**Text

Description automatically generated** **ISO/IEC JTC 1/SC 29/WG 7 N449**

**ISO/IEC JTC 1/SC 29/WG 7  
MPEG 3D Graphics and Haptics Coding   
Convenorship: AFNOR (France)**

**Document type:** Output Document

**Title:** Text for Committee Draft of ISO/IEC 23090-31: Haptics Coding

**Status:** Approved

**Date of document:** 2022-10-27

**Source:** ISO/IEC JTC 1/SC 29/WG 7

**Expected action:** None

**Action due date:** None

**No. of pages:** 167 (with cover page)

**Email of Convenor:** marius.preda @ imt . fr

**Committee URL:** [https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg7](https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3)

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 7 MPEG 3D GRAPHICS AND HAPTICS CODING**

**ISO/IEC JTC 1/SC 29/WG 7 N449**

**October 2022, Mainz**

|  |  |
| --- | --- |
| **Title** | **Text for Committee Draft of ISO/IEC 23090-31: Haptics Coding** |
| **Source** | **WG 7, MPEG 3D Graphics and Haptics Coding** |
| **Status** | **Approved** |
| **Serial Number** | **22111** |

**Abstract**

This document contains a Committee Draft on the Coded Representation of Haptics - Phase 1. The Coded Representation is developed around a JSON-based *.hjif* human-readable exchange format that can be compressed into a .*hmpg* binary file format or into a streamable bitstream. The document describes the architecture and usage of the encoder, the decoder, and the synthesizer. Following the description of the architecture, the *.hmpg*, *.hjif* and the streaming formats are precisely defined and the complete processing model of the codec is detailed.

**Contents**

[1 Scope 1](#_Toc119062883)

[2 Mnemonics 1](#_Toc119062884)

[3 Terms, definitions and abbreviated terms 2](#_Toc119062886)

[3.1 Terms and definitions 2](#_Toc119062887)

[3.2 Abbreviated terms 3](#_Toc119062889)

[4 Normative references 4](#_Toc119062892)

[5 Overview and architecture 5](#_Toc119062893)

[5.1 Overview 5](#_Toc119062894)

[5.2 Codec architecture 5](#_Toc119062896)

[6 Data model 6](#_Toc119062897)

[6.1 Data structure overview 6](#_Toc119062898)

[6.2 Haptic experience 7](#_Toc119062899)

[6.3 Haptic avatar 8](#_Toc119062900)

[6.4 Haptic perception 9](#_Toc119062901)

[6.5 Haptic device 11](#_Toc119062902)

[6.6 Haptic channel 12](#_Toc119062903)

[6.7 Haptic vector 18](#_Toc119062904)

[6.8 Haptic band 18](#_Toc119062905)

[6.9 Haptic effect 21](#_Toc119062906)

[6.10 Haptic keyframe 22](#_Toc119062907)

[7 Interchange file format 24](#_Toc119062908)

[7.1 Overview 24](#_Toc119062909)

[7.2 HJIF Specifications 25](#_Toc119062910)

[7.2.1 MPEG\_haptics 26](#_Toc119062911)

[7.2.2 MPEG\_haptics.avatar 27](#_Toc119062912)

[7.2.3 MPEG\_haptics.perception 27](#_Toc119062914)

[7.2.4 MPEG\_haptics.reference\_device 28](#_Toc119062915)

[7.2.5 MPEG\_haptics.channel 29](#_Toc119062916)

[7.2.6 MPEG\_haptics.vector 31](#_Toc119062917)

[7.2.7 MPEG\_haptics.band 31](#_Toc119062919)

[7.2.8 MPEG\_haptics.effect 32](#_Toc119062920)

[7.2.9 MPEG\_haptics.keyframe 33](#_Toc119062921)

[8 Compressed binary file format 34](#_Toc119062922)

[8.1 Binary header 34](#_Toc119062923)

[8.1.1 MPEG\_haptics metadata 34](#_Toc119062924)

[8.1.2 MPEG\_haptics\_avatar metadata 35](#_Toc119062925)

[8.1.3 MPEG\_haptics\_perception metadata 36](#_Toc119062926)

[8.1.4 MPEG\_haptics\_libraryEffect metadata 37](#_Toc119062927)

[8.1.5 MPEG\_haptics\_referenceDevice metadata 39](#_Toc119062928)

[8.1.6 MPEG\_haptics\_channel metadata 41](#_Toc119062929)

[8.2 Binary payload 43](#_Toc119062930)

[8.2.1 MPEG\_haptics\_band 43](#_Toc119062931)

[8.2.2 MPEG\_Haptics\_compositeEffect 45](#_Toc119062932)

[8.2.3 MPEG\_Haptics\_transientEffect 46](#_Toc119062933)

[8.2.4 MPEG\_Haptics\_curveEffect 47](#_Toc119062934)

[8.2.5 MPEG\_Haptics\_vectorialEffect 47](#_Toc119062935)

[8.2.6 MPEG\_haptics\_waveletEffect 48](#_Toc119062936)

[9 MPEG-I haptic stream (MIHS) format 51](#_Toc119062937)

[9.1 Overview 51](#_Toc119062938)

[9.1.1 Initialization units 52](#_Toc119062940)

[9.1.2 Temporal and spatial units 52](#_Toc119062941)

[9.1.3 MIHS packets 53](#_Toc119062942)

[9.2 Syntax and semantics 54](#_Toc119062943)

[9.2.1 mpegiHapticStream() 54](#_Toc119062944)

[9.2.2 mpegiHapticUnit() 54](#_Toc119062945)

[9.2.3 mpegiHapticPacket() 55](#_Toc119062947)

[9.2.4 MIHSPacketPayload() 56](#_Toc119062949)

[9.2.5 readMetadataTiming() 57](#_Toc119062951)

[9.2.6 readMetadataExperience() 57](#_Toc119062953)

[9.2.7 readAvatar() 58](#_Toc119062955)

[9.2.8 readMetadataPerception() 58](#_Toc119062957)

[9.2.9 readReferenceDevice() 60](#_Toc119062959)

[9.2.10 readMetadataChannel() 62](#_Toc119062960)

[9.2.11 readMetadataBand() 63](#_Toc119062962)

[9.2.12 readLibrary() 65](#_Toc119062964)

[9.2.13 readLibraryEffect() 65](#_Toc119062966)

[9.2.14 readData() 66](#_Toc119062968)

[9.2.15 readEffect() 67](#_Toc119062970)

[9.2.16 readEffectBasis() 67](#_Toc119062972)

[9.2.17 readKeyframe() 68](#_Toc119062974)

[9.2.18 readTransientKeyframe() 69](#_Toc119062976)

[9.2.19 readCurveKeyframe() 69](#_Toc119062978)

[9.2.20 readVectorialKeyFrame() 69](#_Toc119062980)

[9.2.21 readWaveletEffect() 70](#_Toc119062982)

[9.2.22 readCRC() 70](#_Toc119062984)

[9.3 Description of MIHSPacketType 71](#_Toc119062986)

[9.3.1 Timing 71](#_Toc119062987)

[9.3.2 MetadataHaptics 71](#_Toc119062988)

[9.3.3 MetadataPerception 71](#_Toc119062990)

[9.3.4 MetadataChannel 72](#_Toc119062994)

[9.3.5 MetadataBand 72](#_Toc119062995)

[9.3.6 LibraryEffect 73](#_Toc119062997)

[9.3.7 Data 74](#_Toc119062999)

[9.3.8 CRC16 & CRC32 75](#_Toc119063000)

[9.3.9 GlobalCRC16 & GlobalCRC32 75](#_Toc119063001)

[9.4 Application examples 75](#_Toc119063002)

[9.4.1 Initialization units 75](#_Toc119063003)

[9.4.2 Temporal and spatial units 75](#_Toc119063004)

[9.4.3 Silent units 76](#_Toc119063005)

[9.4.4 CRC packets 76](#_Toc119063006)

[9.5 Random access support with MIHS (informative) 77](#_Toc119063008)

[10 Processing Model 78](#_Toc119063009)

[10.1 Encoder (informative) 78](#_Toc119063010)

[10.1.1 Encoder Architecture 78](#_Toc119063011)

[10.1.2 OHM metadata input file 78](#_Toc119063012)

[10.1.3 Descriptive input files 79](#_Toc119063013)

[10.1.4 PCM input file 79](#_Toc119063014)

[10.1.5 Transcoding descriptive content 80](#_Toc119063015)

[10.1.5.1 IVS 80](#_Toc119063016)

[10.1.5.2 AHAP 82](#_Toc119063017)

[10.1.6 Frequency Band decomposition 82](#_Toc119063031)

[10.1.7 Keyframe extraction for low frequencies processing 83](#_Toc119063032)

[10.1.8 Wavelet encoding 84](#_Toc119063033)

[10.2 Decoder 95](#_Toc119063036)

[10.2.1 Decoder Architecture 95](#_Toc119063037)

[10.2.2 Metadata and parametric data decoding 96](#_Toc119063038)

[10.2.3 Wavelet Decoding 96](#_Toc119063039)

[10.3 Synthesizer (informative) 101](#_Toc119063040)

[Annex A JSON schema reference (normative) 103](#_Toc119063041)

[Annex B OHM metadata input file format (informative) 104](#_Toc119063042)

[Annex C Semantic body part and actuator mapping (informative) 105](#_Toc119063043)

[C.1 Semantic Body parts 105](#_Toc119063045)

[C.2 Actuator mapping 106](#_Toc119063046)

[C.3 Up/down-sampling actuator resolution 107](#_Toc119063047)

[Bibliography 109](#_Toc119063048)

**Foreword**

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see [www.iso.org/directives](https://www.iso.org/directives-and-policies.html)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](https://www.iso.org/iso-standards-and-patents.html)).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see [www.iso.org/iso/foreword.html](https://www.iso.org/foreword-supplementary-information.html).

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 23094 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at [www.iso.org/members.html](https://www.iso.org/members.html).

**Introduction**

Haptics provide an additional layer of entertainment and sensory immersion to the user. Therefore, the user experience and enjoyment of media content, be it in ISOBMFF files or streams such as ATSC 3.0 broadcasts, streaming games, and mobile advertisements can be significantly enhanced by the judicious addition of haptics to the audio/video content. To that end, haptics has been proposed as a first-order media type, akin to audio and video, in ISOBMFF. Further, haptics has also been proposed as an addition to the MPEG-DASH standard to signal the presence of haptics in the MP4 segments to the DASH streaming clients. Lastly, the MPEG-I Phase 2 use cases have been augmented with haptics [1] resulting in a set of haptic-specific requirements for MPEG-I Phase 2 [2]. All these proposals are in various stages of the MPEG standardization process.

Haptics digital encoding is the storing of tactile data in a digital format. As with audio and video, digital encoding is of fundamental importance to allow digital haptic devices to function. Haptics encoding gained relevance in the last years with the increased market importance of wideband haptics in consumer peripherals such as the iPhone with the Taptic Engine, and the PlayStation 5 with the DualSense controller. The prior generation of haptics peripherals was based on less expressive haptic actuators usually based on state machine control processes.

The current market and technological landscape are highly fragmented with a large spectrum of encoding formats developed and used by various technology manufacturers. This results in several cross-compatibility and interoperability problems that are preventing wider adoption of haptics in the marketplace. Further, the ongoing haptics standardization efforts in MPEG highlight the need for standardizing a coded representation of haptics. A standard haptics coding format (and associated standardized decoder) will facilitate the incorporation of haptics into the ISOBMFF, MPEG-DASH, and MPEG-I standards, making it easier for content creators as well as media/streaming content providers to incorporate haptics in *a standardized manner* and improve the overall user experience.

In the field of haptics, the signal encoding usually takes two different approaches:

**Quantized**: This representation is generally made from measured data. The samples from the original phenomenon are stored at a specific acquisition frequency inside the file to represent this signal. One example of a quantized haptic signal is proposed through WAV files, originally developed for audio. WAV file formalism allows the capture of real-world data and represents complex wideband haptic feedback. This type of haptics encoding has the disadvantage of being difficult to modify once encoded due to the inability to access the primitives used to create the signal.

**Descriptive**: This representation is used to encode haptic signals as a combination of functions to be synthesised; existing vectorized formats are AHAP and IVS from Apple and Immersion Corporation, respectively. These formats have the advantages of being created with a composition of primitives. They are easily modifiable in runtime by an application and by dedicated programs. Currently, these solutions support only vibrotactile perception, other forms of haptics such as kinaesthetic, temperature, textures and others are not supported. They also tend to be memory inefficient with increasing signal complexity and cannot encode non-periodic phenomenon.

This document describes the coded representation allowing to encode both descriptive and quantized data in a human readable JSON format (*.hjif*) used as an exchange format. This format can be compressed into a binary file format for distribution (*.hmpg*) or into a packetized bitstream for streaming purposes.

**Information technology – Coded representation of immersive media - Part 31: Haptics coding**

# Scope

This part of ISO/IEC 23090-31 specifies technology that supports the efficient transmission and rendering of haptic signals for the playback of immersive experiences in a wide variety of scenarios. The document describes in detail a robust coded representation of haptic media covering the two most popular haptics perception leveraged by devices today: vibrotactile and kinaesthetic. Support for other haptic modalities has also been integrated.

The coded representation allows to encode both descriptive and quantized data in a human readable JSON format used for exchange purposes and a compressed bitstream version, optimised for memory usage and distribution purposes. This approach allows to meet the expectations for compatibility with both descriptive and quantized formats, as required by the market, as well as interoperability between devices for both 3D immersive experiences and distribution purposes.

Information provided in this document related to the decoder is normative, while information related to the encoder and the renderer are informative.

# Mnemonics

The following data types are used for the generated data:

|  |  |
| --- | --- |
| bslbf | Bit string left bit first, where “left” is the order in which bit strings are written. Bit strings are written as a string of 1s and 0s within single quite marks, for example ‘10000001’. |
| string | String data is stored as a character array encoded in UTF-8. |
| uimsbf | Unsigned integer most significant bit first. |
| imsbf | Integer most significant bit first. |
| vlclbf | Variable length code left bit first, where “left” refers to the order in which the variable length codes are written. |
| decimal | Decimal number encoded as uimsbf with a given range. Default range is [-1;1] |
| islif | Integer stream left integer first, where “left” is the order in which integer strings are written. Only for buffering. |
| vlislif | Variable length integer stream left integer first, where “left” refers to the order in which the integers are written. |

# Terms, definitions and abbreviated terms

# Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at <https://www.iso.org/obp>

— IEC Electropedia: available at <https://www.electropedia.org/>

* + 1. actuator

component of a device for rendering haptic sensation

* + 1. avatar

body (or part of body) representation

* + 1. band

component in a channel for containing effects for a specific range of frequencies

* + 1. channel

component in a perception containing one or more bands rendered on a device at a specific body location

* + 1. device

physical system having one or more actuators configured to render haptic sensation corresponding with a given signal

* + 1. effect

component of a band for defining a signal, consist of a haptic waveform, or one or more haptic keyframes

* + 1. experience

top level haptic component containing perceptions and metadata

* + 1. haptics

tactile sensations

* + 1. keyframe

component of an effect mapping a position in time or space to an effect parameter such as amplitude or frequency

* + 1. metadata

global information about an experience, perception, channel, or band

* + 1. MIHS format

self-contained stream for transporting MPEG-I haptic data

* + 1. MIHS initialization unit

MIHS unit containing metadata necessary to reset and initialize a haptic decoder

* + 1. MIHS packet

MIHS data packet which includes metadata or binary effect data

* + 1. MIHS unit

MIHS data unit covering a duration of time

* + 1. modality

type of haptics, such as vibration, force, pressure, position, velocity, or temperature

* + 1. perception

haptic perception containing channels of a specific modality

* + 1. signal

representation of the haptics associated with a specific modality to be rendered on a device

3.1.17   
vector

direction in space which can be used for haptic signals spatialization

# Abbreviated terms

|  |  |
| --- | --- |
| AC | Arithmetic coding/coder |
| AHAP | Apple Haptic and Audio Pattern - JSON-like file format that specifies a haptic pattern |
| ATSC | Advanced Television Systems Committee |
| CDF9/7 | Cohen–Daubechies–Feauveau 9/7 |
| CRC | Cyclic redundancy check |
| DASH | Dynamic Adaptive Streaming over HTTP |
| FFT | Fast Fourier Transform |
| HJIF | Haptics JSON Interchange Format |
| IEC | International Electrotechnical Commission |
| ISO | International Organization for Standardization |
| ISOBMFF | ISO Base media file format (specified in ISO/IEC 14496-12) |
| IVS | Binary file format for representing haptic effects |
| JSON | JavaScript Object Notation |
| LOD | Level of detail |
| MIHS | MPEG-I haptic stream |
| MPEG | Moving Pictures Expert Group |
| MPEG-I | MPEG immersive media |
| OHM | Object Haptic Metadata - Text file format for haptics metadata |
| PCM | Pulse-code modulation |
| SPIHT | Set partitioning in hierarchical trees |

# Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

RFC8259, Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

RFC3986, Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.

ISO 8601-1:2019 - Date and time — Representations for information interchange — Part 1: Basic rules

ISO 8601-2:2019 - Date and time — Representations for information interchange — Part 2: Extensions

# Overview and architecture

# Overview

This document describes a coded representation of haptics based on a data model that enables the encoding of both descriptive and quantized haptic data. Three complementary formats based on this shared data model (clause 6) are detailed in the specifications: an interchange format (*.hjif*) detailed in clause 7, a binary compressed format (*.hmpg*) detailed in clause 8 and a streaming format detailed in clause 9. In addition, a normative decoder and an informative encoder and synthesizer are also described in detail.

The *.hjif* format is a human-readable format based on JSON and is not optimized for memory usage. It can easily be parsed and manually edited which makes it an ideal interchange format, especially when designing/creating content. For distribution purposes, the *.hjif* data can be compressed into a more memory efficient binary *.hmpg* bitstream. This compression is lossy, with different parameters impacting the encoding depth of amplitude and frequencies composing the bitstream. For streaming purposes, the data can be compressed and packetized into a MPEG-I haptic stream (MIHS).

# Codec architecture

The codec architecture can process both waveform PCM signals and descriptive haptic files such as AHAP from Apple, IVS from Immersion, or HJIF the proposed MPEG format. Metadata information is provided to the codec through OHM input files (described in Annex B). An overview of the codec architecture is depicted in Figure 1, and a more detailed description of the input files and encoder architecture are provided in clause 10.

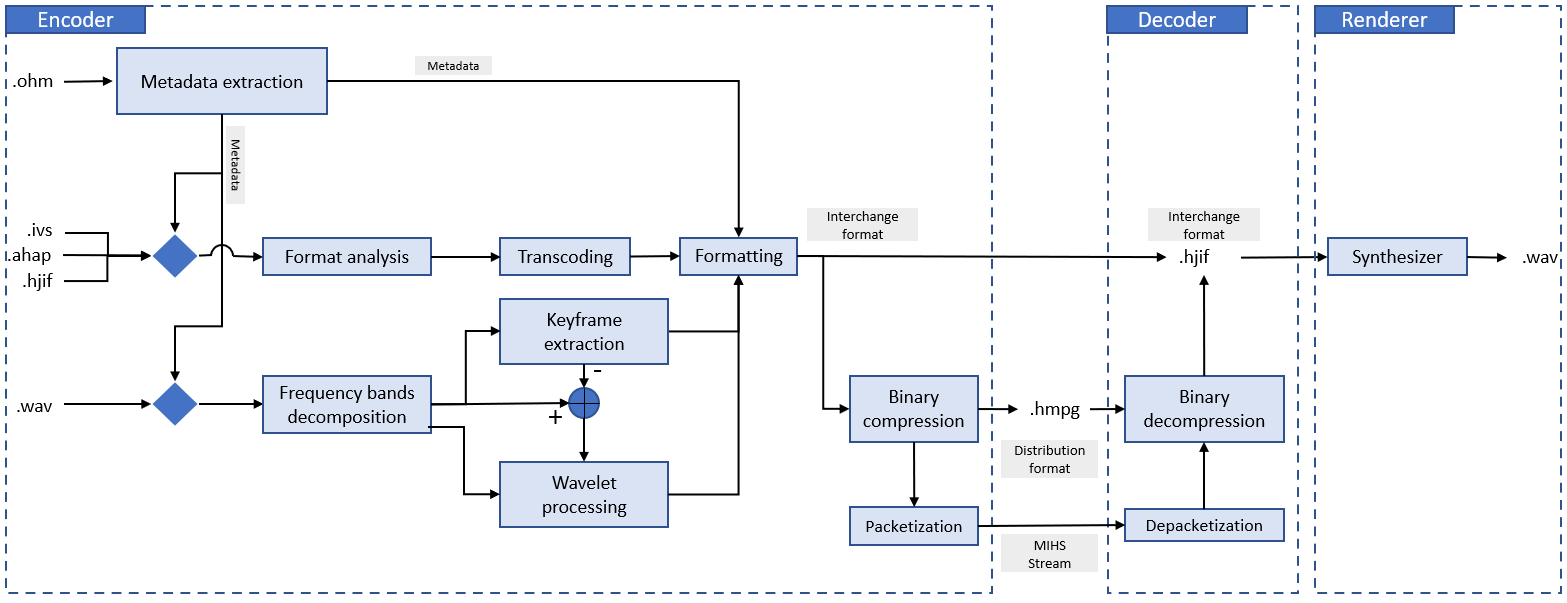


Figure 1: Overview of the codec architecture.

The encoder processes the two types of input files differently. For descriptive content, the input is analysed semantically to transcode (if necessary) the data into the proposed coded representation. For PCM content, the signal analysis process is split into two sub-processes. After performing a frequency band decomposition on the signal, low frequencies are encoded using a keyframe extraction process. The low frequency band(s) is then reconstructed and the error between this signal and the original low frequency signal is computed. This residual signal is then added to the original high frequency band(s), before encoding using Wavelet Transforms. The encoding process is detailed in clause 10.

The encoder can output three types of formats: an interchange file format (*.hjif*) encoded in a JSON readable form, a compressed format encoded as a binary file (*.hmpg*) and a streaming format defining a packetized bitstream. The three formats have complementary purposes and a lossy one-to-one conversion can be operated between them. It is informative and is detailed in clause 10.1

The decoder takes as input a binary *.hmpg* file and outputs a *.hjif* file. Haptic data contained in a *.hjif* file can then be rendered directly on haptic devices or using an intermediate synthesizer generating PCM data. It is normative and is detailed in clause 10.2.

The synthesizer allows to render haptic data from a *.hjif* input file into a PCM output file. It is informative and is detailed in clause 10.3.

# Data model

This clause focuses on the data model of the coded representation of haptics. It specifies the information required by a synthesizer to render the haptic data. The following sub-clauses provide detailed definitions for every property of each element of the data model depicted in Figure 2. The data structure introduced in this clause is shared by the interchange format, the binary format and the streaming format defined respectively in clauses 7, 8, and 9.

# Data structure overview

The data structure of the three formats follows the hierarchical organization illustrated in Figure 2 .

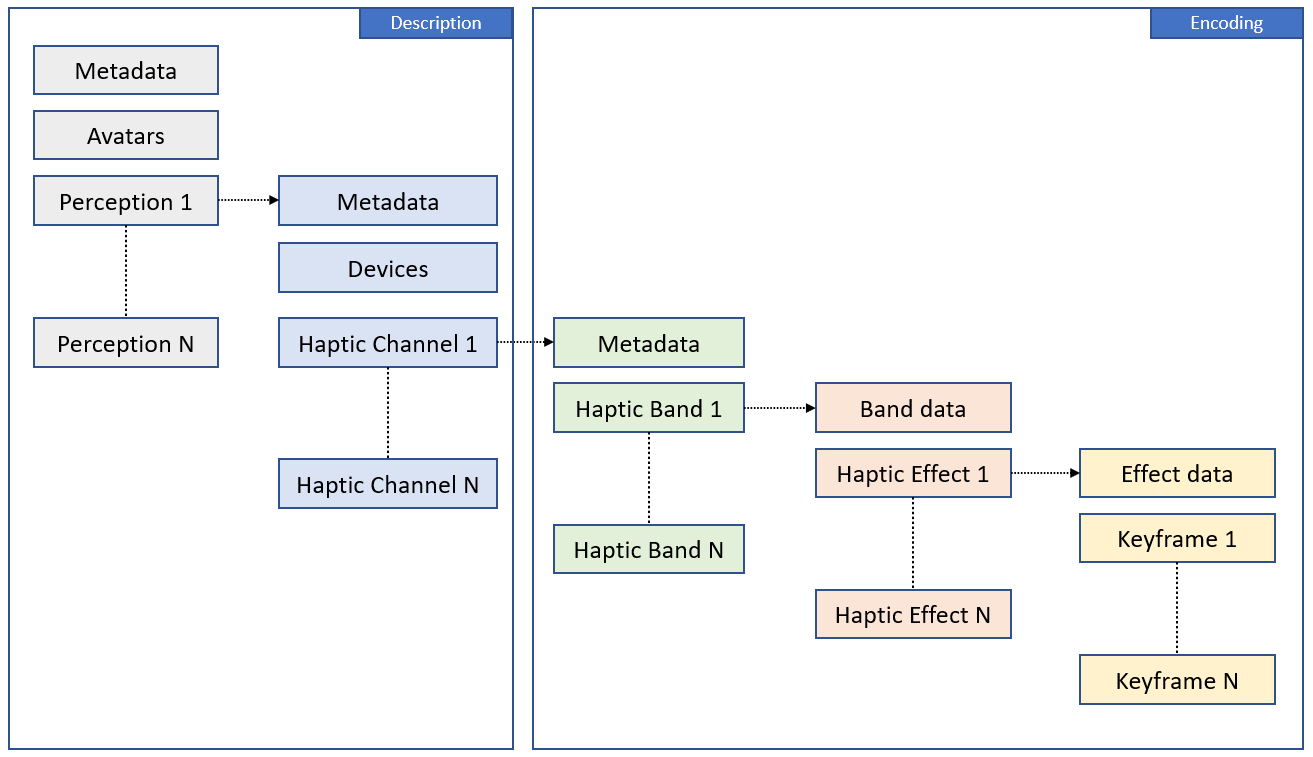


Figure 2: General data model.

The highest level of the structure describes the entire haptic experience defined in the file or stream. It contains some high-level metadata information. It also provides a list of avatars (i.e., body representation) referenced later in the file to specify the desired location of haptic stimuli on the body. Finally, the haptic data itself is described through a list of Perceptions. These perceptions correspond to haptic signals associated with specific perception modalities (e.g., Vibration, Force, Position, Velocity, Temperature, ...).

In addition to specific metadata, a perception contains a list of channels, in each of which the data is decomposed in frequency bands. Each band defines part of the signal in a given frequency range. A band is described by a list of haptic effects, each containing a list of keyframes. The haptic signal in a channel can then be reconstructed by combining the data in the different bands as illustrated in Figure 3. For instance, by adding the high and low frequency bands in this figure, the original signal can be reconstructed.

The Perception and Channel information describe the content (Description part of Figure 2), while the Bands, Effects and Keyframe contain the data of the encoded signals (Encoding part of Figure 2).

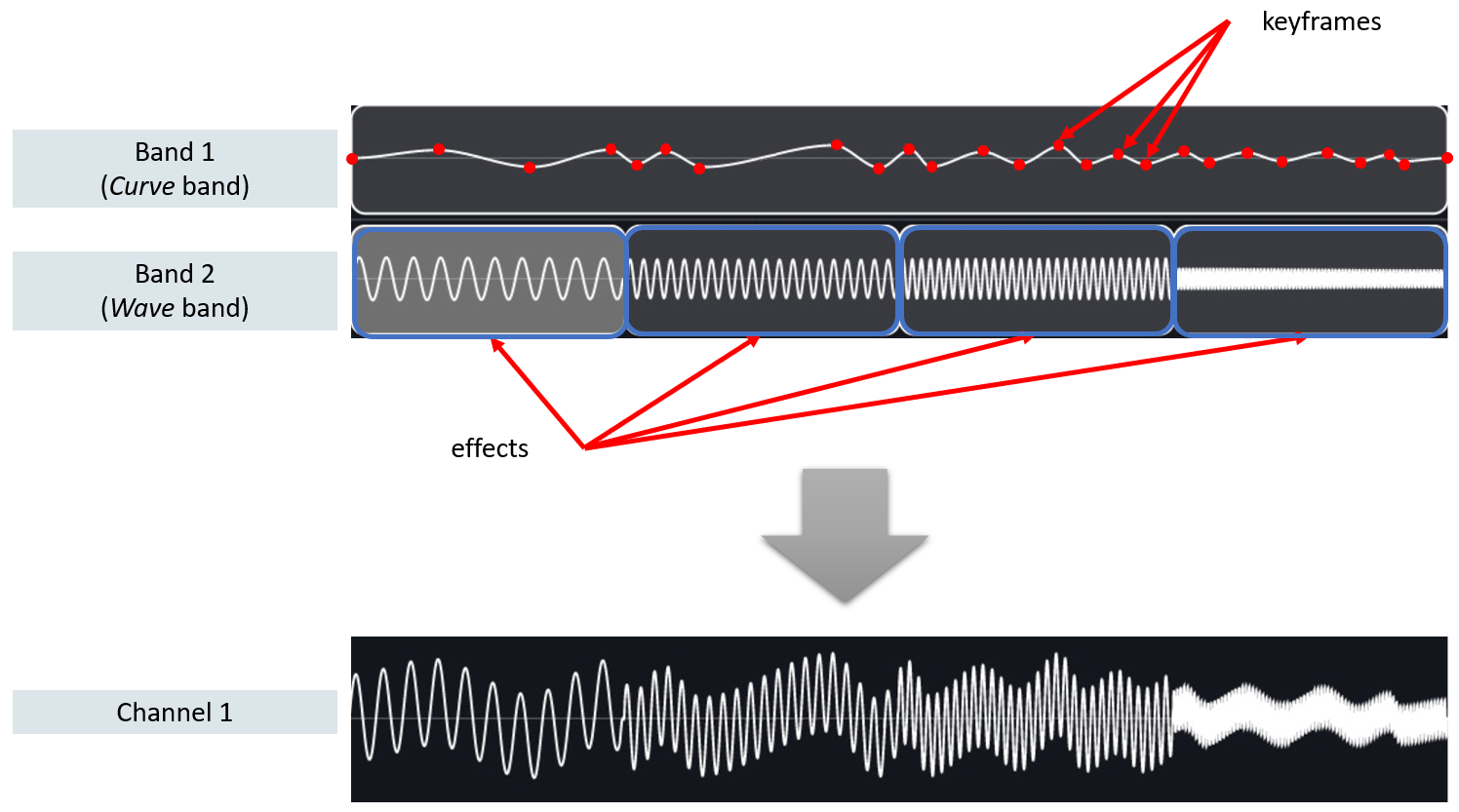


Figure 3: Haptic channel (bottom) and its decomposition in two frequency bands (top). The Bands consist of a set of effects described by keyframes.

