Watermarking API for Encoder Integration

March 2021
Revision: 1.0.1

Ultra HD Forum
8377 Fremont Blvd., Suite 117,
Fremont, CA 94538
UNITED STATES
Notice

This document is intended to serve the public interest by providing recommendations and procedures that promote uniformity of product, interchangeability and ultimately the long-term reliability of audio/video service transmission. This document shall not in any way preclude any member or non-member of the Ultra HD Forum from manufacturing or selling products not conforming to such document, nor shall the existence of such document preclude their voluntary use by those other than Ultra HD Forum members, whether used domestically or internationally.

The Ultra HD Forum assumes no obligations or liability whatsoever to any party who may adopt the API. Such adopting party assumes all risks associated with adoption of this API and accepts full responsibility for any damage and/or claims arising from the adoption of such API.

Attention is called to the possibility that implementation of the recommendations and procedures described in this document may require the use of subject matter covered by patent rights. By publication of this document, no position is taken with respect to the existence or validity of any patent rights in connection therewith. Ultra HD Forum shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of the recommendations and procedures described in this document have been requested to provide information about those patents and any related licensing terms and conditions.

All Rights Reserved
© Ultra HD Forum. 2021

Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision 1.0</td>
<td>February 2021</td>
</tr>
<tr>
<td>Revision 1.0.1</td>
<td>March 2021</td>
</tr>
</tbody>
</table>
Table of Contents

1 Purpose and Scope................................................................. 4
2 Definitions, Acronyms and References........................................ 5
  2.1 Acronyms........................................................................... 5
  2.2 References......................................................................... 5
3 Introduction........................................................................... 6
  3.1 WMP System Interfaces ......................................................... 6
  3.2 Frame-based Processing ....................................................... 7
  3.3 Shared Buffers .................................................................... 7
4 Main Data Structures .............................................................. 8
  4.1 Video Content Description ................................................... 8
  4.2 Video Frame Description ..................................................... 11
  4.3 Watermarked Frames Description ........................................ 12
5 Processing Modes ................................................................. 13
6 Application Programming Interface .......................................... 14
  6.1 WMP Instance Creation ....................................................... 14
  6.2 Content Initialization .......................................................... 15
  6.3 Frame Processing .............................................................. 16
  6.4 Content Purge ................................................................... 18
  6.5 WMP Instance Termination ................................................ 18
  6.6 API Return Status Codes .................................................... 19
Annex A Call Sequence Diagram ................................................. 20
  A.1 Introduction....................................................................... 20
  A.2 Without a Callback Function ............................................. 20
  A.3 With a Callback Function .................................................. 23
Annex B In-Band Metadata ........................................................ 26
  B.1 Introduction....................................................................... 26
  B.2 AVC Metadata Inclusion .................................................... 26
  B.3 HEVC Metadata Inclusion ................................................ 26
  B.4 AV1 Metadata Inclusion ..................................................... 27
1 PURPOSE AND SCOPE

Background knowledge about commercially deployed forensic watermarking systems has been presented in [1]. With a focus on integration, it helps stakeholders understanding what needs to be done.

In this document, for the specific use case of Adaptive Bitrate content, APIs are proposed. This allows an encoder and a watermarking system to process video content and output Variants. There are two flavours of the API, one for watermarking technologies that process baseband video content and one for watermarking technologies that process compressed video content.
2 DEFINITIONS, ACRONYMS AND REFERENCES

In addition to definitions, acronyms and references defined in [1], the following is added.

2.1 Acronyms

API       Application Programming Interface
EBP       Encoder Boundary Point
ENC       Encoder
HDR       High Dynamic Range
IDR       Instantaneous Decoder Refresh point
NALU      Network Access Layer Unit
RAP       Random Access Point
SEI       Supplemental Enhancement Information
VCL       Video Coding Layer
WMP       Watermark Pre-processor

The types ((u)intN_t) used in this document are defined in [2]. See also <inttypes.h> header, https://en.wikipedia.org/wiki/C_data_types#inttypes.h, and <stdint.h>.

2.2 References

3 INTRODUCTION

For watermarking with Variants creation, video content is pre-processed and integration with the encoder is required, especially in the context of Live where latency is critical. The video encoder (ENC) will provide information about the stream, content and location and the Watermark pre-processor (WMP) will create Variants based on its own technology.

