packing": {  "type": "boolean"  },  "frame\_packing": {  "type": "integer"  }  }  },  "cameraSensorType": {  "type": "object",  "properties": {  "cameraOrientation": {  "$ref": "#/definitions/orientationSensorType"  },  "cameraLocation": {  "$ref": "#/definitions/globalPositionSensorType"  },  "cameraAltitude": {  "$ref": "#/definitions/altitudeSensorType"  },  "focalLength": {  "type": "number"  },  "aperture": {  "type": "number"  },  "shutterSpeed": {  "type": "number"  },  "filter": {  "type": "string",  "enum": [  "UV",  "Polarizing",  "NB",  "Diffusion",  "Star"  ]  }  },  "additionalProperties": false  }, |

### Semantics

Semantics of the colorCameraSensorType:

| *Name* | *Definition* |
| --- | --- |
| colorCameraSensorType | Tool for describing sensed information with respect to a color camera sensor. |
| rawVideo | Describes the sensed raw video data by the color camera sensor. |
| rawVideoType | Tool for describing raw video data for color camera sensor. |
| width | Width of the video in number of pixels. |
| height | Height of the video in number of pixels. |
| bit\_depth | Number of bits for each channel sample from the set of permitted values as defined by coding4CC. |
| stride | Size in bytes of one horizontal line. |
| coding4CC | A 4 character code representing the parameters of the raw data as specified by MP4RA. |
| fps | Frames per second of the video stream; if 0 then the frame rate is not known or variable. |
| use\_frame\_packing | This indicates if a frame contains two or more views. |
| frame\_packing | Frame Packing as Coding Independent Code Points. |
| videoData16 | Holds binary video data encoded as a textual string in base-16 format. |
| videoData64 | Holds binary video data encoded as a textual string in base-64 format. |
| rawVideoDataSize | Describes the size of the rawVideoData in bytes. This field is only present in binary representation. |
| rawVideoData | Actual data holder for binary raw video data, only in binary representation. The size of this field is given in rawVideoDataSize field. |
| cameraSensorType | Tool for describing sensed information with respect to a camera sensor. |
| cameraLocation | Describes the location of a camera using the structure defined by globalPositionSensorType. |
| cameraOrientation | Describes the orientation of a camera using the structure defined by orientationSensorType. |
| cameraAltitude | Describes the altitude of a camera using the structure defined by altitudeSensorType. |
| focalLength | Describes the distance between the lens and the image sensor when the subject is in focus, in terms of millimeters (mm). |
| aperture | Describes the diameter of the lens opening. It is expressed as F-stop, e.g. F2.8. It may also be expressed as f-number notation such as f/2.8. |
| shutterSpeed | Describes the time that the shutter remains open when taking a photograph in terms of seconds (sec). |
| filter | Describes kinds of camera filters. |

### Examples

In this example, the camera orientation values are yaw, pitch, and roll values of 60, 30, and 120, respectively. The camera altitude is 123.21 meter and the camera location has a longitude of 23.215 and a latitude of 33.971. The flocal length is 55mm and the aperture is f/2.8. The shutter speed is 0.008sec and the filter indicates that a UV filter is used.

The videoData16 is base-16 encoded data starting with "0314BA3827CFF2938...". The width and height of the image are 640 and 480 pixels, respectively. The bit\_depth is 8 and the stride is 10. The coding4CC is 2 and the fps of the image is 30 frames/sec. The use\_frame\_packing is “true” and the frame\_packing is 10.

|  |
| --- |
| {  “sensedInfoBaseAttributes”: {},  “colorCameraSensorType”: {  “cameraSensor”: {  “cameraOrientation”: {  “orientation”: [60, 30, 120]  },  “cameraLocation”: {  “longitude”: 23.215,  “latitude”: 33.971  },  “microphoneAltitude”: {  “altitude”: 123.1,  “unit”: “meter”  }  “focalLength”: 55,  “aperture”: 2.8,  “shutterSpeed”: 0.008,  “filter”: “UV”  },  “rawVideo”: {  “videoData16”: “0314BA3827CFF2938...”,  “width”: 640,  “height”: 480,  “bit\_depth”: 8,  “stride”: 10,  “coding4CC”: 2,  “fps”: 30,  “use\_frame\_packing”: true,  “frame\_packing”: 10  }  }  } |