The format proposes four types of Haptic Bands: Transient bands, Curve bands, Vectorial Wave bands and Wavelet Wave bands. Each Band is composed of a series of "Effects" of the same type as the band, each defined by a list of "Keyframes". The data contained in the effects and keyframes is interpreted differently for different types of haptic bands.

# Haptic experience

The haptic experience defines the root of the hierarchical data model. It provides information on the date of the file and the version of the format, it describes the haptic experience, it lists the different avatars (i.e., body representation) used throughout the experience and it defines all the haptic perceptions. Table 1 details the list of properties of a haptic experience.

Table 1: List of haptic properties of a haptic experience

|  |  |
| --- | --- |
| **Property** | **Description** |
| **version** | Version of this ISO/IEC 23090-31 specification. |
| **date** | Creation date of this Haptic experience. The date format shall follow the ISO 8601 standard. |
| **description** | Description of this haptic experience. |
| **avatars** | List of haptic avatars defining body representations used in the haptic experience. See clause 6.3. |
| **perceptions** | List of perception describing a haptic signal. See clause 6.4. |

# Haptic avatar

Haptic avatars are used as body representations. Haptic avatars define different types of avatars and allow to reference a custom 3D mesh from a companion file. Each haptic perception of the experience is associated with a haptic avatar, which allows spatialization of haptic effects at the haptic channel level. The same avatar can be used by multiple perceptions.

Using a 3D mesh allows to provide high resolution and accuracy with variable vertex density depending on the application. For instance, the density can be representative of the spatial acuity of a specific perception modality (Figure 4). The format of custom 3D meshes is out of the scope of this document, but traditional formats such as *obj* or *glTF* may be used.

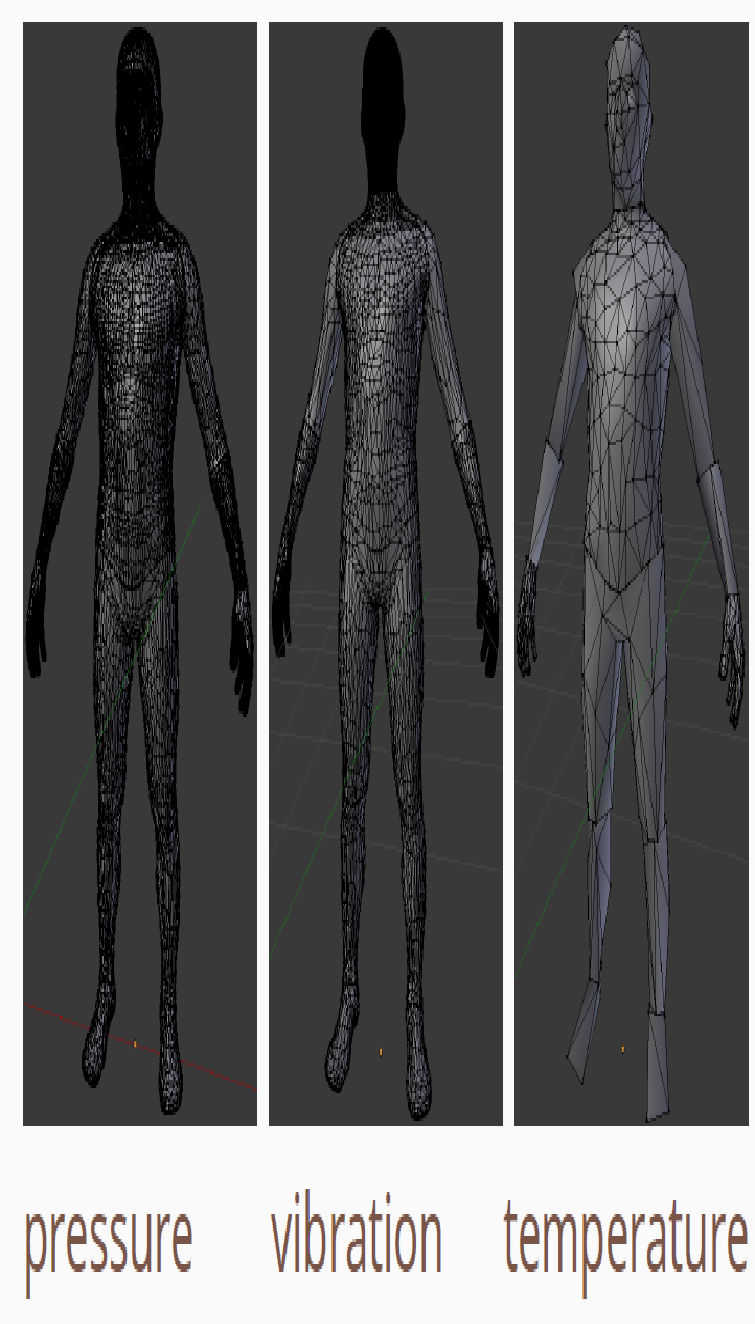


Figure 4: Example of haptic avatar body representations. The mesh resolution may be related to the haptic spatial acuity of the relevant haptic perception.

Table 2 defines the list of properties of a haptic avatar:

Table 2: List of properties of a haptic avatar

|  |  |
| --- | --- |
| **Property** | **Description** |
| **id** | Unique identifier of the avatar. |
| **lod** | If the avatar uses a mesh with several levels of detail (LODs), this indicates which LOD should be used for the avatar. |
| **type** | Type of haptic perception represented by the avatar. It is related to the spatial acuity of the corresponding haptic modality. The avatar type may be:   * Vibration: for an avatar representative for vibrotactile signals. * Pressure: for an avatar representative for pressure signals. * Temperature: for an avatar representative for temperature signals. * Custom: for a custom avatar representative for the application. The mesh is provided as a companion file using the mesh URI. The definition of custom mesh is out of of the scope of this document. |
| **mesh** | URI to access the associated 3D mesh file. The URI must follow the syntax defined in RFC3986 |

# Haptic perception

A haptic perception is a haptic signal associated with a specific perception modality. The format supports modalities encoded in function of the time (Pressure, Acceleration, Velocity, Position, Temperature, Vibrotactile, Water, Wind, Force, Electrotactile) or space (VibrotactileTexture, Stiffness and Friction). The list of supported modalities is provided in Table 4 with the corresponding units.

For each haptic perception, metadata information is provided on the modality, the corresponding avatar representation, and technical characteristics of targeted or compatible haptic devices.

The data associated to a perception may contain multiple channels. A channel is associated to a body location and usually corresponds to a haptic device. For instance: a vibrotactile suit with 16 channels corresponding to 16 "Vibrotactile” actuators or a gamepad with one “Force” feedback trigger.

A haptic perception also contains an effect library. Table 3 describes the full list of properties of a haptic perception.

Table 3: List of properties of a haptic perception

|  |  |
| --- | --- |
| **Property** | **Description** |
| **id** | Unique identifier of the perception. |
| **perception modality** | Type of perception modality of the haptic signal. The detailed list of perception modalities and the associated units is given in Table 4. |
| **description** | Description of the haptic data contained in the perception. |
| **avatar id** | Unique identifier of the associated avatar, defined in the data structure. |
| **effect library** | List of haptic effects as defined in clause 6.9. Effects from the library shall have an id and may be referenced directly in a Band (see clause 6.8). |
| **reference devices** | List of targeted haptic reference devices or actuators used for this haptic perception. More details on reference devices is given in clause 6.5. |
| **channels** | List of haptics channels composing this perception. More details on haptic channels is given in clause 6.6. |
| **unit exponent** | Exponent of the powers of 10 for the SI unit identifying the space of representation of the independent variable.  This property specifies which measurement unit is used to encode the given perception. By default, the considered value is -3.  For example, if the perception modality is set to vibration and this exponent is set to -3, the perception experience is encoded in milliseconds. |
| **perception unit exponent** | Exponent of the powers of 10 for the SI unit measure of the dependent variable. This property specifies which measurement unit is used to output the given perception. By default, the considered value is 0.  For example, if the perception modality is set to stiffness and this exponent is set to 0, the perception experience is encoded in Newton. |

Table 4: Perception modalities and corresponding units.

|  |  |  |
| --- | --- | --- |
| Modality | Perception unit | unit |
| Pressure | Pa | s |
| Acceleration | m/sˆ2 | s |
| Velocity | m/s | s |
| Position | m | s |
| Temperature | K | s |
| Vibrotactile | Normalized to – 1/1 | s |
| Water | mˆ3 | s |
| Wind | m/s | s |
| Force | N | s |
| Electrotactile | Normalized to -1/1 | s |
| Vibrotactile Texture | Normalized to -1/1 | m |
| Stiffness | N | m |
| Friction | Normalized to -1/1 | m |
| Other | Normalized to -1/1 | s |

**Haptic effect library**

The haptic library allows to define a set of shared haptic effects at the perception level. It consists of a list of effects that can be referenced at a band. This allows to avoid unnecessary repetition of identical effects. The effect library is particularly useful for content creators and avoids the duplication of effects inside the channels.

As illustrated in Figure 5, the effect library can be used to reduce the memory footprint of haptic data. The same experience can be stored in an optimized manner using a single haptic effect in the library referenced multiple times at the band level or by storing this effect multiple times directly at the band level.

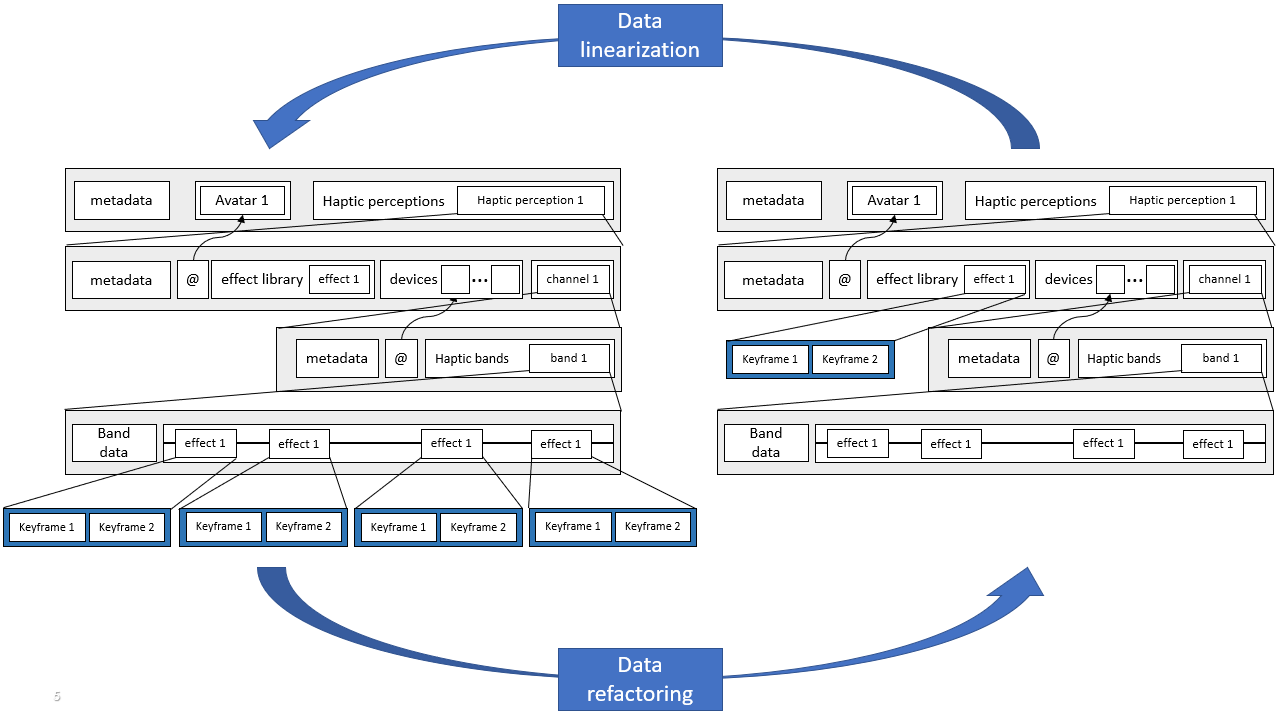


Figure 5: Example of haptic files containing the same data with and without using the effect library

# Haptic device

A haptic experience is usually defined with a reference setup to validate the experience or with a number of specific targeted haptic devices in mind (compatible devices). If the experience is played back on different devices with different capabilities, the associated encoded signal may have to be rendered differently. To perform such adaptation, the capabilities of the original device(s) (reference or compatible devices) must be known. For this purpose, each haptic perception defines a list of reference devices (with their detailed characteristics) and each haptic channel may reference the corresponding device.

A haptic reference device is described through a list of characteristics, including the type of device, the frequency range of the device, the maximum voltage of the device and many other properties. The detailed list of properties is specified in Table 5.

Table 5: List of properties for a haptic device

|  |  |
| --- | --- |
| **Property** | **Description** |
| **id** | Unique identifier of the device. |
| **name** | Name of the device. | |
| **body part mask** | Binary mask specifying the location of the device or actuator on the body based on the body segmentation proposed in clause 6.6 and detailed in Table 7. | |
| **maximum frequency** | Maximum frequency of the actuator (Hz). | |
| **minimum frequency** | Minimum frequency of the actuator (Hz). | |
| **resonance frequency** | Resonance frequency of the actuator (Hz). | |
| **maximum amplitude** | Maximum amplitude value of the targeted device according to the perception modality.  For instance, it relates to the maximum acceleration speed if the perception modality is the acceleration. The corresponding unit is specified in Table 4 in clause 6.4. | |
| **impedance** | Impedance of the actuator (Ω). | |
| **maximum voltage** | Maximum voltage of the actuator (V). | |
| **maximum current** | Maximum current of the actuator (A). | |
| **maximum displacement** | Maximum displacement of the actuator (mm). | |
| **weight** | Weight of the device (Kg). | |
| **size** | Indicates the size of the device (mm). | |
| **custom** | User-defined data. This parameter may be used to specify additional properties of the targeted device. | |
| **type** | Type of actuator. Possible types are:   * LRA, * VCA * ERM * Piezo * Unknown (for other modalities) |

# Haptic channel

Haptic signals can be encoded on multiple channels. Typically, a haptic channel defines a signal to be rendered at a specific body location with a dedicated actuator/device. Metadata stored at the channel level includes information such as the gain associated to the channel, the mixing weight, the desired body location of the haptic feedback and optionally the reference device and/or a direction. Additional information such as the desired sampling frequency or sample count can also be provided. Finally, the haptic data of a channel is contained in a set of haptic bands defined by their frequency range. The list of properties of a haptic channel is detailed in Table 6.

Table 6: List of properties of a haptic channel

|  |  |
| --- | --- |
| **Property** | **Description** |
| **id** | Unique identifier of the channel. |
| **description** | Description of the channel. | |
| **reference device id** | Targeted reference device from the list defined in the perception. | |
| **gain** | Gain associated with the channel to adapt the normalized encoded data values to a typical device, according to:  Where *x* corresponds to the normalized encoded data. | |
| **mixing weight** | Weight of the channel when mixing different channels together to produce a mixed signal. The resulting signal is given by:  Where *Vi* corresponds to the signal of channel *i*. A mixing weight of 0 indicates that the channel is not mixed. | |
| **body part mask** | Binary mask specifying the location of the effect on the body as defined in Table 7.  The binary mask *0x0* indicates that the body part is not specified. The application can render the effect anywhere. The mask *0xFFFFFFFF* corresponds to the full body. It means that the effect is applied on the whole body. For instance, it may be used for background effects such as the impact of an explosion. | |
| **actuator resolution** | Reference actuator resolution used to design the haptic experience. This value linked to **body part target** and **actuator target** can be used together as an experience spatialization model on the human body. | |
| **body part target** | Identification of a unique and/or a group of body parts on the human body semantically. Table 9 describes all the possible values which can be stored here to construct the targeting command. | |
| **actuator target** | List of different coordinates to target actuators on the previously identified human body parts. | |
| **frequency sampling** | Sampling frequency of the original encoded signal (Hz).  This may be used by the synthesizer to reconstruct the original signal. However, the synthesizer may sample the output signal at another sampling frequency. | |
| **sample count** | Present if the frequency sampling value is greater than 0. It is the number of samples of the original encoded signal. This can be used along with the frequency sampling by the synthesizer to ensure that the output signal has the same size and duration as the original file. | |
| **vertices** | List of the vertices from the avatar impacted by the effect. More precisely, it is the list of indices of the vertices from the mesh associated to the avatar of the perception. If the avatar does not specify a mesh, this field should be ignored. The vertices impacted by the effects of this channel are the body locations where the effects should be applied.  The appropriate avatar representation is referenced by the avatar id indicated at the perception level (see 6.4). | |
| **bands** | List of haptic bands composing the channel. A channel can include one or several bands. A band corresponds to a frequency bandwidth as specified in clause 6.8.  If the bands array is empty, it corresponds to a channel without any haptic effect.  As illustrated in Figure 3, the haptic signal of a channel is the sum of the signals in each band. | |
| **direction** | Specifies a spatial direction for the channel as detailed in clause 6.7. This direction metadata should only be used with haptic modalities dependant on the space dimension (i.e. *Vibrotactile Texture*, *Stiffness* and *Friction*). It indicates a preferred rendering direction of the haptic perception of the targeted body part. It can be composed with X, Y and Z following the formalism for unit vectors to indicate any direction in the 3D space. Each integer value stored in this vector will be transformed from its initial range [-127; 127] to the [-1; 1] range to interpret this vector as unitary. |

The Haptic Direction identify the preferred rendering direction of the spatialized haptics experience. The direction is related to the relative position of the body part in its own coordinate system. The information is required to specify the rendering of a haptic effect in a predetermined direction.

The spatialization on the body of the haptic stimuli associated with a channel can de specified either by i) referencing a set of vertices (with the ***vertices*** property) from a custom avatar mesh (Figure 4), ii) using a body part mask based on the body segmentation illustrated in Figure 6 with 32 defined parts, or iii) using a semantic body parts identification combined with a map of actuators on these body parts. Multiple methods can be used combined.

**Custom mesh avatar**

Consists of a set of 3D vertices, with their 3D coordinates.

**Body part mask**

The mask has been defined based on Figure 6 segmentation of the human body.

Diagram

Description automatically generated

Figure 6: Body part segmentation

The 32 body parts and their corresponding masks are detailed in Table 7. These binary masks can be combined to associate haptics stimuli to multiple body parts. Table 8 gives some common body parts combinations.

Table 7: Body part masks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Name | body\_part\_mask | Hexa | Value |
| 0 | Unspecified | 00000000000000000000000000000000 | 0x00000000 | 0 |
| 1 | Head front | 00000000000000000000000000000001 | 0x00000001 | 1 |
| 2 | Head back | 00000000000000000000000000000010 | 0x00000002 | 2 |
| 3 | Head right | 00000000000000000000000000000100 | 0x00000004 | 4 |
| 4 | Head left | 00000000000000000000000000001000 | 0x00000008 | 8 |
| 5 | Right upper chest | 00000000000000000000000000010000 | 0x00000010 | 16 |
| 6 | Left upper chest | 00000000000000000000000000100000 | 0x00000020 | 32 |
| 7 | Abdomen | 00000000000000000000000001000000 | 0x00000040 | 64 |
| 8 | Waist | 00000000000000000000000010000000 | 0x00000080 | 128 |
| 9 | Upper back | 00000000000000000000000100000000 | 0x00000100 | 256 |
| 10 | Lower back | 00000000000000000000001000000000 | 0x00000200 | 512 |
| 11 | Right upper arm | 00000000000000000000010000000000 | 0x00000400 | 1 024 |
| 12 | Left upper arm | 00000000000000000000100000000000 | 0x00000800 | 2 048 |
| 13 | Right forearm | 00000000000000000001000000000000 | 0x00001000 | 4 096 |
| 14 | Left forearm | 00000000000000000010000000000000 | 0x00002000 | 8 192 |
| 15 | Right wrist | 00000000000000000100000000000000 | 0x00004000 | 16 384 |
| 16 | Left wrist | 00000000000000001000000000000000 | 0x00008000 | 32 768 |
| 17 | Right hand palm | 00000000000000010000000000000000 | 0x00010000 | 65 536 |
| 18 | Left hand palm | 00000000000000100000000000000000 | 0x00020000 | 131 072 |
| 19 | Right hand dorsum | 00000000000001000000000000000000 | 0x00040000 | 262 144 |
| 20 | Left hand dorsum | 00000000000010000000000000000000 | 0x00080000 | 524 288 |
| 21 | Right hand fingers | 00000000000100000000000000000000 | 0x00100000 | 1 048 576 |
| 22 | Left hand fingers | 00000000001000000000000000000000 | 0x00200000 | 2 097 152 |
| 23 | Right thigh | 00000000010000000000000000000000 | 0x00400000 | 4 194 304 |
| 24 | Left thigh | 00000000100000000000000000000000 | 0x00800000 | 8 388 608 |
| 25 | Right calf | 00000001000000000000000000000000 | 0x01000000 | 16 777 216 |
| 26 | Left calf | 00000010000000000000000000000000 | 0x02000000 | 33 554 432 |
| 27 | Right foot palm | 00000100000000000000000000000000 | 0x04000000 | 67 108 864 |
| 28 | Left foot palm | 00001000000000000000000000000000 | 0x08000000 | 134 217 728 |
| 29 | Right foot dorsum | 00010000000000000000000000000000 | 0x10000000 | 268 435 456 |
| 30 | Left foot dorsum | 00100000000000000000000000000000 | 0x20000000 | 536 870 912 |
| 31 | Right foot fingers | 01000000000000000000000000000000 | 0x40000000 | 1 073 741 824 |
| 32 | Left foot fingers | 10000000000000000000000000000000 | 0x80000000 | 2 147 483 648 |

Table 8: Some examples of Body Parts Combinations.

|  |  |  |  |
| --- | --- | --- | --- |
| *Name* | *body\_part\_mask* | *Hexa* | *Value* |
| Right arm | 00000000000101010101010000000000 | 0x00015540 | 87 360 |
| Left arm | 00000000001010101010100000000000 | 0x002AA800 | 2 795 520 |
| Right leg | 01010101010000000000000000000000 | 0x55400000 | 1 430 257 664 |
| Left leg | 10101010100000000000000000000000 | 0xAA800000 | 2 860 515 328 |
| Upper body | 00000000001111111111111111111111 | 0x003FFFFF | 4 194 303 |
| Lower body | 11111111110000000000000000000000 | 0xFFC00000 | 4 290 772 992 |
| Full body | 11111111111111111111111111111111 | 0xFFFFFFFF | 4 294 967 295 |

**Semantic body parts & actuators mapping** Alternatively, the haptic signal spatialization on the body representation can be done by a combination of body part targeting and actuator map definition on this body. This approach is using the *body part target* metadata which is a set of successive values configured to represent a human body part or a group semantically. The range of different values is defined in Table 9:

* A *Group Node* is a shortcut command to target a group of body parts
* A *Locational* value is an optional metadata to some body part node to give add precision to this one
* A *Body part Node* is a human body part which is defined hierarchically (cf. Figure 7) to represent the entire human body

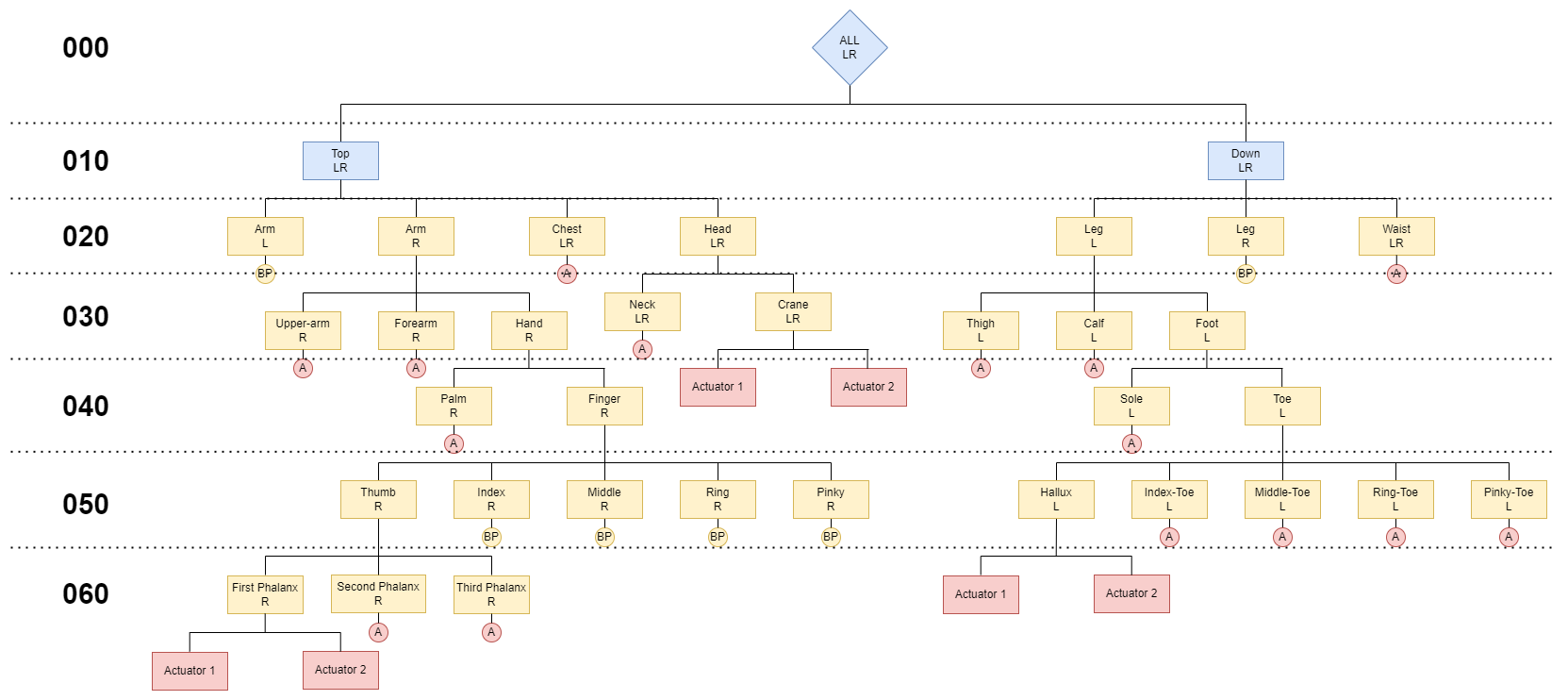
**

Figure 7 Tree representation of the body part targets values

Table 9 body part target values

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| Unknown | Group Node | 0 |
| All | Group Node | 1 |
| Top | Group Node / Locational | 10 |
| Down | Group Node / Locational | 11 |
| Right | Group Node / Locational | 12 |
| Left | Group Node / Locational | 13 |
| Front | Group Node / Locational | 14 |
| Back | Group Node / Locational | 15 |
| Arm | Body part Node | 20 |
| Head | Body part Node | 21 |
| Chest | Body part Node | 22 |
| Waist | Body part Node | 23 |
| Leg | Body part Node | 24 |
| Upper-arm | Body part Node | 30 |
| Forearm | Body part Node | 31 |
| Hand | Body part Node | 32 |
| Crane | Body part Node | 33 |
| Neck | Body part Node | 34 |
| Thigh | Body part Node | 35 |
| Calf | Body part Node | 36 |
| Foot | Body part Node | 37 |
| Palm | Body part Node | 40 |
| Finger | Body part Node | 41 |
| Sole | Body part Node | 42 |
| Toe | Body part Node | 43 |
| Thumb | Body part Node | 50 |
| Index | Body part Node | 51 |
| Middle | Body part Node | 52 |
| Ring | Body part Node | 53 |
| Pinky | Body part Node | 54 |
| Hallux | Body part Node | 55 |
| Index-Toe | Body part Node | 56 |
| Middle\_Toe | Body part Node | 57 |
| Ring-Toe | Body part Node | 58 |
| Pinky-Toe | Body part Node | 59 |
| First Phalanx | Body part Node | 60 |
| Second Phalanx | Body part Node | 61 |
| Third Phalanx | Body part Node | 62 |
| Minus | Operator | 254 |
| Plus | Operator | 255 |

Each body part is then divided into a 3-dimensional right-handed actuator map which fulfils the following requirement:

* The x-axis corresponds to the right axis, represented in red
* The y-axis corresponds to the up axis, represented in green
* The z-axis corresponds to the forward axis, represented in blue
* The up axis on each body part is looking upward based on a standing human looking at his hand outstretched in front of himself (cf. Figure 8 for a visual representation)
* The right axis on each body part is looking rightward based on a standing human looking at his hand outstretched in front of himself (cf. Figure 8 for a visual representation)

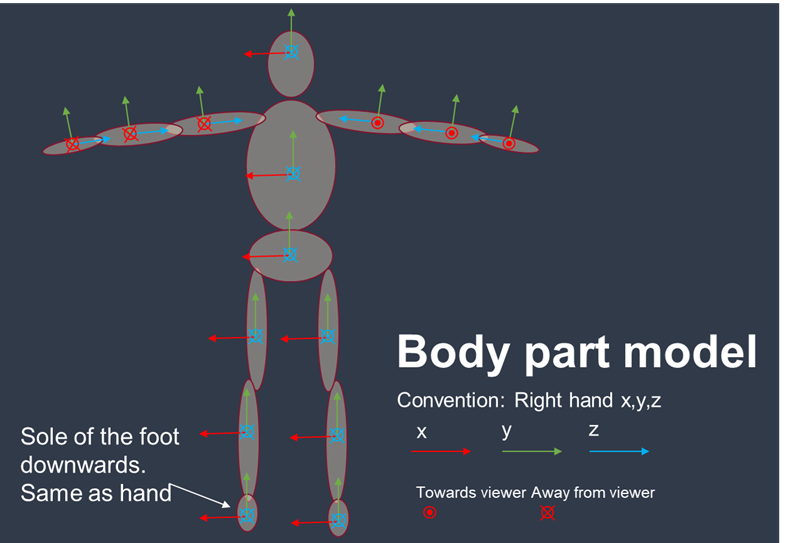


Figure 8 Representation of each body part basis for the actuator map

Each actuator can finally be targeted individually on the targeted body part to improve the precision of a haptic signal spatialization on the human body.