This document provides a generic C-language API for the WMP both for baseband and compressed watermarking technologies. Depending on implementation specifics, only a subgroup of these parameters may be required, and some functionality or parameters may be out of scope. This API is not limited to A/B use case only (generation of two Variants), it allows generating any number of Variants from the input frame. This document extends the guidelines provided in [1]. For example, in the case of ABR content creation as shown in Figure 1 (DASH or HLS with two Variants), the API defined in this document is on the link between the ENC and WMP.

![Figure 1: Scope of the API defined in this document.](image)

3.1 WMP System Interfaces

From a system perspective, the WMP is a software component that accepts as input a video content and outputs one or several Variants, typically two in the case of A/B watermarking as shown in Figure 2.

![Figure 2: WMP interfaces.](image)

This WMP needs to be integrated within the video processing pipeline of the ENC. Depending on the watermarking technology, it will be placed either before the encoding module of the ENC if the WMP processes baseband video content, or after the encoding module if it processes compressed video content. Although this document provides a generic API for both types of watermarking systems, the actual integration in the ENC will be de facto different. In both cases, the ENC is responsible for packaging the produced...
Variants using relevant transport protocols. It is possible to configure the WMP to produce all the Variants of the watermarking system at once or, alternately, to produce only one of these Variants. In the latter case, several ENC will need to be deployed, each one producing a single Variant, in order to have all necessary Variants in the watermarking system.

In addition to the Variants, the WMP may also generate metadata and, more specifically, in-band and/or out-of-band metadata.

The ENC is responsible for encapsulating the in-band metadata in the video stream it outputs (see. Annex B). Depending on which component of the video delivery network shall consume WMP in-band metadata, it may be required to be encrypted or not. For instance, WMP in-band metadata may contain A/B redirection information to be used in the Edge servers of a CDN. In this case, the WMP in-band metadata shall be left in cleartext at the transport layer in order to be accessible at such locations. Alternately, WMP in-band metadata may contain information to perform just-in-time watermarking operations in the end-user device. In that case, the metadata shall be secured at the video transport layer in a way similar to the video content it is bound to.

The WMP may also produce out-of-band metadata that is required to perform watermark detection. This is watermarking vendor-specific metadata exchanged between the WMP and the watermark detector and is typically stored locally on disk by the WMP. The ENC shall then manage this data in a manner defined by the WMP provider. As an example, the WMP provider may request that the file is retrieved and pushed on an archive on a Cloud service on a regular basis. Another requirement can be on the file size that may not exceed a given value. All these elements are to be defined at integration time and are vendor specific.

### 3.2 Frame-based Processing

The WMP operates frame by frame: for each input video frame, the WMP returns Variants of the frame. Depending on whether the watermarking technology operates in the baseband or compressed domain, the content of a ‘frame’ will actually be different. In the case of baseband watermarking, a frame is a raw image represented by a buffer filled with pixel values and all frames have the same size. For compressed domain watermarking system, a frame is a portion of the encoded video stream e.g. an Access Unit (AU) using MPEG terminology. As a result, frames may have different sizes.

Another salient difference between baseband and compressed domain watermarking relates to the delivery schedule of the input frames. In baseband watermarking, the WMP is placed before the encoding module of the ENC and is therefore expecting input frames to be given in presentation order, typically at a regular pace depending on the frame rate of the input video feed. In contrast, with compressed domain watermarking systems, the WMP is typically placed after the encoding module of the ENC and expects input frames to be given in decoding order. In this setup, input frames may arrive in bursts according to the GOP structure.

For ABR distribution, the WMP may require knowing how the video bitstream will be eventually segmented for OTT delivery. This is the reason why dedicated signalling may be used to indicate if a given frame processed by the WMP will be the beginning of a segment at delivery time.

### 3.3 Shared Buffers

The WMP and the ENC have shared buffers for exchanging content (input frames and metadata and marked frames and metadata). The management of these buffers is under the ENC responsibility, the WMP is only responsible for managing its internal memory. Note that this internal memory could be a copy of these shared buffer, but it is not recommended for efficiency purposes. Depending on the operating modes that are expressed with several parameters, the ENC needs to release memory following the guidelines presented in Section 5. For example, when a shared buffer can be released depends on the use or not of a dedicated callback function.