The actuators are mapped on the body part following 3d mapping based on x,y,z resolution. The x,y,z direction are described in Figure 8. For only 1 actuator, a resolution of [1,1,1] can be included and indicates 1 actuator. A resolution of [2,1,1] indicates 2 actuators on the x axis, on the body part. A resolution of [2,2,1] indicates 4 actuator on the x,y plane.A resolution of [2,2,2] indicates 8 actuators arranged with two surfaces. For haptics devices which do not respect a cubic volumetric arrangement, some of the actuators might be virtual.

More details on the use of this representation are provided in Annex C.

# Haptic vector

The detailed list of properties defining a vector is given in Table 10.

Table 10: List of properties of a direction element

|  |  |
| --- | --- |
| **Property** | **Description** |
| X | Unit Right vector.  The range of possible values stored in the axis is [-127, 127]. |
| Y | Unit Up vector.  The range of possible values stored in the axis is [-127, 127]. |
| Z | Unit Forward vector.  The range of possible values stored in the axis is [-127, 127]. |

# Haptic band

A haptic band describes the haptic signal of a channel in a given frequency range. Bands are defined by a type and a sequential list of haptic effects each containing a set of keyframes. Table 11 details the list of properties of a haptic band.

Table 11: List of properties of a haptic band

|  |  |
| --- | --- |
| **Property** | **Description** |
| band type | Type of data contained in the band. There are 4 types of haptics bands: Curve bands, Transient bands, Vectorial wave bands and Wavelet wave bands. For each type of band, the information it contains has a different meaning:   * Curve: Curve bands represent haptic signals with curves, described by a set of control points and a type of interpolation in-between * Transient: Transient bands represent short momentary haptic effects of fixed duration, described with amplitude and frequency parameters. * Vectorial Wave: Vectorial Wave bands represent parametric haptic effects; described by a vector of parameters including temporal or spatial position, amplitude and frequency. The model allows both amplitude and frequency modulation of the signal. * Wavelet Wave: Wavelet Wave bands represent haptic effects encoded with wavelet transform decomposition, quantization, binary tree structure, and entropy coding   Clause 6.10 details precisely how the data contained in keyframes is interpreted depending on the type of bands. |
| curve type | Present only for Curve bands. This specifies the interpolation method that shall be used to synthetize the haptic signal of the band. Possible values are:   * Linear * Cubic * Akima * Bezier * BSpline * Unknown (for application specific functions) | |
| block length | This property is only present for Wavelet Wave bands. It is the duration of a wavelet block. | |
| lower frequency limit | Lower frequency limit of the band (Hz). | |
| upper frequency limit | Upper frequency limit of the band (Hz). | |
| effects | List of Haptic effects as defined in 6.9.  If the effect list is empty, the band does not contain haptic data. |

Figure 6 illustrates two types of haptic bands: a curve band and a transient band.

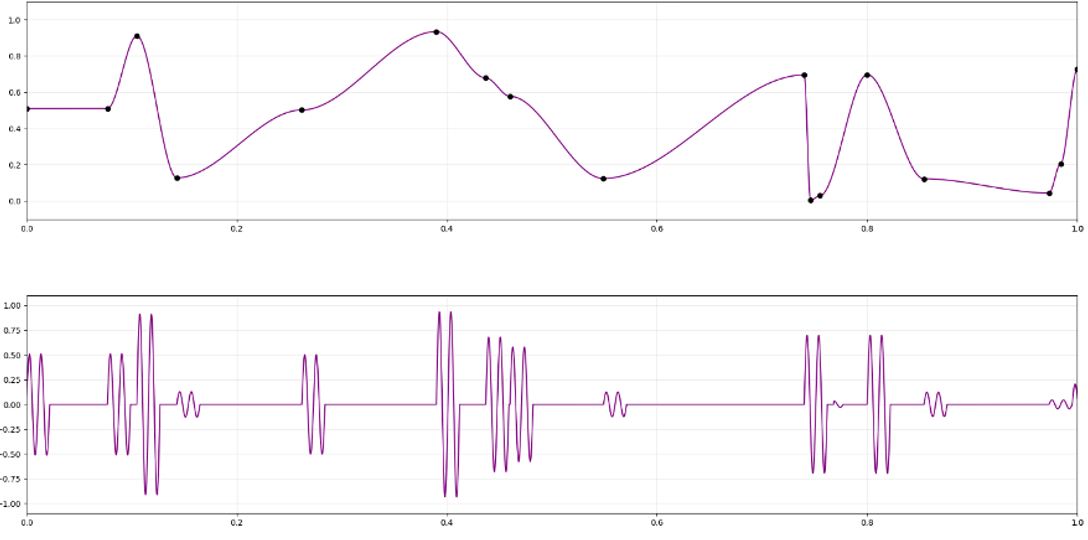


Figure 9: Representation of a Curve band (top) and a Transient band (bottom). Points represent Keyframes.

For Curve bands, different types of curve representations and interpolations can be used. The format currently supports 5 types of curves: Linear, Cubic, Akima, Bézier and BSpline.

* Linear:

Each keyframe is a point in the curve and intermediate values are computed using linear interpolation.

* Cubic

Each keyframe is a point in the curve and intermediate values are computed using Piecewise cubic polynomials.

* Akima

Each keyframe is a point in the curve and intermediate values are computed using Akima interpolation.

* Bézier

Bezier curves are parametric curves defined by a set of control points. Bézier curves are generally used to model smooth curves that can be scaled indefinitely. It is defined by a set of control points P0 through Pn, where n is called the order of the curve (n = 1 for linear, 2 for quadratic, 3 for cubic, etc.). The interpolation function of a quadratic Bézier curve is given by:

*,*

Where *B(t)* is the interpolation for a parameter *t* (between 0 and 1), and (*P0, P1, P2*) are respectively the 3 control points used for the interpolation (see Figure 10).

Chart

Description automatically generated

Figure 10: Bezier curve example, with the three control points Pi and B(t) the sample to be predicted.

Here, the data in curve bands represent piecewise 2nd order Bézier curves: the data is composed of consecutive quadratic Bézier Curves each defined by a set of 3 control points and where the last control point of a Bézier curve is the first control point of the next one. This implies that the data contains at least 3 control points and an uneven number of control points. The control points contain 2D data defining their amplitude and timestamp. To ensure that the haptic data can be rendered properly, the control points must be ordered in time: the timestamp of each control point must be higher than the timestamp of the previous point (if it exists) and lower than the timestamp of the next control point (if it exists).

Figure 11 illustrates the curve produced with a set of 5 control points. The signal is computed using two Bézier curves with P0, P1 and P2 as control points for the first one and P2, P3 and P4 for the second one.

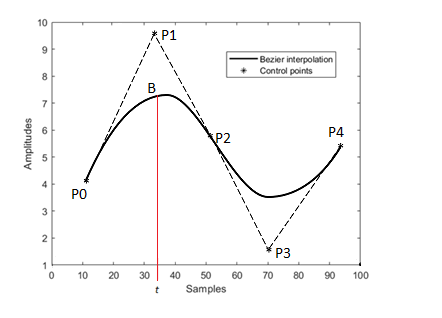


Figure 11: Example of piecewise Bézier curve defined by a set of 5 control points

* B-Splines

A B-spline function is a combination of flexible segments controlled by a number of control points (*Pi*) creating smooth curves (see Figure 8). It is a generalization of Bezier curves, with higher interpolation accuracy. The shape of the curve only depends on the position of the control points. Modifying one control point will only change the curve locally. Here the data in the curve bands represents the control points of the B-Spline.

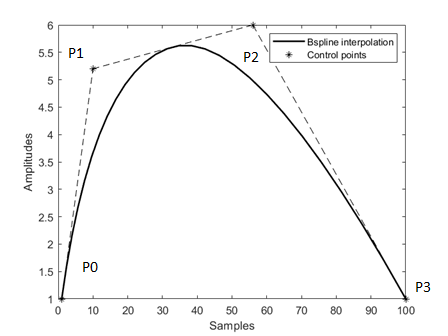


Figure 12: Example of B-Spline defined with a set of four control points P0, P1, P2 and P3.

# Haptic effect

For every type of haptic band, haptic effects are defined with at least a position and a type. Depending on the type of band and the type of effect, additional properties must be specified, including the phase, the base signal, a composition and a number of consecutive haptic keyframes describing the effect. The detailed list of properties of a haptic effect is given in Table 12.

Table 12: List of properties of a haptic effect

|  |  |
| --- | --- |
| **Property** | **Description** |
| **id** | Unique identifier of an effect. This property is required for effects defined at the perception level in the library and for effects of type “Reference”. For “Reference” effects, this property corresponds to the id of the library effect being referenced. |
| **effect type** | Type of haptic effect. Possible values are:   * Basis: Effect containing signal data defined through a set of keyframes * Composite: Effect composed of a set of other effects defined in the composition property. This type of effect does not directly contain keyframes * Reference: Effect referencing an effect in the library. This type of effect is used at the band level to reference effects defined at the perception level in the effect library with a unique id. |
| **position** | Indicates the temporal or spatial position of the effect. The value 0 corresponds to the starting position of the experience. The default unit for temporal haptic feedback will be milliseconds while it will be millimetres for spatial haptic feedback. |
| **phase** | Indicates the phase of the effect given in radian in the range [0; 2π]. This value is used only for vectorial wave bands and is ignored in effects stored inside a band of a different type. |
| **base signal** | This property is only used for Vectorial Wave bands. It defines the type of waveform signal to be used by the synthesizer. Possible values are:   * Sine * Square * Triangle * SawToothUp * SawToothDown |
| **composition** | This property can only be used with Composite effects. It contains a sub-list of effects. |
| **keyframes** | List of keyframes as defined in 6.10  The keyframes list is only used for Reference effects. |

The duration of a Haptic effect is given, depending on the band type, as:

* Transient band: the position of the latest keyframe plus the duration of a transient. In this representation, every transient have a fixed duration that can differ from different implementations.
* Curve band: the position of the latest keyframe
* Vectorial Wave band: the position of the latest keyframe
* Wavelet Wave band: For this type of band, each effect corresponds to a single block. The duration of an effect then corresponds to the block length property specified at the band level.

# Haptic keyframe

A haptic keyframe encodes a point/sample at a given position in time or space.

Table 13: List of properties of a haptic keyframe

|  |  |
| --- | --- |
| **Property** | **Description** |
| amplitude modulation | Amplitude of the keyframe |
| frequency modulation | Relative frequency of the keyframe | |
| relative position | Relative position of the keyframe. |

The type of band indicates how a keyframe should be interpreted. Depending on the type of band, a keyframe stores one or several of the properties defined in Table 13. More precisely:

|  |  |
| --- | --- |
| **Band type** | **Keyframe interpretation** |
| **Transient** | keyframes define with a position, an amplitude, and a frequency |
| **Curve** | keyframes define a position and an amplitude. The keyframes represents the control points of the curve. The type of curve is specified at the band level and is used to compute the signal as detailed in clause 6.8. |
| **Vectorial Wave** | keyframes define a position, an optional amplitude modulation and an optional frequency modulation. A keyframe must contain at least one of these two optional parameters. Any interpolation between keyframes can be used. |
| **Wavelet Wave** | an effect stores the content of one wavelet block (also called effect), in the wavelet domain. It contains a keyframe for every coefficient of the wavelet transformed and quantized signal, with only the amplitude value used. The coefficients are scaled to a range of [-1,1]. Additionally, the original maximum amplitude is stored in a keyframe, as well as the maximum number of used bits. |

The combination of multiple of these primitive keyframes allows to describe a full haptic effect.

# Interchange file format

# Overview

The interchange format is a JSON implementation of the data model. It is not memory-optimized, but it is human-readable and can be manually edited. The data structure of this format is illustrated in Figure 13.

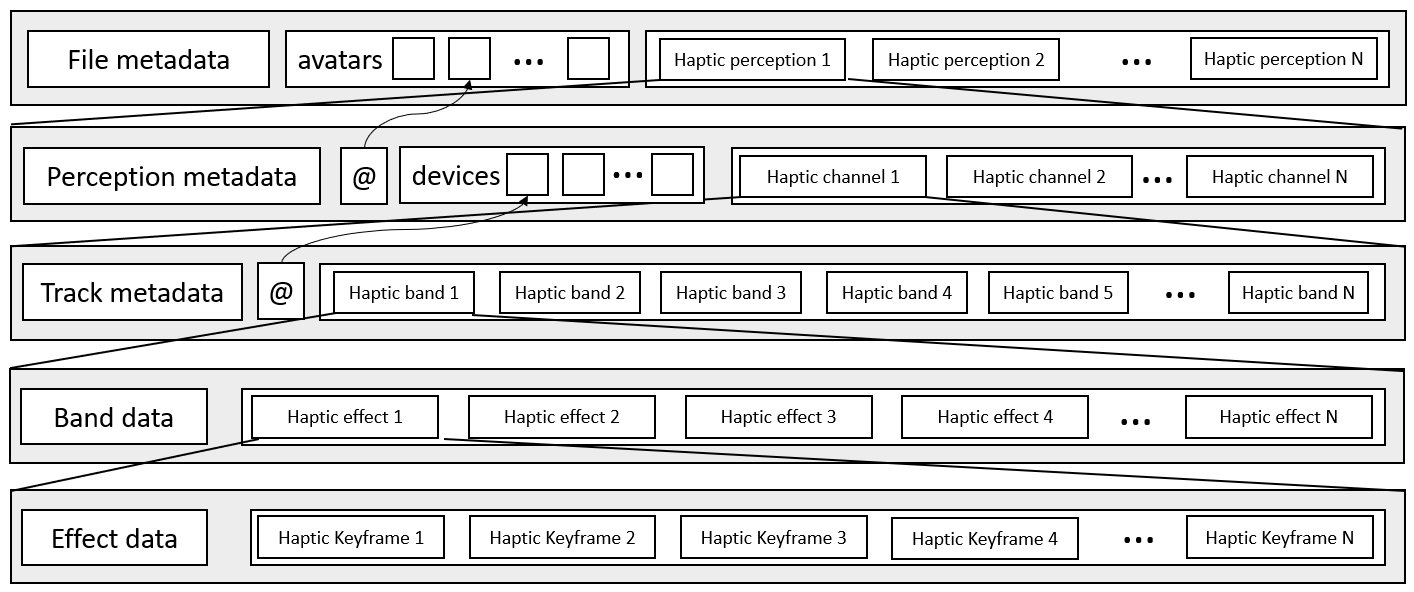


Figure 13: Detailed data structure of the interchange format

As a JSON based format, it is compliant with the RFC8259 and ECMA-404 JSON standard specification. Each component of the data structure is defined as a JSON object and described below using the JSON Schema specification mechanism. The interchange format is described through the following list of JSON schemas. All the Schemas are provided in Annex A

Table 14: List of all the JSON schemas describing the interchange format

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Brief Description** | **Type** | **Subclause** |
| MPEG\_haptics | Describes the haptic experience. This refers to clause 6.2 of the data model. | Generic | 7.2.1 |
| MPEG\_haptics\_avatar | Describes a haptic avatar. This refers to clause 6.3 of the data model. A haptic experience may include several avatars. | Generic | 7.2.2 |
| MPEG\_haptics\_perception | Describes a haptic signal associated to a specific perception modality (Vibration, Force, Temperature, …). This refers to clause 6.4 of the data model. A haptic experience may include several modalities. | Generic | 7.2.3 |
| MPEG\_haptics\_reference\_device | Provides a list of characteristics associated with a targeted haptic device. This refers to clause 6.5 of the data model. A haptic experience may include several reference devices. | Generic | 7.2.4 |
| MPEG\_haptics\_channel | Describes a channel within a perception. Each perception may contain multiple channels. This refers to clause 6.6 of the data model. | Generic | 7.2.5 |
| MPEG\_haptics\_vector | Describes the local direction on which the signal should be defined. This information is relevant for the spatial based haptic signals. This refers to clause 6.7 of the data model. | Generic | 7.2.6 |
| MPEG\_haptics\_band | Describes a frequency band within a haptic channel. This refers to clause 6.8 of the data model. A haptic channel may contain multiple bands. | Generic | 7.2.7 |
| MPEG\_haptics\_effect | Describes a haptic effect within a frequency band. This refers to clause 6.9 of the data model. A haptic band may contain multiple effects. | Generic | 7.2.8 |
| MPEG\_haptics\_keyframe | Describes a keyframe within a haptic effect. Properties contained in a keyframe are interpreted differently depending on the type of haptic band and the specified encoding modality. This refers to clause 6.10 of the data model. | Generic | 7.2.9 |

# HJIF Specifications

This clause details the specifications of the interchange format. Each component of the data structure is defined as JSON object with a list of properties corresponding to its attributes. The following clauses give a detailed description of the JSON objects and associated schemas as specified in Table 14.

The clauses below use the following types to describe properties:

|  |  |
| --- | --- |
| string | The string type is used for strings of text. It may contain Unicode characters. |
| integer | The integer type is used for integral numbers |
| number | The number type is used for any numeric type, either integers or floating-point numbers. |
| boolean | The boolean type matches only two special values: “true” and “false”. |
| array | Arrays are used for ordered lists of elements. |
| object | Objects are the mapping type in JSON. They map “keys” to “values”. In JSON, the “keys” must always be strings. Each of these pairs is conventionally referred to as a “property”. The properties (key-value pairs) on an object are defined using the “properties” keyword. |
| enum | The enum keyword is used to restrict a value to a fixed set of values. |

# MPEG\_haptics

The MPEG\_haptics element is the highest level of the format and describes the global haptic experience as defined in clause 6.2. It includes some metadata such as the date of the file and the version of the format, it provides a description of the haptic experience, it specifies the shape associated with the haptic experience, it lists the different avatars (i.e., body representation) and defines all the perceptions.

The MPEG\_haptics metadata are necessary to configure the global experience, and include all the data structure introduced above, starting with the perceptions. Table 15 details the list of properties of a MPEG\_haptics object corresponding to the properties defined in Table 1 of clause 6.2.

Table 15: Description of the MPEG\_haptics object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| version | string | N/A | Provides the version of this ISO/IEC 23090-31 specification. | Yes |
| date | string | N/A | Indicates the creation date of this haptic experience. | Yes |
| description | string | N/A | The user-defined description of this haptic experience. | Yes |
| avatars | array< MPEG\_haptics.avatar> | N/A | Provides the List of MPEG\_haptics.avatar as described in 7.2.2. | Yes |
| perceptions | array< MPEG\_haptics.perception> | N/A | Provides the List of MPEG\_haptics.perception as described in 7.2.3. | Yes |

# MPEG\_haptics.avatar

The MPEG\_haptics.avatar element defines a haptic avatar as a body representation. It is particularly useful to specify in which part of the body the haptic effect should apply. This corresponds to the clause 6.3 of the data model.

Different types of avatars may be used for different applications or different haptic properties. A custom 3D mesh can be specified for the representation of the avatars as described in clause 0. When using a custom 3D mesh representation, a reference URI to a custom mesh from a companion file must be provided (*.obj* or *.glTF*  file formats may be used).

Table 16 details the list of properties of an MPEG\_haptics.avatar object corresponding to the properties defined in Table 2 of clause 6.3.

Table 16: Description of the MPEG\_haptics.avatar object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| id | integer | N/A | Unique identifier of the avatar. The value must be greater or equal to 0. | Yes |
| lod | integer | N/A | If the avatar uses a mesh with several levels of detail (LODs), this indicates which LOD to use for the avatar. | Yes |
| type | enum<string> | N/A | Indicates the type of haptic perception represented by the avatar. Possible values are:   * "Vibration" * "Pressure * "Temperature" * “Custom” | Yes |
| mesh | string | N/A | URI path to the custom mesh file. The URI must follow the syntax defined in RFC3986 | No |

# MPEG\_haptics.perception

The MPEG\_haptics.perception element specifies a haptic perception with an id, a description of the perception, some metadata information (*e.g.* perception modality and type of encoding), a reference to an avatar, a list of reference devices and the list of channels. Table 17 details the list of properties of an MPEG\_haptics.perception object corresponding to the properties defined in Table 3 of clause 6.4.

Table 17: Description of the MPEG\_haptics.perception object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| id | integer | N/A | Unique identifier of the perception. The value must be greater or equal to 0. | Yes |
| perception\_modality | enum<string> | N/A | Indicates the type of perception as defined in 6.4. Possible values are:   * "Pressure" * "Acceleration" * "Velocity" * "Position" * "Temperature" * "Vibrotactile" * "Water" * "Wind" * “Force” * “Electrotactile” * “Vibrotactile Texture” * “Stiffness” * “Friction”, * "Other" | Yes |
| description | string | N/A | The user-defined description of the haptic perception. | Yes |
| avatar\_id | integer | N/A | Unique identifier of the associated avatar body model from 7.2.2. | Yes |
| effect\_library | array<MPEG\_haptics.effect> | N/A | List of predefined MPEG\_haptics.effect as defined in 7.2.8. The list may be empty. Library effects are referenced directly in the channels. | Yes |
| reference\_devices | array< MPEG\_haptics.reference\_devices> | N/A | List of targeted MPEG\_haptics\_reference.device devices or actuators as defined in 7.2.4 for this haptic perception. | No |
| channels | array< MPEG\_haptics.channel> | N/A | List of MPEG\_haptics.channel as defined in 7.2.5 composing this perception. | Yes |
| unit\_exponent | integer | -3 | Refers to the exponent of the powers of 10 for the SI unit identifying the space of representation of the independent variable (cf.6.4 for each input unit) | No |
| perception\_ unit\_exponent | integer | 0 | Refers to the exponent of the powers of 10 for the SI unit measure of the dependent variable (cf. 6.4 for each output perception unit) | No |

# MPEG\_haptics.reference\_device

The MPEG\_haptics.reference\_device specifies a targeted reference device or actuator with an id, a name, and a body location. Additional properties can be optionally specified for each device. Table 18 details the list of properties of an MPEG\_haptics.reference\_device object corresponding to the properties defined in Table 5 of clause 6.5.

Table 18: Description of the MPEG\_haptics.reference\_device object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| id | integer | N/A | Unique identifier of the device. The value must be greater or equal to 0. | Yes |
| name | string | N/A | The user-defined name of the device. | Yes |
| body\_part\_mask | integer | N/A | Binary mask specifying the location of the device or actuator on the body as defined in Table 7 of clause6.6. The value must be greater or equal to 0. | No |
| maximum\_frequency | number | N/A | Maximum frequency of the actuator (Hz) | No |
| minimum\_frequency | number | N/A | Minimum frequency of the actuator (Hz) | No |
| resonance\_frequency | number | N/A | Resonance frequency of the actuator (Hz) | No |
| maximum\_amplitude | number | N/A | Maximum amplitude value of the targeted device according to the perception\_modality. The corresponding unit is specified in Table 4 clause 6.4. | No |
| impedance | number | N/A | Impedance of the actuator (Ω) | No |
| maximum\_voltage | number | N/A | Maximum voltage of the actuator (V) | No |
| maximum\_current | number | N/A | Maximum current of the actuator (A) | No |
| maximum\_displacement | number | N/A | Maximum displacement of the actuator (mm) | No |
| weight | number | N/A | Weight of the device (Kg) | No |
| size | number | N/A | Size of the device (mm) | No |
| custom | number | N/A | User-defined data. This parameter may be used to specify additional properties of the targeted device. | No |
| type | enum<string> | N/A | Indicates the type of actuator.  value equals one of: "LRA", “VCA”, “ERM”, “Piezo”, indicates the respective type of haptic actuator.  value equal to " Unknown " indicates any other modality. | No |

# MPEG\_haptics.channel

The MPEG\_haptics.channel element specifies a channel by an id, a description, a body part, a mixing weight, a gain value, and a list of haptic bands. Various additional properties can also be specified, such as a reference device id, the desired sampling frequency, or the sample count.

Table 19 details the list of properties of an MPEG\_haptics.channel object corresponding to the properties defined in Table 6 of clause 6.6.

Table 19: Description of the MPEG\_haptics.channel object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| id | integer | N/A | Unique identifier of the channel. The value must be greater or equal to 0. | Yes |
| description | string | N/A | The user-defined description of the channel. | Yes |
| reference\_device\_id | integer | N/A | ***id*** of the targeted reference device or actuator from the list defined in the MPEG\_haptics.perception element. | No |
| gain | number | N/A | Gain associated with the channel. | Yes |
| mixing\_weight | number | N/A | Weight of the channel used when mixing different channels together. Value 0 indicates that the channel will not be mixed. | Yes |
| body\_part\_mask | integer | 0 | Binary mask specifying the location of the effect on the body as defined in Table 7 of clause 6.6. The value must be greater or equal to 0. | Yes |
| actuator\_resolution | MPEG\_haptic\_vector | N/A | Reference actuator resolution used to design the haptic experience. | No |
| body\_part\_target | array<string> | N/A | Semantic identification of a unique and/or group of body parts on the human body. All possible values are described in Table 9 body part target values. | No |
| actuator\_target | array<MPEG\_haptics.vector> | N/A | List of different actuators targeted by the channel and identified by its coordinates. | No |
| frequency\_sampling | integer | N/A | Sampling frequency of the original encoded signal (Hz). If present, the value should be greater or equal to 0. | No |
| sample\_count | integer | N/A | Number of samples of the original encoded signal. If present, the value should be greater or equal to 0. | No |
| vertices | array<integer> | N/A | List of indices of the vertices from the avatar impacted by the effect. The associated avatar representation is given by the ***avatar\_id*** indicated in the MPEG\_haptics.perception (see 7.2.3). | No |
| bands | array<MPEG\_haptics.band> | N/A | List of haptic bands composing the channel. A channel may include one or several bands. A band corresponds to a frequency bandwidth as specified in 7.2.7. If the bands array is empty, the channel does not contain haptic data. | Yes |
| direction | MPEG\_haptic.vector | N/A | Spatial direction for the channel.defined with a MPEG\_haptics.vector (see 7.2.6). This property is only used with haptic modalities dependant on the space dimension (i.e. *Vibrotactile Texture*, *Stiffness* and *Friction*). Each integer value stored in this vector will be transformed from its initial range [-127; 127] to the [-1; 1] range to interpret this vector as unitary. | No |

# MPEG\_haptics.vector

When specified in the MPEG\_haptics.channel, the haptic rendering system will use this spatial direction to synthesize the correct haptic feedback based on the user or object movement. Usually, the movement is provided by an appropriate tracking system.

Table 20 details the list of properties of an MPEG\_haptics.vector object corresponding to the properties defined in Table 10 of clause 6.7.

Table 20: Description of the MPEH\_haptics.vector object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| X | integer | N/A | Unit Right vector.  The range of possible values stored in the axis is [-127, 127]. | Yes |
| Y | integer | N/A | Unit Up vector.  The range of possible values stored in the axis is [-127, 127]. | Yes |
| Z | integer | N/A | Unit Forward vector.  The range of possible values stored in the axis is [-127, 127]. | Yes |

# MPEG\_haptics.band

The MPEG\_haptics.band element defines a haptic band with the type of the band, an interpolation function, a block length, a frequency range, and a list of haptic effects.

Table 21 details the list of properties of an MPEG\_haptics.band object corresponding to the properties defined in Table 11 of clause 6.8.

Table 21: Description of the MPEG\_haptics.band object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| band\_type | enum<string> | N/A | Indicates the type of data contained in the band as specified in clause 6.8. Possible values are:   * “WaveletWave” * “VectorialWave” * “Curve” * “Transient” | Yes |
| curve\_type | enum<string> | Linear | Indicates the type of interpolation function that should be used by the synthesizer as specified in clause 6.8.  Possible values are:   * “Linear” * “Cubic” * “Akima” * “Bezier” * “BSpline” * “Unknown” | No |
| block\_length | number | N/A | Duration of a wavelet effect. This property is required for wavelet wave bands. The unit of this property is milliseconds. | No |
| lower\_frequency\_limit | number | N/A | Lower frequency limit of the band (Hz). The value should be in the range [0;10000]. | Yes |
| upper\_frequency\_limit | number | N/A | Indicates the upper frequency limit of the band (Hz). The value should be in the range [0;10000]. | Yes |
| effects | array< MPEG\_haptics\_effect > | N/A | List of MPEG\_haptics.effect as defined in 7.2.8.  If the effects array is empty, the band does not contain haptic data. | Yes |

# MPEG\_haptics.effect

Each MPEG\_haptics.band is composed of haptic effects defined by the MPEG\_haptics.effect element with an effect type, a position, a phase, a signal type, an optional composition and a list of keyframes.

Table 22 details the list of properties of an MPEG\_haptics.band object corresponding to the properties defined in Table 12 of clause 6.9.

Table 22: Description of the MPEH\_haptics.effect object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| id | integer | N/A | Unique identifier of an effect. This attribute is only required for library effects and “Reference” effects. For “Reference” effects, it corresponds to the id of the library effect being referenced. The value must be greater or equal to 0. | No |
| effect\_type | enum<string> | Basis | Indicates the type of haptic effect.  Effect-type value equals one of: “Basis”, “Composite” and “Reference”, corresponding to the description in clause 6.9. | Yes |
| position | integer | 0 | Indicates the temporal or spatial position of the effect. The value must be greater or equal to 0. | Yes |
| phase | number | 0 | Phase of the effect. The value should be in the range [0, 6.28318]. | Yes |
| base\_signal | enum<string> | Sine | Indicates the type of the waveform signal. This property is required for Vectorial Wave bands. Possible values are:   * “Sine” * “Square” * “Triangle” * “SawToothUp” * “SawToothDown” | No |
| composition | array<MPEG\_haptics.effect> | N/A | This attribute can only be used with Composite effects. It contains a list of effects. This type of effect does not directly contain keyframes. | No |
| keyframes | array<MPEG\_haptics.keyframe> | N/A | List of MPEG\_haptics.keyframes as defined in 7.2.9. This property is required for Basis effects.  If the keyframes array is empty, the effect does not contain haptic data. | No |

# MPEG\_haptics.keyframe

A MPEG\_haptics.effect is described by a set of MPEG\_haptics.keyframes. Depending on the type of the haptic band and the encoding modality, a keyframe is defined with an amplitude modulation, a frequency modulation and/or a relative position.

Table 23 details the list of properties of an MPEG\_haptics.band object corresponding to the properties defined in Table 13 of clause 6.10.

Table 23: Description of the MPEG\_haptics.effect object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Type** | **Default** | **Description** | **Required** |
| amplitude\_modulation | number | N/A | Amplitude of the keyframe. The value must be in the range [-1:1]. | No |
| frequency\_modulation | number | N/A | Relative frequency of the keyframe. The value must be in the range [0:10000] | No |
| relative\_position | integer | N/A | Relative position of the keyframe. The value must be greater or equal to 0. | No |

# Compressed binary file format

The proposed compressed binary file format is based on the data model presented in clause 6 and stores the same information as the interchange format described in clause 7. The goal of this binary format is to compress information while maintaining a one-to-one correspondence with the interchange format. Information is stored in a binary form and data compression is applied for the data at the band level.

As shown in the table below, the data is divided into two mains blocks: the header and the payload.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File format** | | | | |
| **Binary header** | | | **Binary payload** | |
| **Haptic file metadata** | **<< Repeated >>**  **Perception header** | | **<< Repeated >>**  **Band data** | |
| **Perception metadata** | **<<Repeated >>**  **Channel metadata** |
| **Band header** | **Band body** |

Note: << Repeated >> means that several header/data can be inserted in the stream to cope with the number of perception, channels and/or bands.

Metadata information related to the haptic experience, the perceptions and the channels is stored at the beginning of the file in the “Binary header” block, while the band data is stored in the “Binary payload” block.

# Binary header

The specification of the header bitstream is detailed in the following tables. Each property stored in the binary format is part of the data model presented in clause 6 and has its counterpart in the interchange format presented in clause 7.

# MPEG\_haptics metadata

Table 24 describes the syntax used to store the metadata information of the haptic experience as defined in clause 6.2.

Table 24: Syntax for reading the metadata of a haptic experience (MPEG\_haptics()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics() |  |  |
| { |  |  | |
| **versionStringSize** | **8** | **uimsbf** | |
| **version;** | **versionStringSize\*8** | **string** | |
| **dateStringSize** | **8** | **uimsbf** | |
| **date;** | **dateStringSize\*8** | **string** | |
| **descriptionStringSize** | **8** | **uimsbf** | |
| **description;** | **descriptionStringSize\*8** | **string** | |
| **avatarCount;** | **8** | **uimsbf** | |
| for (i = 0; i < **avatarCount**; i++) { |  |  | |
| MPEG\_haptics\_avatar(); |  |  | |
| } |  |  | |
| **perceptionCount;** | **8** | **uimsbf** | |
| for (i = 0; i < **perceptionCount**; i++) { |  |  | |
| MPEG\_haptics\_perception(); |  |  | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 25 and maps to the properties detailed in Table 1 of clause 6.2.

Table 25: Semantics associated to the MPEG\_haptics() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **version** | String specifying the version of this ISO/IEC 23090-31 specification |
| **date** | String indicating the creation date of this haptic experience |
| **description** | User-defined description of the haptic experience |
| **avatarCount** | Number of avatars defined for the haptic experience |
| **perceptionCount** | Number of perceptions contained in the haptic experience |

# MPEG\_haptics\_avatar metadata

Table 26 describes the syntax used to store the metadata information of a haptic avatar as detailed in clause 6.3.

Table 26: Syntax for reading the metadata information of a haptic avatar (MPEG\_haptics\_avatar()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_avatar() |  |  |
| { |  |  | |
| **id;** | **8** | **uimsbf** | |
| **lod;** | **8** | **uimsbf** | |
| **type;** | **4** | **uimsbf** | |
| if (type == 0) { |  |  | |
| **meshStringSize** | **8** | **uimsbf** | |
| **mesh;** | **meshStringSize\*8** | **string** | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 27 and maps to the properties detailed in Table 2 of clause 6.36.3.

Table 27: Semantics associated to the MPEG\_haptics\_avatar() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **id** | Unique identifier of the avatar |
| **lod** | Level of detail of the avatar |
| **type** | Type of avatar. Possible values are provided in Table 28 |
| **mesh** | String defining the URI path to a companion mesh file |

Possible values for the avatar type are detailed in Table 28.

Table 28: Value of type for MPEG\_haptics\_avatar()

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Custom | |
| 1 | Vibration | |
| 2 | Pressure | |
| 3 | Temperature | |
| 4-15 | /\* Unused values \*/ |

# MPEG\_haptics\_perception metadata

Table 29 describes the syntax used to store the metadata information of a haptic perception as detailed in clause 6.4.

Table 29: Syntax for reading the metadata information of a haptic perception (MPEG\_haptics\_perception()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_perception() |  |  |
| { |  |  | |
| **id;** | **8** | **imsbf** | |
| **perceptionModality;** | **8** | **uimsbf** | |
| **descriptionStringSize** | **8** | **uimsbf** | |
| **description;** | **descriptionStringSize\*8** | **string** | |
| **avatarId;** | **8** | **uimsbf** | |
| **unitExponent;** | **8** | **imsbf** | |
| **perceptionUnitExponent;** | **8** | **imsbf** | |
| **effectCount;** | **16** | **uimsbf** | |
| for (i = 0; i < **effectCount**; i++) { |  |  | |
| MPEG\_haptics\_libraryEffect() |  |  | |
| } |  |  | |
| **referenceDeviceCount;** | **8** | **uimsbf** | |
| for (i = 0; i < **referenceDeviceCount**; i++) { |  |  | |
| MPEG\_haptics\_referenceDevice() |  |  | |
| } |  |  | |
| **channelCount;** | **8** | **uimsbf** | |
| for (i = 0; i < **channelCount**; i++) { |  |  | |
| MPEG\_haptics\_channel() |  |  | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 30 and maps to the properties detailed in Table 3 of clause 6.46.4.

Table 30: Semantics associated to the MPEG\_haptics\_perception() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **id** | Unique identifier of the perception |
| **perceptionModality** | Type of perception modality. Possible values are detailed in Table 31 |
| **description** | User-defined description of the perception |
| **avatarId** | Unique identifier of the associated avatar body model defined in 8.1.2 |
| **effectCount** | Number of effects to read in the library |
| **unitExponent** | The scientific notification exponent for the SI unit identifying the space of representation of the independent variable. For example, for temporal modalities, with an exponent of -3, the values are in milliseconds |
| **perceptionUnitExponent** | The scientific notification exponent the SI unit identifying the space of representation of the dependent variable. For example, with the Force perception modality, with an exponent of 3, the values are in kilonewtons |
| **referenceDeviceCount** | Number of reference devices stored in this perception |
| **channelCount** | Number of channels stored in this perception |

Possible values for the perception modality are detailed in Table 31.

Table 31: Value of perceptionModality.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Other | |
| 1 | Pressure | |
| 2 | Acceleration | |
| 3 | Velocity | |
| 4 | Position | |
| 5 | Temperature | |
| 6 | Vibrotactile | |
| 7 | Water | |
| 8 | Wind | |
| 9 | Force | |
| 10 | Electrotactile |
| 11 | Vibrotactile Texture | |
| 12 | Stiffness | |
| 13 | Friction | |
| 14-255 | /\* Unused values \*/ |

# MPEG\_haptics\_libraryEffect metadata

Table 32 describes the syntax used to read an effect from the haptic effect library of a haptic perception as detailed in clause 6.4. Effects from the library have the same structure as effects from a band. The associated list of properties is detailed in Table 12 of clause 6.9.

Table 32: Syntax for reading an effect from the haptic effect library (MPEG\_haptics\_libraryEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_libraryEffect() |  |  |
| { |  |  | |
| **id;** | **16** | **uimsbf** | |
| **position;** | **24** | **uimsbf** | |
| **phase;** | **16** | **decimal** | |
| **baseSignal;** | **4** | **uimsbf** | |
| **effectType;** | **2** | **uimsbf** | |
| **keyframeCount**; | **16** | **uimsbf** | |
| for (i = 0; i < k**eyframeCount**; i++) { |  |  | |
| **informationMask;** | **3** | **uimsbf** | |
| if ((**informationMask** & 0x01) != 0) { |  |  | |
| **relativePosition;** | **16** | **uimsbf** | |
| } |  |  | |
| if ((**informationMask** & 0x02) != 0) { |  |  | |
| **amplitude;** | **8** | **decimal** | |
| } |  |  | |
| if ((**informationMask** & 0x04) != 0) { |  |  | |
| **frequency;** | **16** | **uimsbf** | |
| } |  |  | |
| } |  |  | |
| **compositionEffectCount**; | **16** | **uimsbf** | |
| for (i = 0; i < **compositionEffectCount**; i++) { |  |  | |
| MPEG\_Haptics\_libraryEffect()**;** |  |  | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 33 and maps to the properties detailed in Table 12 of clause 6.9.

Table 33: Semantics associated to the MPEG\_haptics\_libraryEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **id** | Unique identifier of the effect in the library. This id is used by reference effects to reference library effects in the band data |
| **position** | Temporal/spatial starting position of the effect |
| **phase** | Initial phase of the haptic effect given in radians. The range of this decimal number is [0; 6.28318]. 0 is considered as no phase and 6.28318 as one period delay |
| **baseSignal** | Type of waveform signal to be used by the synthesizer. Possible values are detailed in Table 34 |
| **effectType** | Type of effect. Possible values are detailed in Table 35 |
| **keyframeCount** | Number of keyframes of the effect |
| **informationMask** | Binary mask defining which of the keyframe properties are stored |
| **relativePosition** | relative temporal/spatial position of a keyframe |
| **amplitude** | Amplitude value of a keyframe. The range of this decimal number is [-1; 1] |
| **frequency** | Frequency value of a keyframe |
| **compositionEffectCount** | Number of sub-effects contained in the composite effect |

Table 34: Value of baseSignal.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Sine | |
| 1 | Square | |
| 2 | Triangle | |
| 3 | SawToothUp | |
| 4 | SawToothDown | |
| 5-15 | /\* Unused values \*/ |

Table 35: Value of effectType.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Basis | |
| 1 | Reference | |
| 2 | Composite | |
| 3 | /\* Unused value \*/ |

# MPEG\_haptics\_referenceDevice metadata

Table 36 describes the syntax used to store the metadata information of a haptic reference device as detailed in clause 6.5.

Table 36: Syntax for reading a haptic reference device metadata (MPEG\_haptics\_referenceDevice()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_referenceDevice() |  |  |
| { |  |  | |
| **id;** | **8** | **imsbf** | |
| **nameStringSize** | **8** | **uimsbf** | |
| **name;** | **nameStringSize\*8** | **String** | |
| **bodyPartMask;** | **32** | **uimsbf** | |
| **deviceInformationMask;** | **12** | **uimsbf** | |
| if (**deviceInformationMask** & 0x00’01) { |  |  | |
| **maximumFrequency;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’02) { |  |  | |
| **minimumFrequency;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’04) { |  |  | |
| **resonanceFrequency;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’08) { |  |  | |
| **maximumAmplitude;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’10) { |  |  | |
| **impedance;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’20) { |  |  | |
| **maximumVoltage;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’40) { |  |  | |
| **maximumCurrent;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x00’80) { |  |  | |
| **maximumDisplacement;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x01’00) { |  |  | |
| **weight;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x02’00) { |  |  | |
| **size;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x04’00) { |  |  | |
| **custom;** | **32** | **decimal** | |
| } |  |  | |
| if (**deviceInformationMask** & 0x08’00) { |  |  | |
| **type;** | **4** | **uimsbf** | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 37 and maps to the properties detailed in Table 5 of clause 6.5.

Table 37: Semantics associated to the MPEG\_haptics\_referenceDevice() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **id** | Unique identifier of the device. The value must be greater or equal to 0 |
| **name** | The user-defined name of the device |
| **bodyPartMask** | Binary mask specifying the location of the device or actuator on the body |
| **deviceInformationMask** | Binary mask defining which of the reference device properties are stored |
| **maximumFrequency** | Maximum frequency of the actuator (Hz). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’01 is true |
| **minimumFrequency** | Minimum frequency of the actuator (Hz). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’02 is true |
| **resonanceFrequency** | Resonance frequency of the actuator (Hz). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’04 is true |
| **maximumAmplitude** | Maximum amplitude value of the targeted device according to the perception\_modality. The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’08 is true |
| **impedance** | Impedance of the actuator (Ω). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’10 is true |
| **maximumVoltage** | Maximum voltage of the actuator (V). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’20 is true |
| **maximumCurrent** | Maximum current of the actuator (A). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’40 is true |
| **maximumDisplacement** | Maximum displacement of the actuator (mm). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x00’80 is true |
| **weight** | Weight of the device (kg). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x01’00 is true |
| **size** | Diameter of the bounding sphere of the device (mm). The range of this decimal number is [0, 10000]. Present only if **deviceInformationMask** & 0x02’00 is true |
| **custom** | User-defined data. The parameter may be used to specify additional properties of the targeted device. The range of this decimal number is [-10000, 10000]. Present only if **deviceInformationMask** & 0x04’00 is true |
| **type** | Indicate the type of actuator. Possible values are detailed in Table 38. Present only if **deviceInformationMask** & 0x08’00 is true |

Table 38: Value of type.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Unknown | |
| 1 | LRA | |
| 2 | VCA | |
| 3 | ERM | |
| 4 | Piezo | |
| 5-15 | /\* Unused values \*/ |

# MPEG\_haptics\_channel metadata

Table 39 describes the syntax used to store the metadata information of a haptic channel as detailed in clause 6.6.

Table 39: Syntax for reading a haptic channel (MPEG\_haptics\_channel()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_channel() |  |  |
| { |  |  | |
| **id;** | **8** | **uimsbf** | |
| **descriptionStringSize** | **8** | **uimsbf** | |
| **description;** | **descriptionStringSize\*8** | **String** | |
| **deviceId;** | **8** | **uimsbf** | |
| **gain;** | **32** | **decimal** | |
| **mixingWeight;** | **32** | **decimal** | |
| **optionalMetadataMask;** | **3** | **uimsbf** | |
| if ((optionalMetadataMask & 0x01) != 0) { |  |  | |
| **bodyPartMask;** | **32** | **uimsbf** | |
| } |  |  | |
| if ((**optionalMetadataMask** & 0x02) != 0) { |  |  | |
| **trackResolution.X** | **8** | **ilsbf** | |
| **trackResolution.Y** | **8** | **Ilsbf** | |
| **trackResolution.Z** | **8** | **ilsbf** | |
| **bodyPartTargetCount** | **8** | **uilsbf** | |
| for (i = 0; i < **bodyPartTargetCount**; i++) { |  |  | |
| **bodyPartTarget[i]** | **8** | **uilsbf** | |
| } |  |  | |
| **actuatorTargetCount** | **8** | **uilsbf** | |
| for (i = 0; i < **actuatorTargetCount**; i++) { |  |  | |
| **actuatorTarget[i].X** | **8** | **ilsbf** | |
| **actuatorTarget[i].Y** | **8** | **ilsbf** | |
| **actuatorTarget[i].Z** | **8** | **ilsbf** | |
| } |  |  | |
| } |  |  | |
| **frequencySampling;** | **32** | **uimsbf** | |
| if (**frequencySampling** > 0) |  |  | |
| **sampleCount;** | **32** | **uimsbf** | |
| } |  |  | |
| if (**optionalMetadataMask** & 0x04) { |  |  | |
| **direction.X** | **8** | **decimal** | |
| **direction.Y** | **8** | **decimal** | |
| **direction.Z** | **8** | **decimal** | |
| } |  |  | |
| **verticeCount;** | **16** | **uimsbf** | |
| for (i = 0; i < **verticeCount**; i++) { |  |  | |
| **Vertex** | **32** | **uimsbf** | |
| } |  |  | |
| **bandCount;** | **16** | **uimsbf** | |
| for (i = 0; i < **bandCount**; i++) { |  |  | |
| MPEG\_haptics\_band() |  |  | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 40 and maps to the properties detailed in Table 6 of clause 6.6.

The list of different **bodyPartTarget** possibilities and values is described in clause 6.6.

Table 40: Semantics associated to the MPEG\_haptics\_channel() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **id** | Unique identifier of the channel |
| **description** | User-defined description of the channel |
| **deviceId** | Targeted reference device from the list defined in the perception |
| **gain** | Gain associated with the channel to adapt the normalized encoded data values to a typical device |
| **mixingWeight** | Weight of the channel when mixing different channels together to produce a mixed signal |
| **trackResolution** | Reference actuator resolution used to design the haptic experience for each spatial coordinate (X, Y, Z) |
| **bodyPartTarget** | Semantic identification of a unique and/or group of body parts on the human body |
| **actuatorTarget** | List of different actuators targeted by the channel and identified by its coordinates (X, Y, Z) |
| **bodyPartMask** | Binary mask specifying the location of the effect on the body |
| **optionalMetadataMask** | Binary mask to define if optional information is stored or not |
| **frequencySampling** | Sampling frequency of the original encoded signal (Hz) |
| **sampleCount** | Number of samples of the original encoded signal |
| **direction** | Spatial direction of the encoded signal. This value is based on a local representation of the 3-dimensional space given by the presentation engine |
| **verticeCount** | Number of vertices stored in the channel |
| **vertex** | Vertex from the avatar impacted by the effect. The appropriate avatar representation is referenced by the avatar id indicated at the perception level (see 6.4) |
| **bandCount** | Number of bands stored in the channel |

# Binary payload

The binary payload of the compressed format contains the information allowing to reconstruct the effect of each MPEG\_haptics\_band in the stream. The bitstream of the payload block is encoded differently depending on the band type and its encoding modality. Data should be interpreted differently for:

* Transient bands
* Curve bands
* VectorialWave bands
* WaveletWave bands

Each MPEG\_haptics\_band is composed of a band header with the metadata and a band payload containing the encoded data in a list of effects.

# MPEG\_haptics\_band

Table 41 describes the syntax used to store the information of a haptic band as detailed in clause 6.8 and 6.9. The first part of the syntax describes how to read the header of the band containing metadata information and the second part describes how to read the payload with the list of haptic effects.

Table 41: Syntax for reading a haptic band (MPEG\_haptics\_band()).

|  |  |  |  |
| --- | --- | --- | --- |
| Syntax | No. of bits | Mnemonic |  |
| MPEG\_haptics\_band() |  |  |  |
| { |  |  |  | |
| **bandType;** | **3** | **uimsbf** | *Band header* | |
| if (**bandType** == 1 { |  |  |
| **curveType;** | **4** | **uimsbf** |
| } else if (**bandType** == 3) { |  |  |
| **blockLengthLog;** | **8** | **uimsbf** |
| } |  |  |
| **lowerFrequencyLimit;** | **16** | **uimsbf** |
| **upperFrequencyLimit;** | **16** | **uimsbf** |
| **effectCount;** | **16** | **uimsbf** |
| for (i = 0; i < **effectCount**; i++) { |  |  | *Band payload* | |
| **effectType** | **2** | **uimsbf** |
| if (**effectType** == 0 && **bandType** == 3) { |  |  |
| **effectPosition** = i \* **blockLengthMS**; |  |  |
| }else{ |  |  |
| **effectPosition** | **24** | **uimsbf** |
| } |  |  |
| if (**effectType** == 1) { |  |  |
| **effectId**; | **16** | **uimsbf** |
| }else if(**effectType** == 2){ |  |  |
| MPEG\_haptics\_compositeEffect(); |  |  |
| }else{ |  |  |
| if(**bandType** == 0){ |  |  |
| MPEG\_haptics\_transientEffect(); |  |  |
| }else if(**bandType** == 1){ |  |  |
| MPEG\_haptics\_curveEffect(); |  |  |
| }else if(**bandType** == 2){ |  |  |
| MPEG\_haptics\_vectorialEffect(); |  |  |
| }else if(**bandType** == 3){ |  |  |
| MPEG\_haptics\_waveletEffect() |  |  |
| } |  |  |
| } |  |  |
| } |  |  |  | |
| } |  |  |  | |

The semantics associated with this syntax is detailed in Table 42 and maps to the properties detailed in Table 11 and Table 12 of clauses 6.8 and 6.9.

Table 42: Semantics associated to the MPEG\_haptics\_band() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **bandType** | Type of data contained in the band. Possible values are detailed in Table 43 |
| **curveType** | Present only if **bandType** is’Curve’. This specifies the interpolation method that should be used to synthetize the haptic signal. Possible values are detailed in Table 44 |
| **blockLengthLog** | Present only if **bandType** is “wavelet”. It specifies the log number of samples per wavelet block according to:  Where *blockLength* is the number of samples of the block.  This leads to an allowed minimum blockLength of 16 and only powers of 2 in samples, which is required for wavelet coding. |
| **blockLengthMS** | Present only if **bandType** is “wavelet”. It specifies the duration (in ms) of the wavelet effect. |
| **lowerFrequencyLimit** | Lower frequency limit of the band (Hz) |
| **upperFrequencyLimit** | Upper frequency limit of the band (Hz) |
| **effectCount** | Number of effect stored in the band |
| **effectType** | Type of effect. Possible values are detailed in Table 45 |
| **effectPosition** | Indicates the temporal or spatial position of the effect |
| **effectId** | Unique identifier of an effect. Present only if **effectType** is‘Reference’. It corresponds to the id of the library effect being referenced |

Table 43: Value of bandType.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Transient | |
| 1 | Curve | |
| 2 | VectorialWave | |
| 3 | WaveletWave | |
| 4-7 | /\* Unused values \*/ |

Table 44: Value of curveType.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Unknown | |
| 1 | Cubic | |
| 2 | Linear | |
| 3 | Akima | |
| 4 | Bezier | |
| 5 | BSpline | |
| 6-15 | /\* Unused values \*/ |

Table 45: Value of effectType.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Basis | |
| 1 | Reference | |
| 2 | Composite | |
| 3 | /\* Unused value \*/ |

The following sub-clauses describe the syntax and semantics used with the binary encoding to store effect data. The semantic associated with this data is detailed in Table 12 and Table 13 of clauses 6.9 and 6.10. The syntax used to read the data from the binary band varies depending on the type of effect and the type of band. Each sub-clause defines the syntax for a given effect type and band type.

# MPEG\_Haptics\_compositeEffect

Table 46 describes the syntax used to store the information of a composite effect. The associated data structure is detailed in clause 6.8.

Table 46: Syntax for reading a composite effect (MPEG\_haptics\_compositeEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_compositeEffect() |  |  |
| { |  |  | |
| **compositionEffectCount;** | 16 | **uimsbf** | |
| for (i = 0; i < **compositionEffectCount**; i++) { |  |  | |
| **effectType** | **2** | **uimsbf** | |
| **effectPosition** | **24** | **uimsbf** | |
| if (**effectType** == 1) { |  |  | |
| **effectId**; | **16** | **uimsbf** | |
| }else if(**effectType** == 2){ |  |  | |
| MPEG\_haptics\_compositeEffect(); |  |  | |
| }else{ |  |  | |
| if(**bandType** == 0){ |  |  | |
| MPEG\_haptics\_transientEffect(); |  |  | |
| }else if(**bandType** == 1){ |  |  | |
| MPEG\_haptics\_curveEffect(); |  |  | |
| }else if(**bandType** == 2){ |  |  | |
| MPEG\_haptics\_vectorialEffect(); |  |  | |
| }else if(**bandType** == 3){ |  |  | |
| MPEG\_haptics\_waveletEffect() |  |  | |
| } |  |  | |
| } |  |  | |
| } |  |  | |
| } |  |  | |

The semantics associated with this syntax is detailed in Table 47 and maps to the properties detailed in Table 12 of clause 6.9.

Table 47: Semantics associated to the MPEG\_haptics\_compositeEffect() syntax

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **compositionEffectCount** | Number of sub-effects contained in the composite effect |
| **effectType** | Type of haptic effect. Possible values are detailed in Table 45 |
| **effectPosition** | Position of the effect relative to the starting point (spatial reference or temporal reference) of the experience |
| **effectId** | Id of the effect |

# MPEG\_Haptics\_transientEffect

Table 48 describes the syntax used to store the information of a transient effect through a list of keyframes. The associated data structure is detailed in clause 6.10.

Table 48: Syntax for reading a basis effect in a curve band (MPEG\_haptics\_transientEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_transientEffect |  |  |
| { |  |  |
| **keyframeCount;** | **16** | **uimsbf** |
| for (i = 0; i < **keyframeCount**; i++) { |  |  |
| **amplitude**; | **8** | **decimal** |
| **position**; | **16** | **uimsbf** |
| **frequency**; | **16** | **uimsbf** |
| } |  |  |
| } |  |  |

The semantics associated with this syntax is detailed in Table 49 and maps to the properties detailed in Table 13 of clause 6.10.

Table 49: Semantics associated to the MPEG\_haptics\_transientEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **keyframeCount** | Number of transients stored in this effect |
| **Amplitude** | The amplitude of the transient. The range of this decimal number is [-1;1] |
| **Position** | The local position of the transient |
| **Frequency** | The frequency of the transient |

# MPEG\_Haptics\_curveEffect

Table 50 describes the syntax used to store the information of a curve effect through a list of keyframes. The associated data structure is detailed in clause 6.10

Table 50: Syntax for reading a basis effect in a curve band (MPEG\_haptics\_curveEffect()).

|  |  |  |  |
| --- | --- | --- | --- |
| Syntax | No. of bits | Mnemonic | |
| MPEG\_haptics\_curveEffect |  |  |
| { |  |  |
| **keyframeCount** | **16** | **uimsbf** |
| for (i = 0; i < **keyframeCount**; i++) { |  |  |
| **amplitude**; | **8** | **decimal** |
| **position**; | **16** | **uimsbf** |
| } |  |  |
| } |  |  |

The semantics associated with this syntax is detailed in Table 51 and maps to the properties detailed in Table 13 of clause 6.10.