Additional optimization of the memory management is possible with ad-hoc functions that are not defined in this document.
4 MAIN DATA STRUCTURES

This API uses a few data structures to describe (i) the video content that will be processed by a WMP instance, (ii) a video frame given in input to the WMP, and (iii) the Variants returned by the WMP.

4.1 Video Content Description

The description of the video content is encapsulated in a fixed-size data structure that contains the information listed below:

```c
typedef struct {
    uint32_t width;    // Width of a video frame in pixels
    uint32_t height;    // Height of a video frame in pixels
    int32_t stride;    // Number of pixels between consecutive lines
        // May be negative (for example for RGB space)
    wmp_yuv_format_t yuv_format;  // Raster format
    wmp_format_t format;   // Interlacing mode
    wmp_hdr_mode_t hdr_mode;   // HDR description
    uint32_t fps_num;    // Numerator of the fps e.g. 30000
    uint32_t fps_den;    // Denominator of the fps e.g. .1001 for 29.97
    wmp_codec_t codec;   // Codec used to encode the video content
    uint32_t seg_min;    // Min segment duration in frames
    uint32_t seg_max;    // Max segment duration in frames
    wmp_mode_t enc_mode;   // Encoding mode
    int8_t content_identifier[256]; // Identifier of the content, null terminated
} wmp_content_description_t;
```

4.1.1 Raster Format

An exhaustive list of raster formats is defined in ffmpeg (ffmpeg.exe -pix_fmts). Note that it also includes the data size of pixels and bit depth. For example, a YUV 4:2:0 progressive video content with 8-bit depth is represented by yuv420p. The API of the WMP supports the following raster formats:

```c
typedef enum {
    gray                 =  0,
    yuv420p              =  1,
    yuv444p              =  2,
    yuv420p10le          =  4,
    yuv420p12le          =  7,
    yuv422p10le          =  5,
    yuv444p10le          =  6,
    yuv422p12le          =  8,
    yuv444p12le          =  9,
    yuv_format_undefined = 10
} wmp_yuv_format_t;
```

4.1.2 Interlacing

The API of the WMP supports the following modes of interlacing:

```c
typedef enum {
    FF_INTERLACED_TOP_FIRST    = 0, // Interlaced format with odd/even indication
    FF_INTERLACED_BOTTOM_FIRST = 1, // Interlaced format with odd/even indication
    FF_PROG                    = 2, // Progressive mode
    FF_3DSBS                   = 3, // 3-D Side-by-Side
};
```
4.1.3 Dynamic Range

The API of the WMP supports the following dynamic ranges:

```c
typedef enum {
    SDR = 1,
    HDR10 = 2,
    HDR10PLUS = 3,
    HLG = 4,
    DOLBYVISION = 5,
    SL_HDR_1 = 6,
    range_undefined = 99
} wmp_hdr_mode_t;
```

This API does not make any assumption on clamping. If required, this can be added either when integrating the WMP and ENC or when deploying.

4.1.4 Video Codec

The API of the WMP supports the following video codecs:

```c
typedef enum {
    RAW = 0,
    AVC = 1,
    HEVC = 2,
    AV1 = 3,
    MPEG4 = 4,
    VP9 = 5,
    VP10 = 6,
    VVC = 7,
    EVC = 8,
    LCEVC = 9,
    codec_undefined = 99
} wmp_codec_t;
```

This parameter is mandatory for watermarking technologies operating in the compressed domain. It is optional for watermarking technologies operating in the baseband domain, in which case the value codec_undefined shall be used. However, if in-band metadata is produced by the watermarking technology, this parameter shall be properly set to format the metadata accordingly (see Annex B).

4.1.5 ABR Segmentation

The parameters `seg_min` and `seg_max` define the boundaries length of any segment under any segmentation process. They enable the WMP to distribute the watermark embedding accordingly to guarantee maximum of a single variation per segment. Values should be as accurate as possible and setting `seg_min=seg_max` allows for the most efficient bit distribution. `seg_max` is an upper bound.