Table 51: Semantics associated to the MPEG\_haptics\_curveEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **keyframeCount** | Number of control points for this curve effect |
| **Amplitude** | The amplitude of a control point. The range of this decimal number is [-1;1] |
| **Position** | The local position of a control point |

# MPEG\_Haptics\_vectorialEffect

Table 52 describes the syntax used to store the information of a vectorial effect through a list of keyframes. The associated data structure is detailed in clause 6.10.

Table 52: Syntax for reading a basis effect in a vectorial band (MPEG\_haptics\_vectorialEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_vectorialEffect |  |  |
| { |  |  |
| **phase;** | **16** | **decimal** |
| **baseSignal;** | **4** | **uimsbf** |
| **keyframeCount;** | **16** | **uimsbf** |
| for (i = 0; i < **keyframeCount**; i++) { |  |  |
| **informationMask;** | **3** | **uimsbf** |
| if ((**informationMask** & 0x01) != 0) { |  |  |
| **amplitude;** | **8** | **decimal** |
| } |  |  |
| **relativePosition;** | **16** | **uimsbf** |
| if ((**informationMask** & 0x02) != 0) { |  |  |
| **frequency;** | **16** | **uimsbf** |
| } |  |  |
| } |  |  |
| } |  |  |

The semantics associated with this syntax is detailed in Table 53 and maps to the properties detailed in Table 12 and Table 13 of clauses 6.9 and 6.10.

Table 53: Semantics associated to the MPEG\_haptics\_vectorialEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **phase** | The initial phase of the haptic effect given in radian. The range of this decimal number is [0; 6.28318]. 0 is considered as no phase and 6.28318 as one period delay. |
| **baseSignal** | It defines the type of waveform signal to be used by the synthesizer. Possible values are detailed in Table 54. |
| **keyframeCount** | Number of keyframe stored in the effect. |
| **informationMask** | Binary mask to provide if optional information are provided or not. |
| **relativePosition** | The relative position of a control point of the effect’s modulation function |
| **amplitude** | The amplitude of a control point of the effect’s amplitude modulation function. The range of this decimal number is [-1;1]. |
| **frequency** | The frequency of a keyframe of the effect’s frequency modulation function, |

Table 54: Value of baseSignal.

|  |  |  |
| --- | --- | --- |
| **Value** | **Meaning** | |
| 0 | Sine | |
| 1 | Square | |
| 2 | Triangle | |
| 3 | SawToothUp | |
| 4 | SawToothDown | |
| 5-15 | /\* Unused values \*/ |

# MPEG\_haptics\_waveletEffect

In wavelet bands, the signal is stored as a series of wavelet blocks. Each block contains the encoded data of a fixed number of samples of the original signal and each effect contains the data of a single block.

As detailed in Table 55, blocks are stored with a length value **streamsize** in bytes and the encoded bitstream **arithmetic\_stream**. The size of the **aritmethic\_stream** is given by **streamsize**. For empty wavelet blocks, which only contain zero coefficients, the **streamsize** is equal to 0 and no **arithmetic\_stream** shall be decoded.

Table 55: Syntax for reading a basis effect in a wavelet band (MPEG\_haptics\_waveletEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_waveletEffect() |  |  |
| { |  |  |
| **streamsize;** | **16** | **uimsbf** |
| **arithmetic\_stream;** | **streamsize \* 8** | **vlclbf** |
| MPEG\_haptics\_decodeWaveletEffect(**arithmetic\_stream**) |  |  |
| } |  |  |

Table 56: Semantics associated to the MPEG\_haptics\_waveletEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **streamsize** | Size of the **arithmetic\_stream** |
| **arithmetic\_stream** | Bitstream generated by the arithmetic encoder. This is the final compressed bitstream for a wavelet effect |

Inside the function MPEG\_haptics\_decodeWaveletEffect(), the **arithmetic\_stream** is decoded using the SPIHT\_decode() function detailed in clause 10.2.3, since the arithmetic decoding is embedded in the SPIHT decoding. This function allows to decode the stream and output the data necessary for the rendering: **wavmax, maxallocBits** and **wavelet\_coefficients**.

This output fits the data model structure described in clause 6.9 for wavelet effects. Table 57 illustrates how the keyframes data of each effect can be reconstructed from this output.

Table 57: Syntax for decoding a basis effect in a wavelet band (MPEG\_haptics\_decodeWaveletEffect()).

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| MPEG\_haptics\_decodeWaveletEffect (**arithmetic\_stream**) () |  |  |
| { |  |  |
| SPIHT\_decode(**arithmetic\_stream, wavmax, maxallocBits, wavelet\_coefficients**) |  |  |
| for (i = 0; i < block\_length; i++) { |  |  |
| **keyframes[i].amplitude = wavelet\_coefficients[i];** |  |  |
| } |  |  |
| **keyframes[blockLength].amplitude = wavmax;** |  |  |
| **keyframes[blockLength+1].amplitude = maxallocBits;** |  |  |
| } |  |  |

Table 58: Semantics associated to the MPEG\_haptics\_waveletEffect() syntax.

|  |  |
| --- | --- |
| ***Data*** | ***Description*** |
| **keyframes** | List of keyframes of the effect |
| **amplitude** | Amplitude of the effect storing a wavelet coefficient |
| **wavelet\_coefficients** | Array of wavelet coefficients of the current effect, scaled to [-1,1]. This is an output of the SPIHT\_decode() function |
| **wavmax** | It is the quantized absolute maximum amplitude of the wavelet block. This is an output of the SPIHT\_decode() function and needed to recover the original amplitude of the wavelet\_coefficients in the synthesizer. |
| **maxallocBits** | Maximum for quantization allocated bits in the wavelet effect. This is only used inside the SPIHT\_decode() and SPIHT\_encode() function and needed for correct scaling of the wavelet\_coefficients.This is an output of the SPIHT\_decode() function |

Each effect gets an individual position in samples. The **wavelet\_coefficients** vector, obtained from the SPIHT decoder, is scaled to **max(abs(wavelet\_coefficients[i]))==1** before being stored as a keyframes in MPEG\_haptics\_decodeWaveletEffect. From that, the original amplitude of the signal can be restored by multiplying all coefficients with **wavmax** (later also referred to as in formulas). For binary encoding, the **wavelet\_coefficients** are scaled to integer values, which is described in clause 7.1.3.

The encoder for wavelet effects is described in clause 7.1.3.1. The decoder is described in the following paragraphs.

The structure of the decoder is shown in Figure 14. In the binary decoder, first the binary stream has to be decoded block per block. This is done by the arithmetic decoder and SPIHT decoder. After applying the required formatting for the interchange format, the data sent into the binary encoder is recovered, which is the content of the descriptive format.

Then, to gain the fully decoded and synthesized signal, the original scaling is recovered by multiplying each sample with , and inverse wavelet transformation is applied to output a PCM signal with the same sampling frequency as the original signal. At this point, a series of decoded wavelet effects, already transformed back into time domain is obtained. To obtain the structure of the original signal, each effect has to be added to the output signal at the correct position in time. Since the input of the wavelet band processing was a continuous PCM file split into blocks of equal length, every wavelet effect directly follows the previous one (with the first sample directly after the last sample of the previous effect). Therefore, no additional position information is saved in the binary format. This process is called reordering in Figure 14.This decoded high frequency part is then added to the low frequency part.

Further details on the modules are provided in clause 10.2.3.

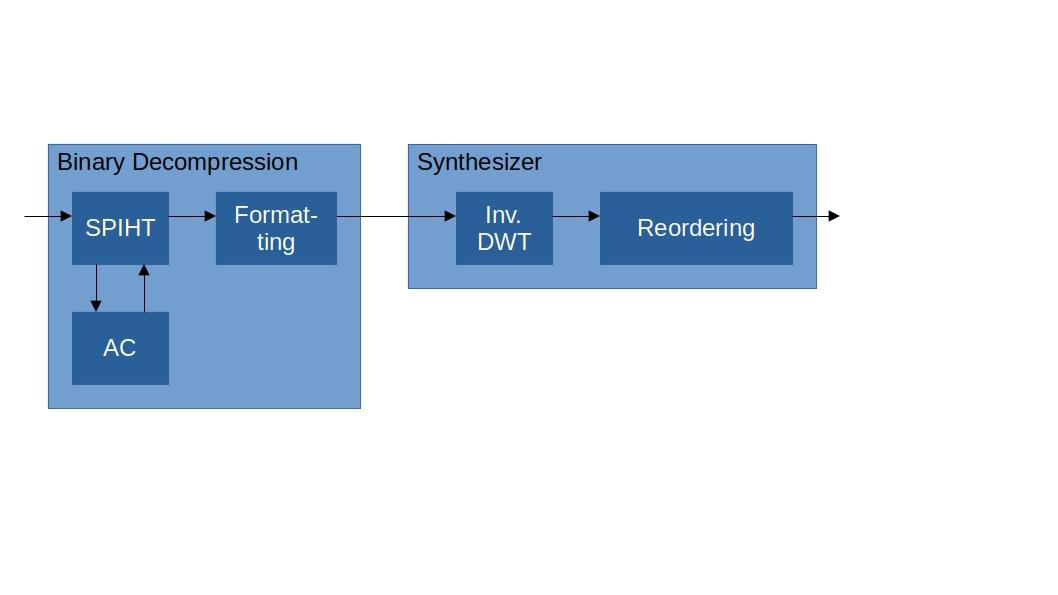


Figure 14: Wavelet bands decoder signal flow.

# MPEG-I haptic stream (MIHS) format

# Overview

This clause defines a self-contained stream format to transport MPEG-I haptic data. The transport mechanism uses a packetized approach. Different packet types are defined to transport the diverse types of information.

The packets are of variable duration and include two levels of packetization:

1. MIHS unit which covers a duration of time and includes zero or more MIHS packets.
2. MIHS packet which includes metadata or haptic effect data.

Figure 15 shows the high-level concept of MIHS packetization.



Figure 15 **—** MIHS packetization

Each MIHS unit covers a nonoverlapping duration of haptic presentation time, i.e., it starts at the end of the previous MIHS unit and covers the duration of time defined by its duration field. The MIHS unit is followed by the next MIHS unit, unless it is the last MIHS unit of the haptic experience. All MIHS packets of a MIHS unit have the starting time and duration of the containing MIHS unit.

A MIHS unit may have one of the following types:

1. **Initialization**. An initialization unit shall contain one timing MIHS packet and may include one or more metadata MIHS packets. The initialization unit starts a new time anchor in haptics presentation time and/or set or change the timestamp and/or timescale. The duration of an initialization unit duration shall be zero.
2. **Temporal**. A temporal unit shall contain one or more MIHS packets. The duration of a temporal unit shall be a positive number.
3. **Spatial**. A spatial unit shall contain one or more MIHS packets. The duration of a spatial unit shall be zero.
4. **Silent**. A silent unit indicates that there is no effect that starts during this time interval and shall not include any MIHS packets. The duration of a silent unit shall be a positive number.

The first MIHS unit in a haptic stream shall be an initialization unit.

A MIHS unit may be a sync unit or a non-sync unit:

1. A sync unit resets the previous effects and therefore provides an independent haptic experience from the previous MIHS units.
2. A non-sync unit is the continuation of previous MIHS units and cannot be independently decoded and rendered without decoding the previous MIHS unit(s).

A MIHS unit comprises the unit header and MIHS packets as shown in Figure 16. Names in the figure are shortened with respect to their syntax counterparts for easier reading.



Figure 16 **—** MIHS unit structure

# Initialization units

An initialization unit is a sync unit and shall have the following constraints:

* Exactly one packet of type PACTYPE\_TIMING
* Zero or more packets of the following types:
  + PACTYPE\_METADATAEXPERIENCE
  + PACTYPE\_METADATAPERCEPTION
  + PACTYPE\_METADATACHANNEL
  + PACTYPE\_METADATABAND
  + PACTYPE\_LIBRARYEFFECTS
* Zero or more packets of the following types:
  + PACTYPE\_CRC16
  + PACTYPE\_CRC32
  + PACTYPE\_GlobalCRC16
  + PACTYPE\_GlobalCRC32
* No MIHS packet of the following types:
  + PACTYPE\_DATA

The sync field shall be ignored for an initialization unit since it is a sync unit.

The duration field shall be ignored for an initialization unit since it does not have a duration.

# Temporal and spatial units

Temporal and spatial units shall have the following constraints:

* One or more packets of the following types:
  + PACTYPE\_DATA
* Zero or more packets of the following types:
  + PACTYPE\_CRC16
  + PACTYPE\_CRC32
  + PACTYPE\_GlobalCRC16
  + PACTYPE\_GlobalCRC32
* No packets of following types:
  + PACTYPE\_TIMING
  + PACTYPE\_METADATAEXPERIENCE
  + PACTYPE\_METADATAPERCEPTION
  + PACTYPE\_METADATACHANNEL
  + PACTYPE\_METADATABAND
  + PACTYPE\_LIBRARYEFFECTS

Temporal units shall have the following additional constraint:

* Only data for the following perception modalities:
  + Pressure
  + Acceleration
  + Velocity
  + Position
  + Temperature
  + Vibrotactile
  + Water
  + Wind
  + Force
  + Other

Spatial units shall have the following additional constraint:

* Only data for the following perception modalities:
  + Vibrotactile texture
  + Stiffness
  + Friction

A perception may include temporal or spatial units but not both.

The sync field shall be ignored for a spatial unit since it is a sync unit by definition.

The duration field shall be ignored for a spatial unit since it does not have a duration by definition.

# MIHS packets

All packets follow the structure shown in Figure 17. Names in the figure are shortened with respect to their syntax counterparts for easier reading. The packet header is common to all **MIHSPacketType** values. The payload structure varies based on the **MIHSPacketType**.



Figure 17 **—** MIHS packet structure

A MIHS packet shall have one of the following types:

1. **Timing**. Defines a timestamp and timescale for subsequent MIHS packets.
2. **MetadataExperience**. Contains metadata for the haptic experience.
3. **MetadataPerception**. Contains metadata for a haptic perception.
4. **MetdataChannel**. Contains metadata for a haptic channel.
5. **MetadataBand**. Contains metadata for a haptic band.
6. **Data**. Contains haptic effect data.
7. **EffectLibrary**. Contains a library of haptic effects that haptic effect data may reference.
8. **CRC16** or **CRC32**. Contains a CRC value which applies to the directly following MIHS packet.
9. **GlobalCRC16** or **GlobalCRC32**. Contains a CRC value which applies to one or more directly following MIHS packets.

# Syntax and semantics

# mpegiHapticStream()

Table 59 **—** Syntax of mpegiHapticStream()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| mpegiHapticStream() |  |  |
| { |  |  |
| while (bitsAvalaible() != 0) { |  |  |
| mpegiHapticUnit(); |  |  |
| } |  |  |
| } |  |  |

# mpegiHapticUnit()

Table 60 **—** Syntax of mpegiHapticUnit()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| mpegiHapticUnit() |  |  |
| { |  |  |
| **MIHSUnitType;** | 6 | uimsbf |
| **MIHSUnitSync;** | 2 | uimsbf |
| **MIHSUnitDuration;** | 8 | uimsbf |
| **MIHSUnitLength;** | 32 | uimsbf |
| while (packetsAvalaible() != 0) { |  |  |
| mpegiHapticPacket(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **MIHSUnitType** | MIHS unit type. The possible values for **MIHSUnitType** are listed in Table 61. Decoders shall skip units with an unknown **MIHSUnitType**. |
| Table 61 **—** Value of MIHSUnitType   |  |  | | --- | --- | | MIHSUnitType | Value | | UNITTYPE\_INITIALIZATION | 0 | | UNITTYPE\_TEMPORAL | 1 | | UNITTYPE\_SPATIAL | 2 | | UNITTYPE\_SILENT | 3 | | */\* unused values \*/* | 4-63 | | |
| **MIHSUnitSync** | Flag indicating whether the MIHS unit is independently interpretable. Table 62 lists the possible values for MIHSUnitSync. |
| Table 62 **—** Value of MIHSUnitSync   |  |  | | --- | --- | | Value | Meaning | | 0 | Independent | | 1 | Dependent | | 2-3 | */\* unused values \*/* | | |
| **MIHSUnitDuration** | Duration of the MIHS unit. All the MIHS packets included in the unit occur in this timespan. **MIHSUnitDuration** must be a power of two. **MIHSUnitDuration** must be zero for **MIHSUnitType** UNITTYPE\_INITIALIZATION and UNITTYPE\_SPATIAL and greater than zero for **MIHSUnitType** UNITTYPE\_TEMPORAL and UNITTYPE\_SILENT. The duration is in the **timescale** of the readMetadataTiming() structure in the most recently sent mpegHapticUnit() with **MIHSUnitType** UNITTYPE\_INITIALIZATION. |
| **MIHSUnitLength** | Total length in bytes of the MIHS unit's MIHS packets (mpegiHapticPacket() structures). |
| mpegiHapticPacket() | MIHS packet structure defined in subclause 9.2.3. |

# mpegiHapticPacket()

Table 63 **—** Syntax of mpegiHapticPacket()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| mpegiHapticPacket() |  |  |
| { |  |  |
| **MIHSPacketType;** | 6 | uimsbf |
| **MIHSLayer;** | 4 | uimsbf |
| **Residual;** | 5 | uimsbf |
| **MIHSPacketLength;** | 17 | uimsbf |
| MIHSPacketPayload(MIHSPacketType); |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **MIHSPacketType** | Payload type in the packet. Table 64 lists the possible values for **MIHSPacketType**. Subclause 9.3 provides further details. Decoders shall skip packets with an unknown **MIHSPacketType**. |
| Table 64 **—** Value of MIHSPacketType   |  |  | | --- | --- | | MIHSPacketType | Value | | PACTYPE\_TIMING | 0 | | PACTYPE\_METADATAEXPERIENCE | 1 | | PACTYPE\_METADATAPERCEPTION | 2 | | PACTYPE\_METADATACHANNEL | 3 | | PACTYPE\_METADATABAND | 4 | | PACTYPE\_DATA | 5 | | PACTYPE\_LIBRARYEFFECTS | 6 | | PACTYPE\_CRC16 | 7 | | PACTYPE\_CRC32 | 8 | | PACTYPE\_GlobalCRC16 | 9 | | PACTYPE\_GlobalCRC32 | 10 | | */\* unused values \*/* | 11-15 | | |
| **MIHSLayer** | Importance of the packet. **MIHSLayer** for scalability. Given a limited bandwidth, decoders may ignore packets with a high layer value. |
| **Residual** | This value is unused; decoders should ignore it. |
| **MIHSPacketLength** | Length of the payload in bytes. |
| MIHSPacketPayload() | Payload for the MIHS packet defined in subclause 9.2.4. |

# MIHSPacketPayload()

Table 65 **—** Syntax of MIHSPacketPayload()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| mpegiPacketPayload(MIHSPacketType) |  |  |
| { |  |  |
| switch (MIHSPacketType) { |  |  |
| case PACTYPE\_TIMING: |  |  |
| readMetadataTiming(); |  |  |
| break; |  |  |
| case PACTYPE\_METADATAEXPERIENCE: |  |  |
| readMetadataExperience(); |  |  |
| break; |  |  |
| case PACTYPE\_METADATAPERCEPTION: |  |  |
| readMetadataPerception(); |  |  |
| break; |  |  |
| case PACTYPE\_METADATACHANNEL: |  |  |
| readMetadataChannel(); |  |  |
| break; |  |  |
| case PACTYPE\_METADATABAND: |  |  |
| readMetadataBand(); |  |  |
| break; |  |  |
| case PACTYPE\_DATA: |  |  |
| readData (); |  |  |
| break; |  |  |
| case PACTYPE\_LIBRARYEFFECTS: |  |  |
| readLibrary(); |  |  |
| break; |  |  |
| case PACTYPE\_CRC16: |  |  |
| case PACTYPE\_CRC32: |  |  |
| case PACTYPE\_GlobalCRC16: |  |  |
| case PACTYPE\_GlobalCRC32: |  |  |
| readCRC(); |  |  |
| break; |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| readMetadataTiming() | Metadata related to the timing of subsequent MIHS packets, described in subclause 9.2.5. |
| readMetadataExperience() | Metadata of the haptic experience being streamed, described in subclause 9.2.6. |
| readMetadataPerception() | Metadata of a perception of the haptic experience, described in subclause 9.2.8. |
| readMetadataChannel() | Metadata of a channel of the haptic experience, described in subclause 9.2.10. |
| readMetadataBand() | Metadata of a band of the haptic experience, described in subclause 9.2.11. |
| readData() | Effects for a band of the haptic experience, described in subclause 9.2.14. |
| readLibrary() | Predefined effects to be referenced from the bands of the haptic experience, described in subclause 9.2.12. |
| readCRC() | Cyclic redundancy check value used to protect one or more directly following packets, described in subclause 9.2.22. |

# readMetadataTiming()

Table 66 **—** Syntax of readMetadataTiming()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readMetadataTiming() |  |  |
| { |  |  |
| **timestamp;** | 32 | uimsbf |
| **timescale;** | 32 | uimsbf |
| } |  |  |

|  |  |
| --- | --- |
| **timestamp** | Timestamp of the haptic experience in ticks, i.e., **timestamp**/**timescale** is the timestamp in seconds. |
| **timescale** | Number of ticks per second. |

# readMetadataExperience()

Table 67 **—** Syntax of readMetadataExperience()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readMetadataExperience() |  |  |
| { |  |  |
| **versionLength;** | 8 | uimsbf |
| **version;** | versionLength\*8 | string |
| **dateLength;** | 8 | uimsbf |
| **date;** | dateLength\*8 | string |
| **descriptionLength;** | 8 | uimsbf |
| **description;** | descriptionLength\*8 | string |
| **perceptionCount;** | 8 | uimsbf |
| **avatarCount;** | 8 | uimsbf |
| for (I = 0; I < avatarCount; i++) { |  |  |
| readAvatar(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **versionLength** | Number of chars in string **version**. |
| **version** | List of chars representing the version of this ISO/IEC 23090-31 specification. |
| **dataLength** | Number of chars in string **date**. |
| **date** | List of chars representing the creation date of the haptic experience in human-readable form. The date format shall follow the ISO 8601 standard. |
| **descriptionLength** | Number of chars in string **description**. |
| **description** | List of chars representing a description of the haptic experience. |
| **perceptionCount** | Number of perceptions in the haptic experience. |
| **avatarCount** | Number of avatars in the haptic experience. |
| readAvatar() | Avatar object defined in subclause 9.2.7. |

# readAvatar()

Table 68 **—** Syntax of readAvatar()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readAvatar() |  |  |
| { |  |  |
| **id**; | 8 | uimsbf |
| **lod**; | 8 | uimsbf |
| **avatarType**; | 4 | uimsbf |
| if (avatarType == 0) { |  |  |
| **meshLength**; | 8 | uimsbf |
| **mesh**; | meshLength\*8 | string |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the avatar. |
| **lod** | Level of detail of the avatar. |
| **avatarType** | Type of haptic perception represented by the avatar. The possible values for **avatarType** are listed in Table 69. |
| Table 69 **—** Value of avatarType   |  |  | | --- | --- | | Value | Meaning | | 0 | Custom | | 1 | Vibration | | 2 | Pressure | | 3 | Temperature | | 4-15 | */\* unused values \*/* | | |
| **meshLength** | Number of chars in the string **mesh.** Only present if **avatarType** is Custom. |
| **mesh** | URI to the custom mesh file. Only present if **avatarType** is Custom. |

# readMetadataPerception()

Table 70 **—** Syntax of readMetadataPerception()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readMetadataPerception() |  |  |
| { |  |  |
| **id**; | 8 | uimsbf |
| **descriptionLength**; | 8 | uimsbf |
| **description**; | descriptionLength\*8 | string |
| **perceptionModality**; | 8 | uimsbf |
| **avatarId** | 8 | uimsbf |
| **effectLibraryCount**; | 16 | uimsbf |
| **unitExponent**; | 8 | uimsbf |
| **perceptionUnitExponent**; | 8 | uimsbf |
| **referenceDeviceCount**; | 8 | uimsbf |
| for (i = 0; i < referenceDeviceCount; i++) { |  |  |
| readReferenceDevice(); |  |  |
| } |  |  |
| **channelCount**; | 8 | uimsbf |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the perception. |
| **descriptionLength** | Number of chars in the string **description**. |
| **description** | Description of the perception. |
| **perceptionModality** | Type of perception represented. Table 71 lists the possible values for **perceptionModality** . |
| Table 71 **—** Value of perceptionModality   |  |  | | --- | --- | | Value | Meaning | | 0 | Other | | 1 | Pressure | | 2 | Acceleration | | 3 | Velocity | | 4 | Position | | 5 | Temperature | | 6 | Vibrotactile | | 7 | Water | | 8 | Wind | | 9 | Force | | 10 | Vibrotactile texture | | 11 | Electrotactile | | 12 | Stiffness | | 13 | Friction | | 14-255 | */\* unused values \*/* | | |
| **avatarId** | Unique identifier of the associated avatar body model defined in subclause 9.2.7 |
| **effectLibraryCount** | Number of effects in the perception's effect library. |
| **unitExponent** | Refers to the 10*x* exponent for the SI unit of the independent variable. |
| **perceptionUnitExponent** | Refers to the 10*x* exponent for the SI unit of the dependent variable (see Table 4). |
| **referenceDeviceCount** | Number of reference devices associated with the perception. |
| readReferenceDevice() | Reference device structure described in subclause 9.2.9. |
| **channelCount** | Number of channels in the perception. |

# readReferenceDevice()

Table 72 — Syntax of readReferenceDevice()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readReferenceDevice() |  |  |
| { |  |  |
| **id**; | 8 | uimsbf |
| **nameLength**; | 8 | uimsbf |
| **name**; | nameLength \*8 | string |
| **bodyPartMask**; | 32 | uimsbf |
| **optionalFieldMask**; | 12 | uimsbf |
| if (optionalFieldMask & 0x00’01) { |  |  |
| **maximumFrequency**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’02) { |  |  |
| **minimumFrequency**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’04) { |  |  |
| **resonanceFrequency**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’08) { |  |  |
| **maximumAmplitude**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’10) { |  |  |
| **impedance**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’20) { |  |  |
| **maximumVoltage**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’40) { |  |  |
| **maximumCurrent**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x00’80) { |  |  |
| **maximumDisplacement**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x01’00) { |  |  |
| **weight**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x02’00) { |  |  |
| **size**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x04’00) { |  |  |
| **custom**; | 32 | decimal |
| } |  |  |
| if (optionalFieldMask & 0x08’00) { |  |  |
| **type**; | 4 | uimsbf |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the reference device. |
| **nameLength** | Number of chars in the string **name**. |
| **name** | Name of the reference device. |
| **bodyPartMask** | Binary mask specifying the location of the device on the human body according to subclause 6.6 and Table 7. |
| **optionalFieldMask** | Binary mask for the optional parameters of the reference device. |
| **maximumFrequency** | Maximum frequency of the actuator in Hertz. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’01 is true. |
| **minimumFrequency** | Minimum frequency of the actuator in Hertz. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’02 is true. |
| **resonanceFrequency** | Resonance frequency of the actuator in Hertz. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’04 is true. |
| **maximumAmplitude** | Maximum amplitude of the device according to the perception\_modality. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’08 is true. |
| **impedance** | Impedance of the actuator in Ohms. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’10 is true. |
| **maximumVoltage** | Maximum voltage of the actuator. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’20 is true. |
| **maximumCurrent** | Maximum current of the actuator in Amperes. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’40 is true. |
| **maximumDisplacement** | Maximum displacement of the actuator in millimeters. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x00’80 is true. |
| **weight** | Weight of the device in kilograms. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x01’00 is true. |
| **size** | Size of the device in millimeters. The range of this decimal number is [0; 10000]. Present only if **optionalFieldMask** & 0x02’00 is true. |
| **custom** | Custom data. The range of this decimal number is [-10000; 10000]. Present only if **optionalFieldMask** & 0x04’00 is true. |
| **type** | Type of actuator, present only if **optionalFieldMask** & 0x08’00 is true. Table 73 lists the possible values for **type**. |

Table 73 **—** Value of type

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | Unknown |
| 1 | LRA |
| 2 | VCA |
| 3 | ERM |
| 4 | Piezo |
| 5-15 | */\* unused values \*/* |

# readMetadataChannel()

Table 74 — Syntax of readMetadataChannel()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readMetadataChannel() |  |  |
| { |  |  |
| **id**; | 8 | uimsbf |
| **perceptionId;** | 8 | uimsbf |
| **descriptionLength**; | 8 | uimsbf |
| **description**; | descriptionLength\*8 | string |
| **deviceId**; | 8 | uimsbf |
| **gain**; | 32 | decimal |
| **mixingWeight**; | 32 | decimal |
| **optionalMetadataMask;** | 8 | uimsbf |
| if ((optionalMetadataMask & 0x01) != 0) { |  |  |
| **bodyPartMask**; | 32 | uimsbf |
| } |  |  |
| if ((optionalMetadataMask & 0x02) != 0) { |  |  |
| **trackResolution.X;** | 8 | imsbf |
| **trackResolution.Y;** | 8 | imsbf |
| **trackResolution.Z;** | 8 | imsbf |
| **bodyPartTargetCount;** | 8 | uimsbf |
| for (i = 0; i < bodyPartTargetCount; i++) { |  |  |
| **bodyPartTarget[i];** | 8 | uimsbf |
| } |  |  |
| **actuatorTargetCount;** | 8 | uimsbf |
| for (i = 0; I < actuatorTargetCount; i++) { |  |  |
| **actuatorTarget[i].X** | 8 | imsbf |
| **actuatorTarget[i].Y** | 8 | imsbf |
| **actuatorTarget[i].Z** | 8 | imsbf |
| } |  |  |
| } |  |  |
| **frequencySampling**; | 32 | uimsbf |
| if (frequencySampling > 0) { |  |  |
| **sampleCount**; | 32 | uimsbf |
| } |  |  |
| if ((optionalMetadataMask & 0x04) != 0) { |  |  |
| **direction.X**; | 8 | decimal |
| **direction.Y**; | 8 | decimal |
| **direction.Z**; | 8 | decimal |
| } |  |  |
| **verticesCount**; | 16 | uimsbf |
| for (i = 0; i < verticesCount; i++) { |  |  |
| **vertex**; | 32 | uimsbf |
| } |  |  |
| **bandCount**; | 16 | uimsbf |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the channel. |
| **perceptionId** | ID of the perception to which the channel is attached. |
| **descriptionLength** | Number of chars in the string **description**. |
| **description** | Description of the channel. |
| **deviceId** | ID of the associated device. |
| **gain** | Gain associated with the channel. |
| **mixingWeight** | Mixing weight of the channel. |
| **optionalMetadataMask** | Binary mask to define if optional information is stored or not. |
| **bodyPartMask** | Binary mask specifying body parts on which to apply the effect body according to subclause 6.6 and Table 7. |
| **trackResolution** | Reference actuator resolution used to design the haptic experience for each spatial coordinate (X, Y, Z) |
| **bodyPartTargetCount** | Number of bodyPartTarget in the channel. |
| **bodyPartTarget** | Semantic identification of a unique and/or group of body parts on the human body |
| **actuatorTargetCount** | Number of actuatorTarget in the channel. |
| **actuatorTarget** | List of different actuators targeted by the channel and identified by its coordinates (X, Y, Z) |
| **frequencySampling** | Sampling frequency of the original encoded signal in Hertz. |
| **sampleCount** | Sample count of the original encoded signal. Present only if **frequencySampling** is greater than 0. |
| **directionFlag** | Flag indicating the presence of a **direction**. |
| **direction** | Spatial direction of the encoded signal. This value is based on a local representation of the 3-dimensional space given by the presentation engine |
| **verticesCount** | Number of vertices in the list **vertices**. |
| **vertices** | List of vertices on the avatar representation affected by the effect. |
| **bandCount** | Number of bands associated with the channel. |

# readMetadataBand()

Table 75 — Syntax of readMetadataBand()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readMetadataBand() { |  |  |
| **id;** | 8 | uimsbf |
| **perceptionId;** | 8 | uimsbf |
| **channelId;** | 8 | uimsbf |
| **bandType;** | 3 | uimsbf |
| if (bandType == 1) { |  |  |
| **curveType;** | 4 | uimsbf |
| } else if(bandType == 3) { |  |  |
| **blockLengthLog;** | 8 | uimsbf |
| } |  |  |
| **lowerFrequency;** | 16 | decimal |
| **upperFrequency;** | 16 | decimal |
| **effectsCount;** | 16 | uimsbf |

|  |  |
| --- | --- |
| **id** | ID of the band. |
| **perceptionId** | ID of the perception to which the band belongs. |
| **channelId** | ID of the channel to which the band belongs. |
| **bandType** | Type of band. Table 76 lists the possible values for **bandTtype**. |
| Table 76 **—** Value of bandType   |  |  | | --- | --- | | Value | Meaning | | 0 | Transient | | 1 | Curve | | 2 | VectorialWave | | 3 | WaveletWave | | 4-7 | */\* unused values \*/* | | |
| **curveType** | Interpolation function to use. Table 77 lists the possible values for **curveType** . |
| Table 77 **—** Value of curveType   |  |  | | --- | --- | | Value | Meaning | | 0 | Unknown | | 1 | Cubic | | 2 | Linear | | 3 | Akima | | 4 | Bezier | | 5 | BSpline | | 6-15 | */\* unused values \*/* | | |
| **blockLengthLog** | The **blockLength** is coded in samples instead of milliseconds for the binary format, allowing a more compact representation. Additionally, it is transformed to the logarithmic domain using the formula  This leads to an allowed minimum **blockLengthLog** of 16 and only powers of 2 in samples, which is required for wavelet coding. |
| **lowerFrequency** | Lower frequency limit of the band in Hertz. The range of this decimal number is [0; 10000]. |
| **upperFrequency** | Upper frequency limit of the band in Hertz. The range of this decimal number is [0; 10000]. |
| **effectsCount** | Number of effects present in the band. |

# readLibrary()

Table 78 — Syntax of readLibrary()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readLibrary() { |  |  |
| **perceptionId;** | 8 | uimsbf |
| **effectCount**; | 16 | uimsbf |
| for (i = 0; i < effectCount; i++) { |  |  |
| readLibraryEffect() |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **perceptionId** | ID of the perception to which the library is attached. |
| **effectCount** | Number of effects in the library. |
| readLibraryEffect() | Effect structure as described in subclause 9.2.13. |

# readLibraryEffect()

Table 79 — Syntax of readLibraryEffects()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readLibraryEffect() { |  |  |
| **id;** | 16 | uimsbf |
| **position**; | 25 | imsbf |
| **phase**; | 16 | uimsbf |
| **baseSignal** | 4 | uimsbf |
| **effectType**; | 2 | uimsbf |
| **keyframesCount**; | 16 | uimsbf |
| for (i = 0; i < keyframesCount; i++) { |  |  |
| **mask;** | 3 | uimsbf |
| if (mask & 0x001) { |  |  |
| **relativePosition**; | 16 | uimsbf |
| } |  |  |
| if (mask & 0x010) { |  |  |
| **amplitude**; | 8 | decimal |
| } |  |  |
| if (mask & 0x100) { |  |  |
| **frequency**; | 16 | uimsbf |
| } |  |  |
| } |  |  |
| **compositeEffectCount**; | 16 | uimsbf |
| for (i = 0; i < compositeEffectCount; i++) { |  |  |
| readLibraryEffect(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the effect in the library. |
| **position** | Position of the effect, relative to the position of the calling effect. |
| **phaseS** | Phase of the effect. |
| **baseSignal** | Base signal of the effect. Table 80 lists the possible values for **baseSignal**. |
| Table 80: Value of baseSignal   |  |  | | --- | --- | | Value | Meaning | | 0 | Sine | | 1 | Square | | 2 | Triangle | | 3 | SawToothUp | | 4 | SawToothDown | | 5-15 | */\* Unused values \*/* | | |
| **effectType** | Type of the effect. Table 81 lists the possible values for **effectType.** |
| Table 81: Value of effectType   |  |  | | --- | --- | | Value | Meaning | | 0 | Basis | | 1 | Reference | | 2 | Composite | | 3 | */\* Unused value \*/* | | |
| **keyframesCount** | Number of keyframes in the effect. |
| **mask** | Information mask indicating which keyframe parameters are present. |
| **relativePosition** | Keyframe position relative to the effect position, present only if **mask** & 0x001 is true. |
| **amplitude** | Keyframe amplitude, present only if **mask** & 0x010 is true. |
| **frequency** | Keyframe frequency in Hertz, present only if **mask** & 0x100 is true. |
| **compositeEffectCount** | Number of effects comprising the composite effect. |
| readLibraryEffect() | Recursive structure if the effect is a composite effect and contains other effects. |

# readData()

Table 82 — Syntax of readData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readData() { |  |  |
| **packetDependency**; | 1 | bool |
| **perceptionId**; | 8 | uimsbf |
| **channelId**; | 8 | uimsbf |
| **bandId**; | 8 | uimsbf |
| **effectsCount**; | 16 | uimsbf |
| if (bandType != 3) { |  |  |
| for (i = 0; i < effectsCount; i++) { |  |  |
| readEffect(); |  |  |
| } |  |  |
| } else { |  |  |
| readWaveletEffect(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **packetDependency** | Type of packet. If 1, the packet depends on one or more previous packets to be interpreted. If 0, the packet can be interpreted independently of any previous packets. |
| **perceptionId** | ID of the perception associated with the packet. |
| **channelId** | ID of the channel associated with the packet. |
| **bandId** | ID of the band associated with the packet. |
| **effectsCount** | Number of effects in the packet. |
| readEffect() | Effect structure described in clause 9.2.15. |
| readWaveletEffect() | Wavelet effect structure described in clause 9.2.21. |

# readEffect()

Table 83 — Syntax of readEffect()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readEffect() { |  |  |
| **id**; | 16 | uimsbf |
| **effectType**; | 1 | uimsbf |
| **effectPosition**; | 25 | imsbf |
| if (effectType == 0) { |  |  |
| readEffectBasis(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the effect. If the effect is of type Reference, this ID allows to retrieve the effect in the effect library. |
| **effectType** | Type of the effect. Table 84 lists the possible values for **effectType**. |
| Table 84 — Value of effectType   |  |  | | --- | --- | | Value | Meaning | | 0 | Basis | | 1 | Reference | | |
| **effectPosition** | Effect position relative to the packet timestamp for temporal data. For spatial data the effect position is relative to the origin. |
| readEffectBasis() | Keyframes of the, described in subclause 9.2.16 and present only if **effectType** is 0 (Basis). |

# readEffectBasis()

Table 85 — Syntax of readEffectBasis()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readEffectBasis() { |  |  |
| **keyframesCount**; | 16 | uimsbf |
| if (bandType == 2) { |  |  |
| **phase**; | 16 | decimal |
| **baseSignal**; | 4 | uimsbf |
| } |  |  |
| for (i = 0; i < keyframesCount; i++) { |  |  |
| readKeyframe(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **keyframesCount** | Number of keyframes in the packet belonging to the current effect. |
| **phase** | Phase of the effect. Present only if **bandType** == 2 (VectorialWave). |
| **baseSignal** | Base signal of the effect. The possible values for baseSignal are listed in Table 86. Present only if **bandType** == 2 (VectorialWave). |
| Table 86 — Value of BaseSignal   |  |  | | --- | --- | | Value | Meaning | | 0 | Sine | | 1 | Square | | 2 | Triangle | | 3 | SawToothUp | | 4 | SawToothDown | | 5-15 | */\* unused values \*/* | | |
| readKeyframe() | Keyframes in the packet belonging to the current effect, described in subclause 9.2.17. |

# readKeyframe()

Table 87 — Syntax of readKeyframe()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readKeyframe() { |  |  |
| if (bandType == 0) { |  |  |
| readTransientKeyframe(); |  |  |
| } else if (bandType == 1) { |  |  |
| readCurveKeyframe(); |  |  |
| } else if (bandType == 2) { |  |  |
| readVectorialKeyframe(); |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| readTransientKeyframe() | Transient effect keyframes, described in subclause 9.2.18.. |
| readCurveKeyframe() | Curve keyframes, described in subclause 9.2.19.. |
| readVectorialKeyframe() | Vectorial wave keyframes, described in subclause 9.2.20. |

# readTransientKeyframe()

Table 88 — Syntax of readTransientKeyframe()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readTransientKeyframe() { |  |  |
| **amplitude**; | 8 | decimal |
| **position**; | 16 | uimsbf |
| **frequency**; | 16 | uimsbf |
| } |  |  |

|  |  |
| --- | --- |
| **amplitude** | Amplitude of the keyframe. |
| **position** | Keyframe position relative to effect position. |
| **frequency** | Frequency of the keyframe. |

# readCurveKeyframe()

Table 89 — Syntax of readCurveKeyframe()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readCurveKeyframe() { |  |  |
| **amplitude**; | 8 | decimal |
| **position**; | 16 | uimsbf |
| } |  |  |

|  |  |
| --- | --- |
| **amplitude** | Amplitude of the keyframe. |
| **position** | Keyframe position relative to effect position. |

# readVectorialKeyFrame()

Table 90 — Syntax of readVectorialKeyframe()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readVectorialKeyframe() { |  |  |
| **informationMask**; | 2 | uimsbf |
| if (informationMask & 0x01) { |  |  |
| **amplitude**; | 8 | decimal |
| } |  |  |
| **position**; | 16 | uimsbf |
| if (informationMasl & 0x02) { |  |  |
| **frequency**; | 16 | uimsbf |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **informationMask** | Information mask indicating which parameters are present. |
| **amplitude** | Amplitude of the keyframe. Present only if **informationMask** & 0x01 is true. |
| **position** | Keyframe position relative to effect position. |
| **frequency** | Relative frequency of the keyframe. Present only if **informationMask** & 0x02 is true. |

# readWaveletEffect()

Table 91 — Syntax of readWaveletEffect()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readWaveletEffect() |  |  |
| { |  |  |
| **id;** | 16 | uimsbf |
| **effectType;** | 2 | uimsbf |
| MPEG\_haptics\_decodeWaveletEffect(); |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **id** | ID of the decoded effect. |
| **effectType** | Type of the effect, either Basis effect if 0, or Reference if 1. |
| MPEG\_haptics\_decodeWaveletEffect() | Wavelet block structure described in subclause 8.2.6. |

# readCRC()

Table 92 — Syntax of readCRC()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| readCRC() |  |  |
| { |  |  |
| if (MIHSPacketType == PACTYPE\_CRC16) { |  |  |
| **CRC16Value;** | 16 | uimsbf |
| } else if (MIHSPacketType == PACTYPE\_CRC32) { |  |  |
| **CRC32Value;** | 32 | uimsbf |
| } else if (MIHSPacketType == PACTYPE\_GlobalCRC16) { |  |  |
| **protectedPacketsCount;** | 8 | uimsbf |
| **CRC16Value;** | 16 | uimsbf |
| } else if (MIHSPacketType == PACTYPE\_GlobalCRC16) { |  |  |
| **protectedPacketsCount;** | 8 | uimsbf |
| **CRC32Value;** | 32 | uimsbf |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **CRC16Value** | The cyclic redundancy check value computed using the polynomial |
|  | |
| **CRC32Value** | The cyclic redundancy check value computed using the polynomial |
|  | |
| **protectedPacketsCount** | Number of packets protected by the CRC check. Present only if **MIHSPacketType** is PACTYPE\_GlobalCRC16 or PACTYPE\_GlobalCRC32. |

# Description of MIHSPacketType

# Timing

The MIHSPacketType PACTYPE\_TIMING identifies a timing MIHS packet which embeds a structure, readMetadataTiming() defined in subclause 9.2.5, in the MIHSPacketPayload() defined in subclause 9.2.4, containing the timestamp of haptic effects in ticks, and the timescale which defines the number of ticks per second.

A timing MIHS packet shall be included in an initialization MIHS packet.

# MetadataHaptics

The MIHSPacketType PACTYPE\_METADATAEXPERIENCE identifies a haptic experience metadata MIHS packet which embeds a haptic experience structure, readMetadataExperience() defined in subclause 9.2.6, in the MIHSPacketPayload() defined in subclause 9.2.4.

A haptic experience metadata MIHS packet may be sent at regular intervals in an initialization MIHS unit to enable random access of the haptic data.

Figure 18 illustrates details of the haptic experience metadata MIHS packet payload.

Diagram, table

Description automatically generated

Figure 18 - MIHSPacketPayload structure for a packet type PACTYPE\_METADATAEXPERIENCE

# MetadataPerception

The MIHSPacketType PACTYPE\_METADATAPERCEPTION identifies a haptic perception metadata MIHS packet which embeds a haptic perception structure, readMetadataPerception() defined in subclause 9.2.8, in the MIHSPacketPayload() defined in subclause 9.2.4.

Haptic perception metadata MIHS packets may be sent at regular intervals in an initialization MIHS unit to enable random access of the haptic data.

Figure 19 illustrates details of the haptic perception metadata MIHS packet payload.

Diagram

Description automatically generated

Figure 19 - MIHSPacketPaylaod structure for packet type PACTYPE\_METADATAPERCEPTION

# MetadataChannel

The MIHSPacketType PACTYPE\_METADATACHANNEL identifies a haptic channel metadata MIHS packet which embeds a haptic channel structure, readMetadataChannel() defined in subclause 9.2.10, in the MIHSPacketPayload() defined in subclause 9.2.4.

Haptic channel metadata MIHS packets may be sent at regular intervals in an initialization MIHS unit to enable random access of the haptic data.

Figure 20 illustrates details of the haptic channel metadata MIHS packet payload.

Diagram

Description automatically generated

Figure 20 - MIHSPacketPayload structure for packet type PACTYPE\_METADATACHANNEL

# MetadataBand

The MIHSPacketType PACTYPE\_METADATABAND identifies a haptic band metadata MIHS packet which embeds a haptic band structure, readMetadataBand() defined in subclause 9.2.11, in the MIHSPacketPayload() defined in subclause 9.2.4.

Haptic band metadata MIHS packets may be sent at regular intervals in an initialization MIHS unit to enable random access of the haptic data.

Figure 21 illustrates details of the haptic band metadata MIHS packet payload.

Diagram

Description automatically generated

Figure 21— MIHSPacketPayload structure for packet type PACTYPE\_METADATABAND

# LibraryEffect

The MIHSPacketType PACTYPE\_LIBRARYEFFECTS identifies a haptic effect library MIHS packet which embeds a haptic effect library structure, readLibrary() defined in subclause 9.2.12, in the MIHSPacketPayload() defined in subclause 9.2.4 .

A haptic effect library MIHS packet may be sent at regular intervals in an initialization MIHS unit to enable random access of the haptic data. The effect library may change dynamically during the haptic experience.

Figure 22 illustrates details of the haptic effect library MIHS packet payload.

A picture containing table

Description automatically generated

Figure 22— MIHSPacketPayload structure for packet type PACTYPE\_LIBRARYEFFECTS

# Data

The MIHSPacketType PACTYPE\_DATA identifies a haptic data MIHS packet which embeds data for effects and keyframes using a haptic data structure, readData() defined in subclause 9.2.14, in the MIHSPacketPayload() defined in subclause 9.2.4.

Haptic data MIHS packets contain the haptic data of the experience being streamed. The structure depends on the type of band and the effect type. The **packetDependency** flag indicates whether the packet is independent or whether the packet needs information from the previous packet.

A MIHS packet may include one or more effects as shown in Figure 23.

Table

Description automatically generated

Figure 23 **—** Example of a MIHS packet

Figure 24 illustrates details of the packet payload.

Diagram

Description automatically generated

Figure 24— MIHSPacketPayload structure for packet type PACTYPE\_DATA

# CRC16 & CRC32

The MIHSPacketType PACTYP\_CRC16 or PACTYP\_CRC32 identifies a CRC MIHS packet which may be used to detect errors in the subsequent MIHS packet including both the MIHSPacketHeader and MIHSPacketPayload portions of the packet. CRC MIHS packets contain a readCRC() structure defined in subclause 9.2.22 in the MIHSPacketPayload() defined in subclause 9.2.4.

A CRC MIHS packet shall be directly followed by the MIHS packet to which the CRC MIHS packet's **CRC16Value** or **CRC32Value** applies.

CRC MIHS packets may be beneficial when a MIHS stream is conveyed over an error prone channel.

# GlobalCRC16 & GlobalCRC32

The MIHSPacketType PACTYPE\_GlobalCRC16 or PACTYPE\_GlobalCRC32 identifies a global CRC MIHS packet which may be used to detect errors in the subsequent **protectedPacketsCount** MIHS packets including both the MIHSPacketHeader and MIHSPacketPayload portions of the packets. Global CRC MIHS packets contain a readCRC() structure defined in subclause 9.2.22 in the MIHSPacketPayload() defined in subclause 9.2.4.

A global CRC MIHS packet shall be directly followed by the **protectedPacketsCount** MIHS packets to which the global CRC MIHS packet's **CRC16Value** or **CRC32Value** applies.

Global CRC MIHS packets may be beneficial when a MIHS stream is conveyed over an error prone channel.

# Application examples

The figures below show examples of the order in which MIHS units and packets may be sent to convey the haptic experience.

# Initialization units

Initialization MIHS units, introduced in subclause 9.1.1, contain information essential to render the haptic signal and should be sent regularly to allow random access. The typical MIHS packets that may be present in an initialization unit are illustrated in Figure 25 and may be sent in any order.



Figure 25 — Initialization unit

# Temporal and spatial units

Temporal and Spatial MIHS units, introduced in subclause 9.1.2, contain haptic effect data defining the haptic signal. Temporal units contain data defining time-dependent effects. Spatial units contain data defining effects controlled by a spatial position. Both temporal and spatial units contain data MIHS packets as illustrated in Figure 26.



Figure 26 — Temporal and spatial units

Figure 27 shows a basic example on how MIHS units could be conveyed to stream a haptic experience.



Figure 27 — Basic example of MIHS unit sequencing

Temporal MIHS units contain a non-zero duration. To improve synchronization in the stream, initialization MIHS units, which contain a timing MIHS packet with a timestamp and timescale, may be sent periodically.

# Silent units

During intervals when there is no haptic data, silent MIHS units, which indicate a duration and contain no MIHS packets, may be added to the stream as illustrated in Figure 28



Figure 28 — Example with silent units

# CRC packets

MIHS packets of type PACTYPE\_CRC16, PACTYPE\_CRC32, PACTYPE\_GlobalCRC16, or PACTYPE\_GlobalCRC32 may be used for error detection within initialization, temporal, or spatial MIHS units as shown Figure 29 and Figure 30.



Figure 29 — GlobalCRC16 or GlobalCRC32 packet in an initialization unit



Figure 30 — CRC packets in temporal or spatial units

# Random access support with MIHS (informative)

The current streaming format supports random-access for streaming applications. This clause explains how applications should use random access.

Random access means that playback of a haptic experience can start at any time after a pre-defined maximum delay (the random-access period).

There are two ways to create a streamable haptic experience:

1. The content creation ensures that applications can decode haptic streams every random-access period (say *n* ms). The content editor must add keyframes every *n* ms to ensure independently decodable MIHS packets. The **packetDependency** flag in the data packets must indicate that the packets are independently decodable (see subclause 9.2.14). The independently decodable data packets must comprise a temporal MIHS unit (**MIHSUnitType** == UNITTYPE\_TEMPORAL) with the **MIHSUnitSync** flag indicating an independently decodable unit (see subclause 9.2.2). This is the most natural way to manage haptic streaming, as done with other media types.
2. The content has been created and stored in a different format, and no regular or specific random access has been planned. It might correspond to an experience created for file-based systems (where content is played back from the start of the stream). In that case some transcoding of the original content must be done (usually during the content preparation process of traditional distribution pipelines and before encoding). This transcoding aims at creating regular keyframes every random-access period, even when no keyframe exists. Similarly, these keyframes packets must be signaled as being independently decodable and grouped into MIHS units that are also signaled as independently decodable.

The streaming server encapsulates the independently decodable MIHS units (and MIHS packets) in relevant network signaling to ensure appropriate signaling for the decoder. Similarly, the streaming server must provide all the necessary metadata regularly by relevant means to ensure that the decoder can decode incoming packets.

The decoder, starting at any time, reads incoming MIHS units and waits for the next independently decodable unit. If missing metadata, the decoder must wait for the next initialization unit to get the missing metadata.

# Processing Model

This clause details the processing model of the codec. It describes every component of the encoder and decoder architecture presented in Figure 1.

The clause details the processing model for both the PCM and descriptive input formats.

# Encoder (informative)

# Encoder Architecture

Figure 31 depicts the encoder architecture in detail. The encoder is able to process three types of input files: OHM metadata files, descriptive haptics files (*.ivs, .ahap,* and *.hjif*) or waveform PCM files (*.wav*). The behavior of the encoder for each of these inputs is detailed respectively in clauses 10.1.2, 10.1.3 and 10.1.4. The encoder supports three types of output coded representation formats: the interchange format detailed in clause 7, a binary compressed format detailed in clause 8 and a streaming format detailed in clause 9.

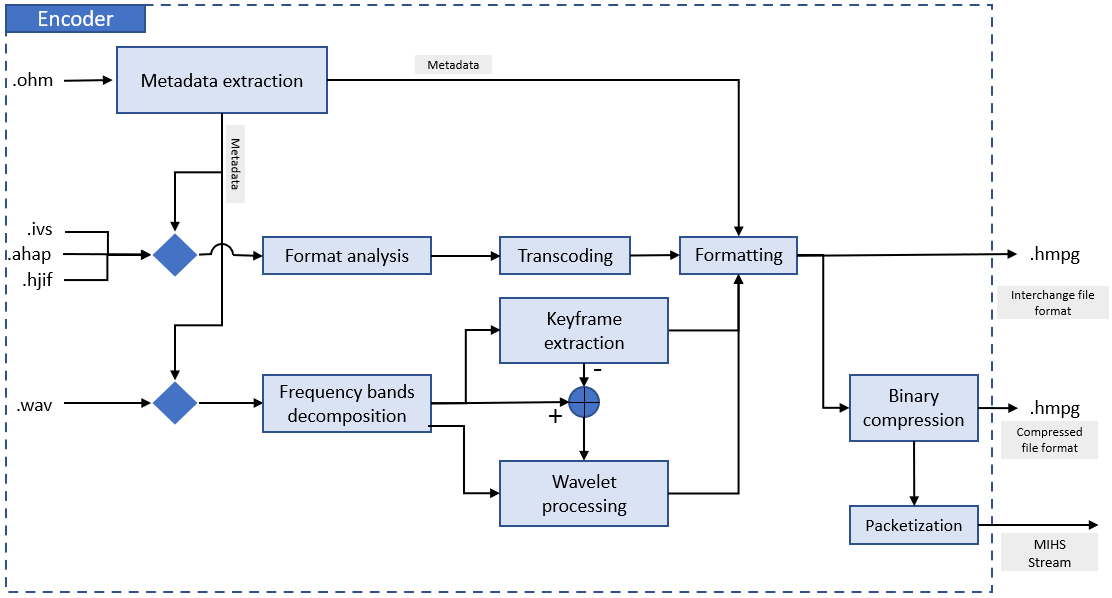


Figure 31: Encoder architecture.

# OHM metadata input file

This metadata file contains a description of the haptic system and setup. In particular, it provides the name of each associated haptic file (either descriptive or PCM) along with a description of the signals. It also provides a mapping between each channel of the signals and the targeted body parts on the user's body. The body mapping used in the OHM file is identical to the one detailed in Table 7, clause 7.2.5. The syntax of the OHM metadata file is given in Annex B.

When an OHM file is given as input, the encoder performs two operations:

* For each haptic\_element in the OHM file, it retrieves the associated haptic file from the URI (given in the haptic\_element\_file\_name field) and encodes it based on its type (see clauses 10.1.3 and 10.1.4). An OHM file may reference multiple haptic files of different types.
* It extracts metadata information from the OHM file and maps it to metadata information of the data model presented in clause 6. The mapping of the metadata between the two formats is detailed in Table 93.

Table 93: Mapping of the fields from an OHM file to the proposed data model

|  |  |  |
| --- | --- | --- |
| **OHM field** | **Data model component** | **property** |
| description\_string | Experience (clause 6.2) | **description** |
| element\_description\_string | Perception (clause 6.4) | **description** |
| channel\_description\_string | Channel (clause 6.6) | **description** |
| channel\_gain | Channel (clause 6.6) | **gain** |
| body\_part\_mask | Channel (clause 6.6) | **body part mask** |

# Descriptive input files

Descriptive input files are encoded through a simple process. The encoder first identifies specifically the input format. If the input format is a HJIF file, then no transcoding is necessary, the file can be further edited, compressed into the binary format and eventually packetized into an MIHS stream. If AHAP or IVS input files are used, a transcoding is necessary. The encoder first analyses semantically the input file information and transcodes it to be formatted into the data model presented in clause 6. The transcoding process for AHAP and IVS files is detailed in clause 10.1.5. After transcoding, the data can be exported as the *.hjif* file, a *.hmpg* binary file or an MIHS stream.

# PCM input file

Wav files cannot be transcoded directly into the output format. A signal analysis is performed to interpret the signal structure and convert it into the proposed encoded representation. To overcome the limitations of traditional frequency-based analysis processes on low frequencies, the encoding was split into two sub-processes. After performing a frequency band decomposition (detailed in clause 10.1.6) to separate the low frequencies from the high frequencies of the signal, the first process extracts the local extrema from the low frequencies and stores them as keyframes in a curve band. This process is detailed in clause 10.1.7. This first frequency band is then reconstructed using the appropriate interpolation method (as done at the synthesizer), and the residual error with the original signal is computed. This residual error is then added to the original high frequency bands. Finally, the second process applies a wavelet transformation on the high frequency spectrum (completed with the residual error). The wavelet processing is detailed in greater length in clause 10.1.8.

In the case that several low frequency bands are used, the residual errors from all the low frequency bands are added to the high frequency band before encoding. In the case that several high frequency bands are used, the residual errors from the low frequency band(s) are added to the first high frequency band before encoding. The user may choose to encode the input PCM signal using only one frequency band coding structure, and a specific band type.

If a specific target bitrate has to be met, encoder parameters have to be adjusted. In general, it is possible to design a custom encoder or at least a function that determines the encoder parameters based on the specified target bitrate. This can be done without having to change normative parts of the codec. For PCM files, a parametrization function has been developed that can be used if the signal is encoded only with the wavelet coding module. It determines the bit budget for the wavelet quantization, while all other parameters are kept constant, and takes the target bitrate as input. The basis to develop this function was a range of training files. They were encoded using bit budgets ranging from 1 to 135 to obtain the resulting bitrates (both in kbits/s). The maximum bitrate over all files for each bit budget was then calculated. These maximum values were used to fit a cubic function as approximation, such that any desired target bitrate can be taken to calculate the bit budget.

The determined function is

x denotes the target bitrate in kbits/s. The output of this function has to be rounded to the next lower integer and limited to the range [1,(log2(blockLength)-1)\*15] such that a valid bit budget is set.

If desired, this concept can be extended to an optimization algorithm, that adapts parameters to the input signal, for example by iteratively encoding the signal and changing parameters after each iteration.

# Transcoding descriptive content

Descriptive haptic codecs store parametrized commands to allow a synthesizer or a rendering software to reproduce the expected output. The current encoder ingests these commands from the input format to reproduce the same haptic behavior inside an effect of this proposed haptic codec. In the current specification, three types of descriptive files can be ingested and transcoded: AHAP, IVS and MPEG proposed HJIF files.

This descriptive format mainly uses Vectorial and Transient effects (or combination of) to describe haptic effects.

AHAP and IVS are existing proprietary formats to describe haptic effects. The reference software encoder can ingest those descriptions and translate it to the MPEG descriptive format. But it is not mandatory to describe an effect using those formats. The MPEG descriptive format proposed here can be used directly.

# IVS

IVS codec represents the expected haptic output by a set of basis effects parametrized by a set of parameters:

* An attack: duration and level
* A fade: duration and level
* An effect duration
* A period duration
* A wave type

Chart

Description automatically generated

Figure 32: Effect extracted from an IVS file and represented inside Immersion Haptic Studio.

Then a timeline is defined in which 2 types of commands can be raised:

* A launch event which will play a referenced basis effect at a specific time and can override its properties
* A repeat event which will create a loop inside the timeline to play multiple times the basis effects raised inside the repeated interval

The MPEG haptic encoder proposed will ingest the timeline of the IVS file to transcode it following these simple rules:

For each launch event:

* Find the corresponding basis effect
* Create an effect with matching waveform, magnitude and frequency. The frequency is computed by with the period length in milliseconds.
* Modulate the effect amplitude to match the configured attack or fade if exists. A linear interpolation is needed here. At least 2 keyframes will be stored in the effect, if an attack or a fade is configured, one and/or two keyframes will be inserted inside the effect to recreate a linearly interpolated amplitude at the beginning and/or ending of the effect.
* Find a vectorial wave band in which no effect is overlapping the created one. If none exists, create a new vectorial wave band with [0 Hz; 1000 Hz] as frequency range
* Insert the new effect inside the band

For each repeat effect:

* Find every effect after the repeat range and delay it
* Find every effect in the repeat range and copy it

# AHAP

AHAP codec represents the expected haptic output by a set of modulated continuous signals and a set of modulated transients parametrized.

Continuous: parametrized sine function configured by:

* A time: the timestamp at which the signal will be rendered
* A duration: the duration of the signal
* An intensity: the amplitude of the signal
* A sharpness: a value between 0 and 1 where 0 is the minimum frequency of the device and 1, its maximum frequency

Transients: parameterized API call configured by:

* A time: the timestamp at which the transient will be raised
* An intensity: the amplitude of the transient
* A sharpness: a value between 0 and 1 defined by 0 is the minimum frequency of the device and 1 its maximum frequency

After these primitive haptic effects are defined, the designer can modulate the amplitude and frequency of the whole experience by modulation functions composed of a set of keyframes.

The haptic encoder proposed will ingest the .ahap file to transcode it following these simple rules:

For each transient:

* Create a Transienteffect composed of one keyframe
* Fill the keyframe amplitude with the transient intensity
* Fill the keyframe frequency with the transient sharpness. The sharpness will be transformed and remapped from the original range [0; 1] into the frequency band range which will be configured as [65 Hz; 300 Hz]

For each continuous:

* Create a vectorial effect with a sine waveform
* The modulated parameters of the continuous will be stored in keyframes so 2 keyframes will be generated by default corresponding to the beginning and end of the effect. Then the keyframes can be modified and/or others can be added in this range to recreate the design intention drawn by the different modulation functions of the original AHAP file.
* Find a vectorial wave band in which no effect is overlapping the created one. If none exists, create a new vectorial wave band with [65 Hz; 300Hz] as frequency range
* Store the effect in the previously found band

# Frequency Band decomposition

The frequency band decomposition splits the signal into a low frequency and a high frequency band. In the reference implementation, this is done using Butterworth highpass and lowpass filters of order 8 in a forward and a consecutive backward pass for zero phase filtering. It follows the scipy.signal.filtfilt method from the scipy Python library[3], for filter coefficients, the function scipy.signal.butter[3] can be used as reference, since the coefficients will change based on the chosen sampling frequency and cutoff frequency. This filter is not normative, so different filters can be utilized.

It is recommended to set the cut-off frequency in a range of 20 to 72.5 Hz or to 0 Hz, which is equivalent to using only the wavelet band for the complete signal.

The number of low and high frequency bands is not fixed, however a common practice is to use two as described here. The user can select other configurations.

# Keyframe extraction for low frequencies processing

The Curve band processing takes the lower frequency band from the frequency band decomposition and analyses its content in the time domain. An algorithm of local extremum extraction is applied to the signal to extract points defined by their timestamps and amplitude. Each one of these points is then encoded into keyframes composing a unique effect on a curve band (see Figure 10).

Diagram, line chart

Description automatically generated

Figure 33: Curve bands processing.

The algorithm used for this keyframe extraction is described in Table 94 where the input *“filteredSignal”* is a list of samples extracted and filtered by the method described in clause 10.1.6 and *“sampleRate”* is the sample rate of the signal previously mentioned.

Table 94 Pseudocode for the Curve band encoding

|  |
| --- |
| Syntax |
| EncodeCurveBand(filteredSignal, sampleRate) |
| {  encodedEffect = new Effect()  encodedEffect.insertKeyframeAt(0, 0)  for(i = 1; i < filteredSignal.size() – 1; i++) {  lastSample = filteredSignal[i - 1]  currentSample = filteredSignal[i]  nextSample = filteredSignal[i + 1]  isFlat = lastSample == currentSample and currentSample == nextSample  isMinima = lastSample >= currentSample and currentSample <= nextSample  isMaxima = lastSample <= currentSample and currentSample >= nextSample  if(not isFlat and (isMinima or isMaxima)) {  timestamp = 1000 \* i / samplerate  amplitude = currentSample  encodedEffect.insertKeyframeAt(timestamp, amplitude)  }  }  encodedEffect.insertKeyframeAt(1000 \* filteredSignal.size() / sampleRate, 0)  encodedBand = new Band()  encodedBand.insert(encodedEffect)  return encodedBand  } |

# Wavelet encoding

The wavelet band (WaveletWave type) processing of the encoder takes the high frequency band from the frequency band decomposition and the low frequency residual, and splits it into blocks of equal size, which is called blockLength. The blockLength has to be a power of 2 and at least 16 samples. It is typically set to 1024.

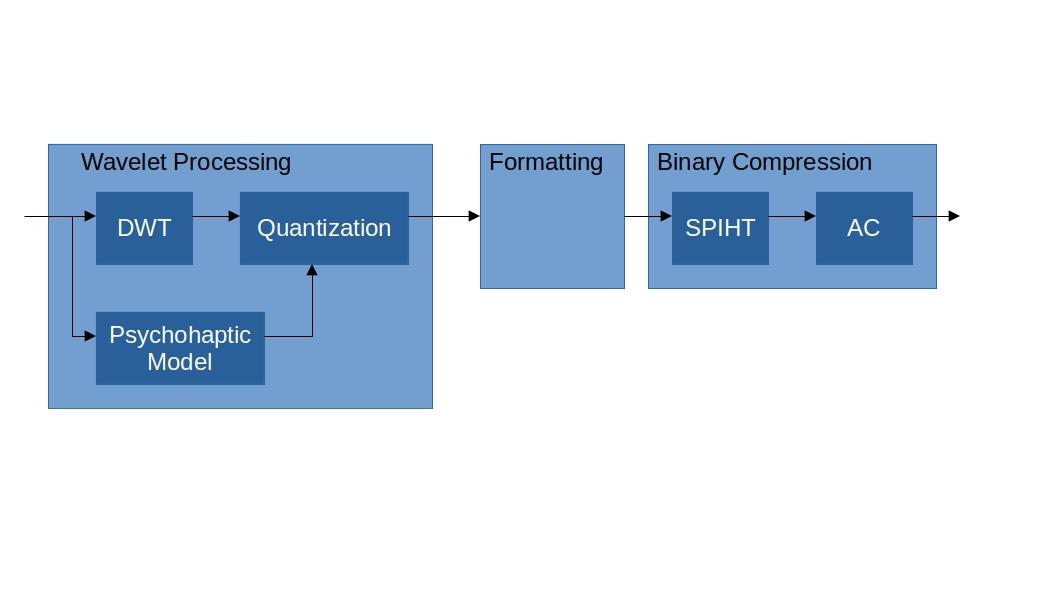


Figure 34: Wavelet bands encoder signal flow.

The signal block is then analysed in the psychohaptic model. The lossy compression is applied by wavelet transforming the block and quantizing it, aided by the psychohaptic model. In the end, each block is then saved into a separate effect in a single band, which is done in the formatting. The binary compression applies lossless compression using the SPIHT algorithm and Arithmetic Coding (AC). This process is illustrated in Figure 34.

The bitrate scaling is achieved by adjusting the bit budget parameter, which will be explained later on. It is currently set to 3 for 2 kbits/s, 16 for 16 kbits/s, and 66 for 64 kbits/s.

**Wavelet transformation**

The signal block is wavelet transformed using CDF9/7 filters. Their coefficients are shown in table 13.

table 13: coefficients of the wavelet filters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | LP | HP | LP\_reconstruction | HP\_reconstruction |
| -4 | 0.037828455506995 |  |  | -0.037828455506995 |
| -3 | -0.023849465019380 | -0.064538882628938 | -0.064538882628938 | -0.023849465019380 |
| -2 | -0.110624404418423 | 0.040689417609559 | -0.040689417609559 | 0.110624404418423 |
| -1 | 0.377402855612654 | 0.418092273222212 | 0.418092273222212 | 0.377402855612654 |
| 0 | 0.852698679009404 | -0.788485616405665 | 0.788485616405665 | -0.852698679009404 |
| 1 | 0.377402855612654 | 0.418092273222212 | 0.418092273222212 | 0.377402855612654 |
| 2 | -0.110624404418423 | 0.040689417609559 | -0.040689417609559 | 0.110624404418423 |
| 3 | -0.023849465019380 | -0.064538882628938 | -0.064538882628938 | -0.023849465019380 |
| 4 | 0.037828455506995 |  |  | -0.037828455506995 |

The transformation separates the block into a higher and a lower frequency band and applies down-sampling with factor 2 on both bands. This is iteratively repeated for the lowest band. The number of iterations, or dwtlevels, is calculated by

where *blockLength* is the length of the signal block to process. The iterative filtering leads to increasing band size from low to high frequencies, with each band two times as large as the next lower one. The figure below illustrates this splitting of coefficients into sub-bands for 32 samples total size of the block.

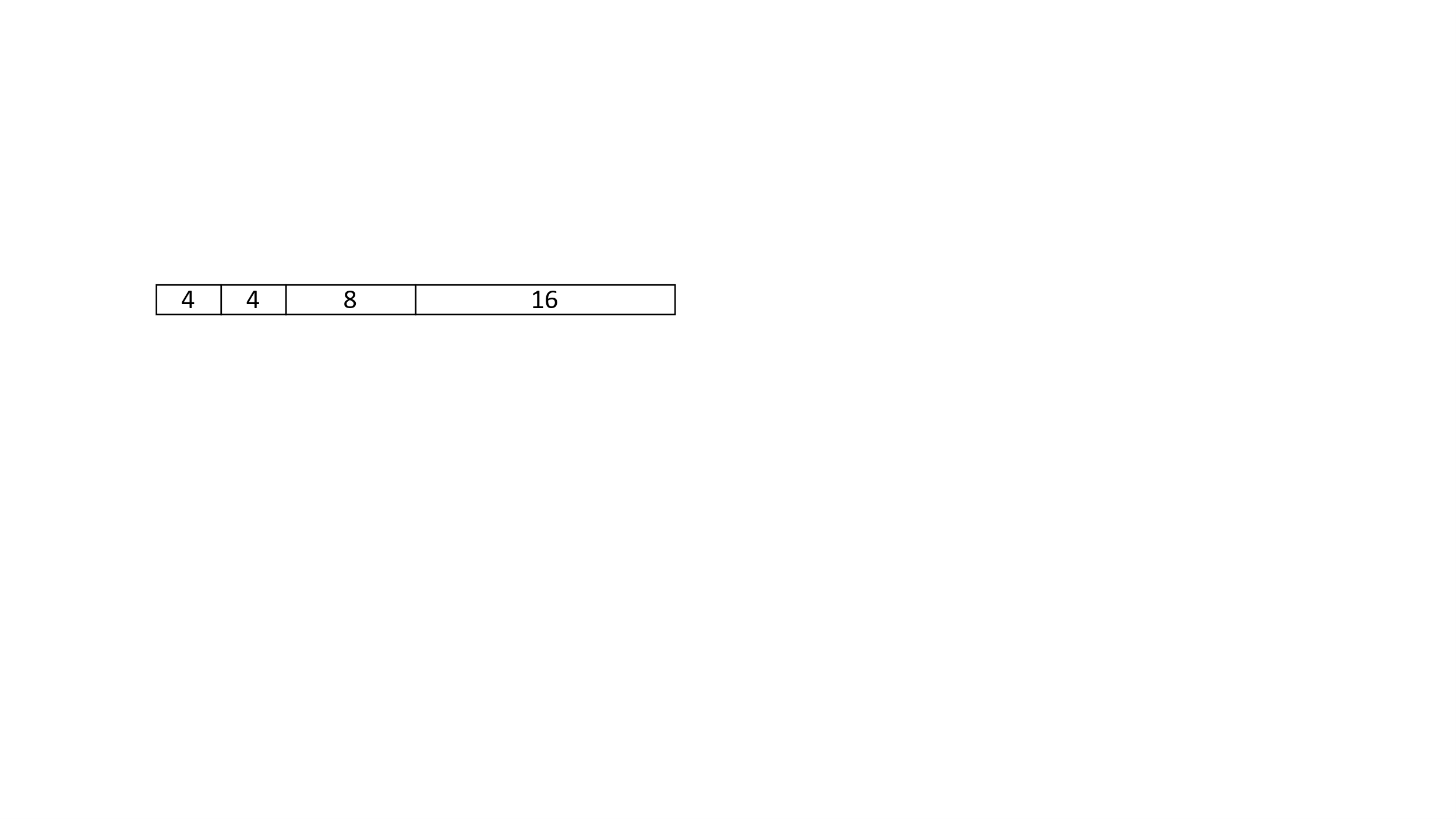


Figure 35: Wavelet block coefficients in sub-bands.

**Psychohaptic model**

The psychohaptic model is used to evaluate the perceptual importance of each wavelet band. For this, masking in the frequency domain and the frequency dependence of the sensitivity of the human sense of touch are assessed. Masking in the frequency domain occurs, when content with higher amplitude overshadows other components of the signal with lower amplitude and similar frequency.

To model this phenomenon, first all local maxima, also called peaks, are detected in the FFT spectrum (logarithmic domain, root power). To get as many frequency bins as input samples, the signal block is zero padded to twice its size before the FFT. Then, some of the peaks are sorted out. Peaks with an amplitude 45 dB below the maximum amplitude in the spectrum are omitted as well as peaks with a prominence below 12 dB. The prominence is obtained by first finding the two surrounding local minima on the left and right side of the peak and then calculating the amplitude difference between the peak and the larger one of the minima. For each remaining peak with index *p* a masker function *mp(f)* is generated. It is a polynomial function centered at the frequency of the local peak and calculated using the formula:

,

where *fS* is the sampling frequency and *fp* the center frequency of the peak. *ap* is the amplitude of the spectrum at *fp*.The masking threshold is then calculated by taking the maximum at each frequency as:

.

*P* denotes the quantity of peaks.

The frequency dependence of the human perception is modelled by the absolute threshold of perception *t(f)*, calculated by:

Additionally, this function is limited to 0 dB. This way, high frequency content will not be ignored in the evaluation and is a measure similar to the threshold of pain from the audio domain. There it is assumed that when a high frequency signal reaches a certain amplitude, it will be painful and therefore perceivable.

The masking threshold and absolute threshold of perception are then combined to the global masking threshold. This is done by addition in the linear domain:

As the last step, a Signal-to-Mask-Ratio (SMR) is calculated. For this, the signal energy *Esig,b* for each wavelet band *b* and the mask energy *Emask,b* are calculated. This is done by summing up all coefficients of the input signal block and the mask corresponding to the respective band (linear domain). The SMR then calculates as

**Quantization**

The quantization is performed using an embedded quantizer with individual bit depths for each wavelet band. The embedded property guarantees that the quantization steps for low bit depths are also present when using a higher bit depth.

The quantizer model used for the wavelet quantization is a uniform quantizer. The quantization step size Δ is:

,

where is the quantized maximum wavelet coefficient of the signal block and *b* is the bit depth in the band. The quantization is performed using:

Here, is the quantized version of wavelet coefficient .

To determine the bit depth for each wavelet band, a bitBudget is iteratively distributed on the bands. So, this bit budget represents the sum of bit depths over the wavelet bands. The bit budget is used to control the quality of the encoded signal and also scales the resulting length of the bitstream and therefore the compression ratio. In each iteration, the band with the lowest Mask-to-Noise-Ratio (MNR) is found and a bit is allocated to it. The MNR is calculated from the SMR and the Signal-to-Noise-Ratio (SNR) by

The bit allocation is only informative, since other methods to determine the bit depths can be found. The reference software follows this pseudocode:

|  |
| --- |
| Syntax |
| bitAllocation(w, bitBudget, w\_max) |
| {  bitalloc\_sum = 0;  w\_quant = zeros\_like(w);  while(bitalloc\_sum < bitBudget) {  index = argmin(MNR);  bitalloc[index] += 1;  bitalloc\_sum += 1;  w\_quant = quantization(w, w\_quant, index, bitalloc[index]);  SNR = updateSNR(w, w\_quant, SNR, index);  MNR[index] = SNR[index] – SMR[index];  if(bitalloc[index] >= 15) {  MNR[index] = INFINITY;  }  }  } |

w refers to the input signal block in the wavelet domain, bitBudget to the maximum sum of allocated bits. w\_max is the absolute maximum wavelet coefficient in the specific block and w\_quant is the output block, which is the quantized version of the input block.

zeros\_like(w) refers to a function that generates a vector containing zeros with the length of vector w. quantization(w, w\_quant, index, bitalloc[index]) quantizes the input w in band index using bitalloc[index] bits and updates this specific band in w\_quant. updateSNR(w, w\_quant, SNR, index) updates the SNR of w\_quant relative to w in band index.

As the last step of the quantization, the signal is scaled to values in the range [-1,1] by:

and then the data is saved in a new effect, including the quantized wavelet coefficients scaled to the range [-1,1], the quantized original maximum wavelet coefficient and the maximum allocated bits over all bands. The relation between w\_quant and wavelet\_coefficients is that w\_quant is scaled to the original amplitude, while wavelet\_coefficients is the version scaled to [-1,1]. So, the new effect contains the lossy compression that is applied on the signal and can directly be used for the lossless binary compression.

**Wavelet band binary compression**

The binary compression for wavelet bands processes each quantized effect in the wavelet band and transforms it into a bitstream in a lossless fashion. The goal is to get a stream that has compact structure and can be transmitted. It consists of an 1D-SPIHT coder and an Arithmetic Coder (AC). The signal has to be scaled first from the floating point values in the range [-1,1] to integer values before the compression can be applied. This is necessary to have a quantization step of one. This is done by multiplying it with , where is the maximum bit depth used in quantization.

**SPIHT**

Set Partitioning In Hierarchical Trees (SPIHT) is an algorithm based on Embedded Zerotree Wavelet (EZW) coding and was introduced in [4]. It exploits the self-similarity of the wavelet coefficients across the bands and bitplane and operates iteratively on each bitplane of the signal, starting with the most significant bit. Its output is a binary stream.

The nth bitplane consists of all nth bits of the samples in a signal, with indexing starting at n=0 for the least significant bit. The signs of the coefficients are treated separately. The algorithm consists of a significance pass that finds significant coefficients in the block and a refinement pass that only codes the next lower bitplane of the significant samples in each iteration.

Significance is a notion that separates larger and more important coefficients from the others, which can be omitted. The exact formulation for it will be presented later in this clause. In both the significance pass and the refinement pass, bits are written to a bitstream, which enable the decoder to reconstruct the data.

In 1D SPIHT, coefficients are arranged in a tree structure, where each coefficient is represented by a node and the two coefficients with the same spatial orientation in the next lower band are set to be the child, or offspring, of that node. The resulting structure is called spatial orientation tree. In general, for natural signals, it is likely that coefficients with the same spatial orientation are similar across sub-bands. Also, the signal energy tends to decrease to higher frequencies. So, using the spatial orientation tree results in good magnitude ordering of the coefficients. This tree is further explained in [4], extended to the 2D case, and illustrated in Fig. 13.

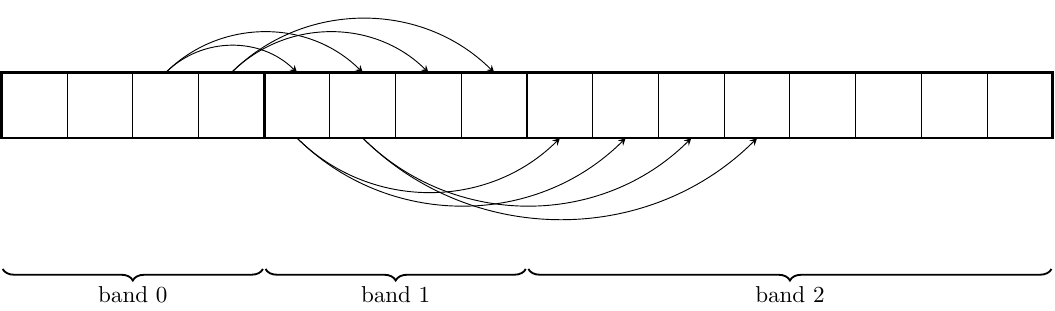


Figure 36: SPIHT Spatial Orientation Tree.

The algorithm, reduced to the 1D case, uses the following sets to implement the spatial orientation tree:

* O(i) set of indices of all offspring of node i;
* D(i) set of indices of all descendants of node i;
* H set of indices of all spatial orientation tree roots;
* L(i) = D(i) - O(i).

O(i) is obtained using

The partitions for the algorithm are initialized with the sets {(i)} and D(i) for all . If D(i) is found to be significant during a sorting pass, it is partitioned into L(i) and two single-element sets with . If L(i) is significant, it is divided into two sets D(k) with .

To determine the significance in bitplane n of a set of samples , the expression

is evaluated. is 1 if the expression is valid, otherwise 0. Effectively, this is carried out as a check for ones in the respective bitplane n in the concrete algorithm. If is 0, is called insignificant. Sets that are insignificant over all bitplanes are not coded in the refinement pass, so coding bits are saved for those sets. Due to the spatial orientation tree, it is likely that the significance is similar for whole sets and therefore good compression performance can be reached.

During the algorithm, three ordered lists are used: the list of insignificant sets (LIS), list of insignificant pixels (LIP), and list of significant pixels (LSP). The term pixels originates from image processing and correlates to coefficients in the case of wavelet transformed vibrotactile signals. In the LIP and LSP, single coefficients are saved, and in the LIS, sets of coefficients from set L(i) or D(i) are put. Coefficients in the set D(i) are called type A and type B if they are from L(i).

In each iteration of the algorithm, first the significance is checked in the sorting pass. Pixels that were insignificant so far are in the LIP, therefore the LIP is checked in each iteration. If they are found to be significant, they are put in the LSP and their sign is transmitted. Sets are also checked and partitioned if they are found to be significant. If the newly partitioned sets are single coefficients, they are put in the LIP instead.

In the refinement pass, the current bitplane is transmitted for the pixels in the LSP, excluding those added in the latest sorting pass, because they are already known to have a 1 in that plane from the significance check.

The bits in the stream serve different purposes as explained above, so they have different so-called contexts. These contexts are written to a separate output stream and sent to the AC alongside the bitstream. The bits can either be significance bits, sign bits, refinement bits or side information bits. Since the significance is checked for different cases, it can be further subdivided. This results in the following list of possible contexts:

* significance: LIP
* significance: LIS, type A
* significance: descendants of type A in LIS
* significance: LIS, type B
* sign bits
* refinement bits
* side information bits

The following pseudocode describes the functionality of the SPIHT encoder. It generates two temporary streams: **SPIHT\_stream** and **context**. The variable *wavelet\_coefficients* refers to the quantized wavelet coefficients arranged in an array. They are originally in the range [-1,1], but scaled to integer values in the beginning of the function. *dwtlevel*is the number of wavelet transformation iterations and *maxallocbits* is the maximum bit depth of the wavelet coefficients.

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| SPIHT\_encode(wavelet\_coefficients, dwtlevel, bitwavmax, maxallocbits, SPIHT\_stream, context) {  length = wavelet\_coefficients.size();  for (i=0; I < length; i++) {  wavelet\_coefficients[i] = wavelet\_coefficients[i] \* (2^n-1);  }  **SPIHT\_stream.push\_back(maxallocbits);**  **SPIHT\_stream.push\_back(bitwavmax);**  **context.append\_zeros(12);** // 4 for maxallocbits, 8 for bitwavmax  int bandsize = 2 << (log2(length) - dwtlevel);  for (i = 0; i < bandsize; i++) {  LIP.push\_back(i);  }  for (int i = (bandsize / 2); i < bandsize; i++) {  LIS1.push\_back(i);  LIS2.push\_back(0);  }  initMaxDescendants(wavelet\_coefficients);  n = maxallocbits;  while (0 <= n) {  compare = 1 << n;  LSP\_index = LSP.size();  for (it = LIP.begin(); it != LIP.end();) {  if (abs(in[LIP[it]]) >= compare) {  addToOutput(1, 2, **SPIHT\_stream**, **context**);  addToOutput(in[LIP[it]] >= 0, 1, **SPIHT\_stream**, **context**);  LSP.push\_back(it);  it = LIP.erase(it);  } else {  addToOutput(0, 2, **SPIHT\_stream**, **context**);  it++;  }  }    it1 = LIS1.begin();  it2 = LIS2.begin();  LISsize = LIS1.size();  for (i = 0; i < LISsize; i++) {  if (in[LIS2[it2]] == 0) {  int max\_d = maxDescendant(LIS1[it1], LIS2[it2]);  if (max\_d >= compare) {  addToOutput(1, 3**, SPIHT\_stream**, **context**);  y = LIS1[it1];  index = 2 \* y;  if (abs(in[index]) >= compare) {  LSP.push\_back(index);  addToOutput(1, 4, **SPIHT\_stream**, **context**);  addToOutput(in[index] >= 0), 1, **SPIHT\_stream**, **context**);  } else {  addToOutput(0, 4, **SPIHT\_stream**, **context**);  LIP.push\_back(index);  }  index = 2 \* y + 1;  if (abs(in[index]) >= compare) {  LSP.push\_back(index);  addToOutput(1, 4, **SPIHT\_stream**, **context**);  addToOutput(in[index] >= 0), 1, **SPIHT\_stream**, **context**);  } else {  addToOutput(0, 4, **SPIHT\_stream**, **context**);  LIP.push\_back(index);  }  if ((4 \* y + 3) < length) {  LIS1.push\_back(LIS1[it1]);  LIS2.push\_back(1);  LISsize++;  }  it1 = LIS1.erase(it1);  it2 = LIS2.erase(it2);  } else {  addToOutput(0, 3, **SPIHT\_stream**, **context**);  it1++;  it2++;  }  } else {  max\_d = maxDescendant(LIS1[it1], LIS2[it2]);  if (max\_d >= compare) {  addToOutput(1, 5, **SPIHT\_stream**, **context**);  int y = LIS1[it1];  LIS1.push\_back(2 \* y);  LIS1.push\_back(2 \* y + 1);  LIS2.push\_back(0);  LIS2.push\_back(0);  LISsize += 2;  it1 = LIS1.erase(it1);  it2 = LIS2.erase(it2);  } else {  addToOutput(0, 5, **SPIHT\_stream**, **context**);  it1++;  it2++;  }  }  }    refinementPass(wavelet\_coefficients, LSP, LSP\_index, n, **SPIHT\_stream**, **context**);  n--;  }  } | **4**  **8**  **12** | **uilsbf**  **blsbf**  **islif** |

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| refinementPass(data, LSP, LSP\_index, n, outstream, context) {  it = LSP.begin();  temp = 0;  while (temp < LSP\_index) {  s = bitget(floor(abs(data[LSP[it]])), n + 1);  **outstream.push\_back(s);**  **context.push\_back(6);**  temp++;  it++;  }  } | **1**  **1** | **vlclbf**  **vlislif** |

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| addToOutput(bit, c, outstream, context) {  **outstream.push\_back(bit);**  **context.push\_back(c);**  } | **1**  **1** | **vlclbf**  **vlislif** |

|  |
| --- |
| maxDescendant(j, type) {  if (type == 1) {  if (j >= maxDescendants1.size()) {  return 0;  }  return maxDescendants1[j];  }  if (j >= maxDescendants.size()) {  return 0;  }  return maxDescendants[j];  } |

|  |
| --- |
| initMaxDescendants(signal) {  length = signal.size();  start = length >> 1;  p1 = start;  p2 = p1 + 1;  target = start >> 1;  for (i = 0; i < (start >> 1); i++) {  v1 = abs(signal[p1]);  v2 = abs(signal[p2]);  if (v1 > v2) {  maxDescendants[target] = v1;  } else {  maxDescendants[target] = v2;  }  p1 += 2;  p2 += 2;  target++;  }  width = start >> 1;  p1 = width;  p2 = p1 + 1;  target = width >> 1;  while (target > 1) {  for (i = 0; i < (width >> 1); i++) {  v1 = maxDescendants[p1];  v2 = maxDescendants[p2];  if (v1 > v2) {  maxDescendants1[target] = v1;  } else {  maxDescendants1[target] = v2;  }  v1 = abs(signal[p1]);  if (v1 > maxDescendants1[target]) {  maxDescendants[target] = v1;  } else {  maxDescendants[target] = maxDescendants1[target];  }  v2 = abs(signal[p2]);  if (v2 > maxDescendants[target]) {  maxDescendants[target] = v2;  }  p1 += 2;  p2 += 2;  target++;  }  width = width >> 1;  p1 = width;  p2 = p1 + 1;  target = width >> 1;  }  } |

The following function describes how the quantized maximum wavelet coefficient qwavmax is encoded, which is then carried over to the encode function of SPIHT as bitwavmax:

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| maximumWaveletCoefficient(qwavmax, bitwavmax) {  i\_part = 0;  mode = 0;  if (qwavmax < 1) {  i\_bits = 0;  f\_bits = 7;  } else {  i\_part = 1;  i\_bits = 4;  f\_bits = 3;  mode = 1;  }  **bitwavmax.push\_back(mode);**  **bitwavmax.push\_back((qwavmax - i\_part) \* pow(2, f\_bits));**  } | **1**  **i\_bits + f\_bits** | **bslbf**  **bslbf** |

|  |
| --- |
| bitget(in, bit) {  mask = 1 << (bit - 1);  if ((in & mask) > 0) {  return 1;  }  return 0;  } |

**Arithmetic Coder**

The AC takes the bitstream generated by SPIHT and compresses it further, utilizing the context information from SPIHT.

Arithmetic coding is a type of entropy coding. It works by iteratively subdividing an interval based on the input symbol sequence and the probability of the symbols. Each possible input sequence results in a different unique subinterval, so the sequence can be recovered from the interval unambiguously. [0,1024) is chosen as initial interval, in integer values. Since the calculation precision is finite, the interval has to be upscaled if it falls below a certain width. At each upscaling, bits are set to the output bitstream, representing the interval.

Since SPIHT puts out contexts for each bit of the stream, the AC can refine the probability estimation by calculating the probabilities for each context separately. This is done by counting occurrences of zeros and ones for the contexts. The probability p(0,n) of an occurring zero in the nth context is calculated using

with c0(n) being the counter for zeros in the nth context and ctot(n) being the counter for all symbols in context n. c0(n) is initialized with 8 and ctot(n) with 16. After each block, the counters are reset to their initial state.

The pseudocode describing the arithmetic encoder is displayed below. The instream input corresponds to the SPIHT\_stream with its context .The output is written to the arithmetic\_stream introduced above.

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| AC\_encode(SPIHT\_stream, context, arithmetic\_stream) {  range\_lower = 0;  range\_upper = 1024;  bits\_to\_follow = 0;  range\_diff = 0;  new\_symbol = 0;  c = 0;  range\_add = 0;  for (i = 0; i < SPIHT\_stream.size(); i++) {  range\_diff = range\_upper - range\_lower;  new\_symbol = **SPIHT\_stream[i]**;  c = context[i];  p = round(counter[c] / counter\_total[c] \* 1024);  range\_add = (range\_diff \* p) / 1024;  if (range\_add == 0) {  range\_add = 1;  } else if (range\_add == range\_diff) {  range\_add = range\_diff - 1;  }  if (new\_symbol == 0) {  range\_upper = range\_lower + range\_add;  } else {  range\_lower = range\_lower + range\_add;  }  while (true) {  if (range\_upper <= 512) {  if (bits\_to\_follow > 0) {  outstream.push\_back(0);  for (j = 0; j < bits\_to\_follow; j++) {  **arithmetic\_stream.push\_back(1);**  }  bits\_to\_follow = 0;  } else {  **arithmetic\_stream.push\_back(0);**  }  } else if (range\_lower >= 512) {  if (bits\_to\_follow > 0) {  **arithmetic\_stream.push\_back(1);**  for (j = 0; j < bits\_to\_follow; j++) {  **arithmetic\_stream.push\_back(0);**  }  bits\_to\_follow = 0;  } else {  **arithmetic\_stream.push\_back(1);**  }  range\_lower -= 512;  range\_upper -= 512;  } else if (range\_lower >= 256 && range\_upper <= 768) {  bits\_to\_follow++;  range\_lower -= 256;  range\_upper -= 256;  } else {  break;  }  range\_lower = range\_lower << 1;  range\_upper = range\_upper << 1;  }  if (instream[i] == 0) {  counter[c]++;  }  counter\_total[c]++;  }  remainder(bits\_to\_follow, arithmetic\_ stream, range\_lower, range\_upper);  index\_end = arithmetic\_stream.size() - 1;  while (arithmetic\_stream[index\_end] == 0 && index\_end >= 0) {  index\_end--;  }  **arithmetic\_stream.resize(index\_end + 1);**  resetCounter();  } | **1**  **1**  **1**  **1**  **1** | **vlclbf**  **vlclbf**  **vlclbf**  **vlclbf**  **vlclbf** |

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| remainder(bits\_to\_follow, arithmetic\_stream, range\_lower, range\_upper) {  if (bits\_to\_follow > 0) {  **arithmetic\_stream.push\_back(1);**  } else {  int val = 512;  while (range\_lower > 0) {  if (val < range\_upper) {  **arithmetic\_stream.push\_back(1);**  range\_lower -= val;  range\_upper -= val;  } else {  **arithmetic\_stream.push\_back(0);**  }  val = val >> 1;  }  }  } | **1**  **1**  **1** | **vlclbf**  **vlclbf**  **vlclbf** |

|  |
| --- |
| resetCounter() {  for (size\_t i = 0; i < 7; i++) {  counter[i] = 16 / 2;  counter\_total[i] = 16;  }  } |

# Decoder

# Decoder Architecture

Figure 37 illustrates the architecture of the decoder. The decoder may take as input either a *.hmpg* compressed binary file format or an MIHS bitstream. It outputs a *.hjif* interchange format that can be used directly for rendering.

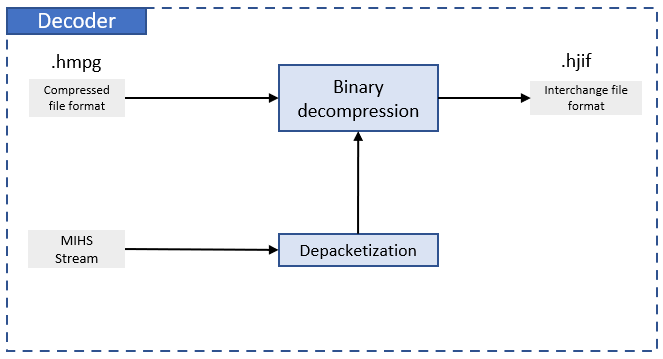


Figure 37: Decoder architecture

The two input formats shall go through a binary decompression step to extract both the metadata and the data itself from the file and map it to the data structure detailed in clause 6. After this step, the data can be exported in the *.hjif* format following the specifications of clause 7.

The decoding for both formats is based on the same processes. For metadata and parametric data (i.e. data in Transient bands, Curve bands or Vectorial bands), the decoding process is explained in clause 10.2.2. For Wavelet bands, typically encoded from PCM content the decoding process is detailed in clause 10.2.3.

# Metadata and parametric data decoding

For metadata and data contained in Transient bands, Curve bands or Vectorial bands, the binary data follows the data model structure detailed in clause 6 and can therefore be decoded directly.

For a *.hmpg* binary input file format, the binary decompression process is detailed in clause 8.

For the MIHS stream, the depacketization and binary decompression are detailed in clause 9.

# Wavelet Decoding

The data contained in Wavelet bands require specific processing. Clauses 8 and 9 describe how to extract the encoded wavelet data from respectively a compressed binary file or a MIHS stream. This clause describes the processing modules used to decode this data and output the wavelet coefficients required to render the haptic signal.

**Wavelet band binary decompression**

The decoder for the binary wavelet compression has to reverse the AC and SPIHT coding. Since the context of the SPIHT bits is only known for the very next bit, the AC has to work nested into the SPIHT coder. Each bit is first decoded with the AC and then with SPIHT, before the next bit can be processed. Therefore, the SPIHT decoder takes the arithmetic\_stream as input and there is no intermediate SPIHT\_stream or context stream. The AC obtains the symbol probability by keeping track of the symbol frequency and updating it after each symbol like the encoder does. SPIHT can recover the original signal by inferring the members of the sets defined in the encoder from the transmitted bitstream. In the end of the SPIHT decoder, the coefficients are divided by to invert the scaling done in the binary encoder. So, then the wavelet\_coefficients are in a range of [-1,1] to be stored in the keyframes of the effect structure of the descriptive format.

The following functions describe the algorithm of the SPIHT decoder. It takes the arithmetic\_stream as input, which is the binary compressed bitstream, as well as the block\_length defined in the band header, dwtlevel from the wavelet transform settings, wavmax for the original amplitude of the signal and bit\_depth, which is the maximum bit depth used for the quantization. The output is written to wavelet\_coefficients. The list LSP is split into LSP1 for the indices and LSP2 for the type of the coefficient. bi2de() refers to a function to convert binary streams into unsigned integers.

|  |
| --- |
| SPIHT\_decode(arithmetic\_stream, wavelet\_coefficients, block\_length, dwtlevel, wavmax, bit\_depth) {  AC\_initDecoding(arithmetic\_stream);  bit\_depth = getMaxAllocBits();  wavmax = getWavmax();  initLists(block\_length, level);  n = bit\_depth;  while (0 <=n) {  compare = 1 << n;  LSP\_index = LSP.size();  sortingPass(wavelet\_coefficients, block\_length, compare);  refinementPass(wavelet\_coefficients, LSP\_index, compare);  n--;  }  AC\_resetCounter();  for (i=0; I < wavelet\_coefficients.size(); i++) {  wavelet\_coefficients[i] = wavelet\_coefficients[i] / (2^n-1);  }  } |

|  |
| --- |
| initLists(block\_length, dwtlevel) {  bandsize = 2 << log2(block\_length)-dwtlevel;  for(i = 0; i < bandsize; i++) {  LIP.push\_back(i)  }  for(i = bandsize/2; i < bandsize; i++) {  LIS.push\_back(i);  LIS2.push\_back(0);  }  } |

|  |
| --- |
| getMaxAllocBits() {  maxallocBitsArray = getBits(4, 0);  return bi2de(maxallocBitsArray, 4);  } |

|  |
| --- |
| getWavMax() {  mode = getBit(0);  wavMaxArray = getBits(7, 0);  temp = bi2de(wavmaxArray,7, 0);  wavmax = 0;  if(mode == 0) {  wavmax = temp \* 2^(-7);  else {  wavmax = temp \* 2^(-3) + 1;  }  return wavmax;  } |

|  |
| --- |
| sortingPass(wavelet\_coefficients, block\_length, compare) {  for(it = LIP.begin(); it != LIP.end();) {  if(getBit(2) == 1) {  if(getBit(1) == 1) {  **wavelet\_coefficients[LIP[it]] = compare;**  } else {  **wavelet\_coefficients[LIP[it]] = -compare;**  }  LSP.push\_back(it);  it = LIP.erase(it)  } else {  it++;  }  }  it1 = LIS1.begin();  it2 = LIS2.begin();  LISsize = LIS1.size();  for(i = 0; i < LISsize; i++) {  if(LIS2[it2] == 0) {  if(getBit(3) == 1) {  y = LIS1[it1];  index = 2\*y;  if(getBit(4) == 1) {  LSP.push\_back(index);  if(getBit(1) == 1) {  **wavelet\_coefficients[index] = compare;**  } else {  **wavelet\_coefficients[index] = -compare;**  }  }else {  LIP.push\_back(index);  }  index = 2\*y+1;  if(getBit(4) == 1) {  LSP.push\_back(index);  if(getBit(1) == 1) {  **wavelet\_coefficients[index] = compare;**  } else {  **wavelet\_coefficients[index] = -compare;**  }  }else {  LIP.push\_back(index);  }  if((4\*y+3) < block\_length) {  LIS1.push\_back(LIS1[it])  LIS2.push\_back(1);  LISsize++;  }  it1 = LIS1.erase(it1);  it2 = LIS2.erase(it2);  } else {  it1++;  it2++;  }  } else {  if(getBit(5) == 1) {  y = LIS1[it1];  LIS1.push\_back(2\*y);  LIS1.push\_back(2\*y+1);  LIS2.push\_back(0);  LIS2.push\_back(0);  LISsize += 2;  it1 = LIS1.erase(it1);  it2 = LIS2.erase(it2);  } else {  it1++;  it2++;  }  }  }  } |

|  |
| --- |
| refinementPass(wavelet\_coefficients, LSP\_index, compare) {  it = LSP.begin();  temp = 0;  while(temp < LSP\_index) {  if(getBit(6) == 1) {  **wavelet\_coefficients[LSP[it]] +=** sign(**wavelet\_coefficients[LSP[it]]**) \* compare;  }  temp++;  it++;  }  } |

|  |
| --- |
| getBit(context) {  return AC\_decode(context);  } |

|  |
| --- |
| getBits(length, context) {  for(i = 0; i < length; i++) {  out[i] = AC\_decode(context);  }  return out;  } |

The following code blocks describe the functionality of the arithmetic decoder.

|  |
| --- |
| AC\_initDecoding(arithmetic\_stream) {  in\_index = 0;  max\_index = arithmetic\_stream.size();  in\_leading = 0;  shift = 9  for (i = 0; i < 10; i++) {  if (i < arithmetic\_stream.size()) {  in\_leading += **arithmetic\_stream[in\_index]** << shift;  in\_index++;  shift--;  } else {  break;  }  }  range\_diff = 1024;  range\_lower = 0;  range\_upper = 1024;  } |

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| AC\_decode(context) {  p = round(counter[context] / counter\_total[context] \* 1024);  compare = (range\_diff \* p) / 1024;  if (compare == 0) {  compare = 1;  } else if (compare == range\_diff) {  compare = range\_diff - 1;  }  value = in\_leading - range\_lower;  int s = 0;  if (value < compare) {  range\_upper = range\_lower + compare;  } else {  s = 1;  range\_lower = range\_lower + compare;  }  while (true) {    if (range\_upper <= 512) {  range\_lower = range\_lower << 1;  range\_upper = range\_upper << 1;  if (in\_index < max\_index) {  in\_leading = (in\_leading << 1) + **arithmetic\_stream[in\_index]**;  in\_index++;  } else {  in\_leading = in\_leading << 1;  }  } else if (range\_lower >= 512) {  range\_lower = (range\_lower - 512) << 1;  range\_upper = (range\_upper - 512) << 1;  if (in\_index < max\_index) {  in\_leading = ((in\_leading - 512) << 1) + **arithmetic\_stream[in\_index]**;  in\_index++;  } else {  in\_leading = (in\_leading - 512) << 1;  }  } else if (range\_lower >= 256 && range\_upper <= 768) {  range\_lower = (range\_lower - 256) << 1;  range\_upper = (range\_upper - 256) << 1;  if (in\_index < max\_index) {  in\_leading = ((in\_leading - 256) << 1) + **arithmetic\_stream[in\_index]**;  in\_index++;  } else {  in\_leading = (in\_leading - 256) << 1;  }  } else {  break;  }  }  range\_diff = range\_upper - range\_lower;  if (s == 0) {  counter[context]++;  }  counter\_total[context]++;  return s;  } | **1**  **1**  **1** | **vlclbf**  **vlclbf**  **vlclbf** |

|  |
| --- |
| AC\_resetCounter() {  for (i = 0; i < 7; i++) {  counter[i] = 16 / 2;  counter\_total[i] = 16;  }  } |

# Synthesizer (informative)

The synthesizer parses the input files and performs the high-level synthesis distribution between vectorial, wavelets, etc. The synthesis process then goes down to the Band component of the codec in which an Evaluation (or synthesis) process is called. Then all the bands of a given channel are mixed by a simple addition operator to recreate the desired haptic signal.

The evaluation method of a Band and the effects stored inside depends on the band type and other possible metadata to lead to the following switch case:

|  |
| --- |
| EvaluateBand(bandType):  case bandType == Transient:  EvaluateTransients(position)  case bandType == Curve:  EvaluateCurve(position, curveType)  case bandType == VectorialWave:  EvaluateVectorial(position, baseSignal)  case bandType == WaveletWave:  EvaluateWavelet(position) |

The band type directly impacts the evaluation of each effect stored inside the mentioned band. The effect synthesis will then match the following requirements:

* **Transient Effects:** each keyframe stored in the effect is configured by a timestamp and an amplitude. The synthesizer will generate two sine periods with a period length of 11 ms starting at the timestamp and the configured amplitude. In the case of two overlapping transients, an addition will be applied.
* **Curve Effects:** Each keyframe stored in the effect is configured by a timestamp and an amplitude. The synthesizer will compute the value at a given timestamp using an interpolation function between the two nearest keyframes of this timestamp. The interpolation function is selected by the *“curveType”* metadata. Clause 6.8 describes every useful information possible to recreate the synthesis function used here.
* **Vectorial Effects:** Each keyframe stored in the effect is configured by a timestamp, an optional amplitude and an optional frequency. The signal is generated by a waveform defined by the *“baseSignal”* metadata on which the amplitude and the frequency are configured at the keyframe level. To obtain the instantaneous amplitude at a given timestamp a linear interpolation is used in the interval of the two nearest keyframes containing an amplitude. If no keyframe contains an amplitude, 1 is the default value. To obtain the instantaneous amplitude at a given timestamp a linear chirp interpolation is in the interval of the two nearest keyframes containing a frequency. The instantaneous phase of the waveform is also computed at a given timestamp by taking the origin phase parametrized and propagating it through the effect by a phase matching between each chirp intervals.
* **Wavelet Effects:** some preprocessing has to be done from the descriptive format. On each effect, the original amplitude has to be restored by multiplying each sample contained in the keyframes with wavmax. Then, the inverse wavelet transform has to be applied, using the filters described in clause 10.1.8. Then, knowing the start time of the effect, the sample closest to the current timestamp has to be found to get the output sample.

One evaluation step takes a timestamp as input and outputs the amplitude value to be saved in the output .wav file at this timestamp. The complete vector of points is then sent to a WAV generator to output the synthesized file.

1. **JSON schema reference**(normative)

The JSON schema of the HJIF interchange format is provided as an electronic attachment. The attachment contains the following list of files:

|  |  |
| --- | --- |
| **MPEG\_haptics.schema.json** | JSON schema defining a haptic experience A detailed description of the schema is given in clause 7.2.1. |
| **MPEG\_haptics.avatar.schema.json** | JSON schema defining a haptic avatar. A detailed description of the schema is given in clause 7.2.2. |
| **MPEG\_haptics.perception.schema.json** | JSON schema defining a haptic perception. A detailed description of the schema is given in clause 7.2.3. |
| **MPEG\_haptics.reference\_device.schema.json** | JSON schema defining a haptic reference device. A detailed description of the schema is given in clause 7.2.4. |
| **MPEG\_haptics.channel.schema.json** | JSON schema defining a haptic channel within a perception. A detailed description of the schema is given in clause 7.2.5. |
| **MPEG\_haptics.vector.schema.json** | JSON schema defining the local direction on which the signal should be defined. A detailed description of the schema is given in clause 7.2.6. |
| **MPEG\_haptics.band.schema.json** | JSON schema defining a haptic band within a haptic channel. A detailed description of the schema is given in clause 7.2.7. |
| **MPEG\_haptics.effect.schema.json** | JSON schema defining a haptic effect within a haptic band. A detailed description of the schema is given in clause 7.2.8. |
| **MPEG\_haptics.keyframe.schema.json** | JSON schema defining a keyframe within a haptic effect. A detailed description of the schema is given in clause 7.2.9. |

1. **OHM metadata input file format**(informative)

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bytes** | **Data format** |
| file\_description () {  format\_id\_string  format\_version  number\_of\_haptic\_elements  description\_string  for (i=0; i<number\_of\_haptic\_elements; i++) {  haptic\_element\_file\_name  element\_description\_string  number\_of\_haptic\_channels  for (i=0; i<number\_of\_haptic\_channels; i++) {  channel\_description\_string  channel\_gain  body\_part\_mask  }  }  } | 4  2  2  32  64  32  2  32  4  4 | char  unsigned int  unsigned int  char  char  char  unsigned int  char  single float  unsigned int |

format\_id\_string – unique character identifier “OHM”

format\_version – version number of the file format: 1.

number\_of\_haptic\_elements – number of haptic elements compiling the content. An element typically maps to an end-user haptic device.

description\_string – description string containing a human readable content description. If shorter than 32 bytes, it is followed by padding null characters. If the string is 32 bytes long, the string is terminated without a null character.

haptic\_element\_file\_name – description string containing the file name of the according haptic element file (.wav, .ivs, .ahap, .hjif). If shorter than 64 bytes, it is followed by padding null characters. If the string is 64 bytes long, the string is terminated without a null character. Note that an element might include more than one channel. This file is assumed to be located in the same directory as the OHM file (i.e. same path).

element\_description\_string – description string containing a human readable content description. If shorter than 32 bytes, it is followed by padding null characters. If the string is 32 bytes long, the string is terminated without a null character.

number\_of\_haptic\_channels – number of simultaneous channels for each haptic element (up to 65535).

channel\_description\_string – description string containing a human readable content description. If shorter than 32 bytes, it is followed by padding null characters. If the string is 32 bytes long, the string is terminated without a null character.

body\_part\_mask – binary mask specifying the body part(s) on which to apply the effect (as described in 4.6).

channel\_gain – a single precision float value that describes the amplitude gain for the haptic channel. A value of 1.0 indicates that the channel should be rendered at nominal voltage. Higher values indicate an overdrive state for the actuator.

1. **Semantic body part and actuator mapping**(informative)

The present annex describes one of the three different approaches of haptic spatialization on the human body as described in clause 6.6, as well as the associated actuator mapping. It uses a couple (body part, list of actuators) to specify how to render a haptic effect on the user.

# Semantic Body parts

As explained in clause 6.6, this approach is based on a semantic description of the body part where to play the haptic effect. It follows a set of predefined values, described in Table 9, composing the semantics used to target a part on the human body. It also includes a set of locational information (such as Up, Down, Left, Right, Top, Bottom), some operators (such as Plus, Minus) to specify the combinations and a set of nodes which refers to human body parts. The combination of these values of different types will create a semantic command which will target one are multiple body parts of the human body.

Figure 7 and Figure 38 show the hierarchical decomposition of the body model used to achieve this semantic identification. By combining the different values, it is possible to specify different levels of abstraction as described in Figure 37.



Figure 38 Hierarchical decomposition of the body parts.. For instance, the Left Body in orange can be specified with [“Head”, “Chest”, “Waist”, “Left Arm”, “Left Leg”]. The Right Hand with [“Right”, “Hand”], and the Right Leg with [“Right”, “].].Leg”].

This information is user-defined, some examples are provided in Figure 39 to help the reader to understand how this information should be used:

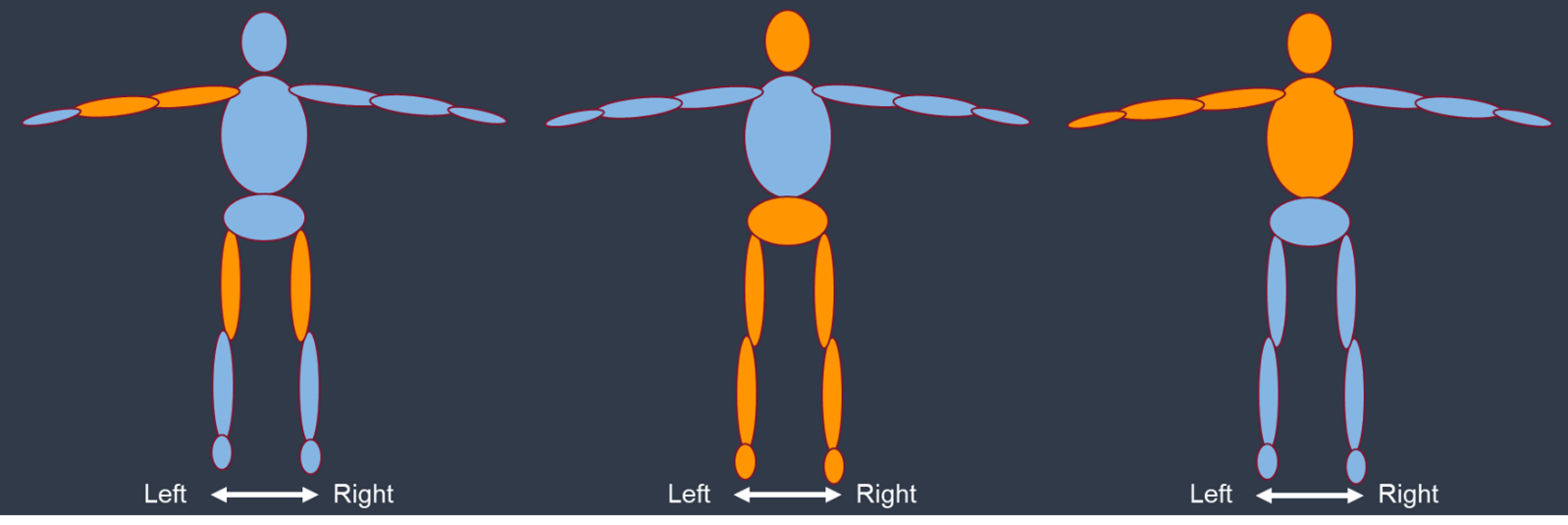


Figure 39 Some examples of segmented body parts.

From Figure 39, the respective body parts will be encoded with the following format:

* Figure 39 Left: The description in the interchange format will be ["Left”, "Arm”, "Minus”, "Hand”, "Plus”, "Thigh”]. The values in the distribution format will be [13, 20, 254, 32, 255, 35]
* Figure 39 Middle: The description in the interchange format will be ["Head”, "Plus”, "Down”]. The values in the distribution format will be [21, 255, 11]
* Figure 39 Right: The description in the interchange format will be ["Top”, “Minus”, "Left”]. The values in the distribution format will be [10, 254, 13]

# Actuator mapping

In addition to the previously presented body part model, an actuator mapping model is defined, mapping effects from the selected body parts to a set of actuators (comprising a device). This mapping is composed of two different values described in clause 6.6 as ***actuatorResolution*** and ***actuatorTarget***.

The haptic effect can be targeted/mapped to one or a multitude actuator. An illustration of the model is shown in Figure 40 where the body part is defined as a space composed by 3D voxels. Each voxel will be interpreted as an actuator.

Funnel chart

Description automatically generated

Figure 40 Actuator resolution illustration

The ***actuatorTarget*** parameter is simply a list of coordinates on which each coordinate corresponds to an actuator. Each targeted actuator should then receive the haptic signal defined by the current haptic channel. The origin of this 3D space representing the body part is its barycenter and the axis are defined in section 6.6.

In the Figure 40 example, the haptic resolution used will be [2, 2, 2]. After that if the user wants to target the red dot specifically, he will need to precise its coordinates with the value [1, 1, 0].

# Up/down-sampling actuator resolution

If the haptic device connected to a haptic player on which the experience is played doesn’t have the same actuator resolution as the one defined in the codec, the synthesizer should be able to remap/resample the actuators grid.

For this up/down-sampling, only the haptic resolution of the connected haptic device is needed. Then a mapping function can be designed to target the good actuators sampling as shown in the example Figure 41. For up-sampling cases, the presentation engine can remap the actuator targeted to a set of actuators placed in the same region. For down-sampling cases, the presentation engine can search for the closest actuator present on the device. Any other method can also be used, it is application dependent.

A picture containing text, cup, indoor, coffee cup

Description automatically generated

Figure 41 Example redirection function on the actuator targeting

**Bibliography**

[1] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

[2] Said, Amir & Pearlman, William. (2001). A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. IEEE Transactions on Circuits and Systems for Video Technology. 6. 10.1109/76.499834.

[3] Sobolewski, J. S. (2003). Cyclic redundancy check. *Encyclopedia of Computer Science* (pp. 476-479).