Note: This may be required for VOD content when every segment is not necessarily carrying watermark information and a certain number of bits must be embedded per a given duration (e.g. 10 bits for 5 minutes). These parameters are then helpful for proper calculation of the bit distribution.
4.1.6 Encoding Mode

As shown in Figure 3, in VOD content preparation workflow, the WMP may be used in two passes as are video encoders. The first pass is used to analyse the video content to be processed and the second pass actually generates the Variants. For Live content preparation workflow, because of real-time constraints, there is no possibility to have a second pass.

Figure 3: Two pass encoding workflow where mode = ENC_TWO_PASS.

The API of the WMP supports the following encoding modes:

```c
typedef enum {
    ENC_ONE_PASS = 1, // VBR/CBR encoding using one pass
    ENC_TWO_PASS = 2, // VBR/CBR encoding using two passes
    ENC_LIVE = 3      // Live encoding
} wmp_mode_t;
```

It is expected that the WMP instance manages internally the status of encoding a frame when in ENC_TWO_PASS mode. It means that the WMP knows whether this is the first or second pass when receiving a frame to process.
4.2 Video Frame Description

In essence, the WMP operates frame by frame. Each video frame that is fed to the WMP is described using a data structure that contains the information listed below:

```c
typedef struct {
    wmp_frame_type_t   frame_type;  // Type of frame
    uint32_t           frame_number; // Frame sequence number in presentation order
    uint64_t           PTS;           // Presentation Time Stamp (see [5])
    uint64_t           DTS;           // Decoding Time Stamp (see [5])
    uint32_t           encoder_id;   // ENC identifier
    uint32_t           seq_number;   // Sequence number
    uint8_t            frag_start;   // New EBP fragment indicator: 0 or 1
    uint8_t            seg_start;    // New EBP segment indicator: 0 or 1
    uint32_t           frame_buffer_len; // Length of the frame buffer
    uint8_t*           frame_buffer; // Frame in specified raster or codec format
    uint8_t*           enc_data;     // Blob containing ENC vendor-specific data
} wmp_frame_description_t;
```

### 4.2.1 Type of Frame

The WMP may adjust its watermarking strategy depending on the type of the video frame. The API of the WMP supports the MPEG types of frames listed below. If the ENC cannot provide this information, this field shall be set to the value `NN`. If an I-frame is also an IDR frame, then it shall be declared as an IDR frame.

```c
typedef enum {
    NN   = 0,
    IDR  = 1,
    I    = 2,
    P    = 3,
    B    = 4,
    b    = 5 // (non referenced B-frame)
} wmp_frame_type_t;
```

### 4.2.2 PTS and DTS

These parameters are not relevant for watermarking technologies operating in the baseband domain. In that case, they can be arbitrarily set to zero.

### 4.2.3 Encoder Identifier

A WMP instance may generate all the Variants of the watermarking system or, alternately, it may generate only one of them. In the latter case, several ENCs will run in parallel and the field `encoder_id` will let the WMP know which encoder it is interacting with and which Variant shall be generated. Examples are provided in Section 6.3.1.

### 4.2.4 Sequence number and ABR Segmentation

In addition to the high level ABR segmentation signalling defined in Section 4.1.5, the WMP may require to know if a video frame is the beginning of an ABR segment. This information can be inferred if the ENC can provide a sequence number (`seq_number`) for each frame. Alternately, the ENC can rely on Encoder Boundary Point (EBP) signalling such as defined in [5]. The binary flags `frag_start` and `seg_start` can
be used to indicate the beginning of short fragments (typically a DASH 2 seconds segment) and long segments (typically an HLS 6 seconds segment) respectively. When used, seq_number follows this rule: first = 1 (0 if unavailable).

4.2.5 Frame Buffer

If the WMP operates with baseband video, the buffer frame_buffer contains the pixel values of a video frame according to the pixel format defined in the content description (see Section 4.1). In contrast, if the WMP operates in the compressed domain, the buffer frame_buffer contains a portion of encoded video e.g. an MPEG Access Unit. In both cases, the length in bytes of the frame buffer is given by frame_buffer_len.

4.3 Watermarked Frames Description

The structure wmp_mark_description_t is composed of:

```c
typedef struct {
    uint32_t   nb_variants;    // Number of Variants
    wmp_frame_description_t variants[max_variants];  // Generated Variants
    uint8_t   inband_metadata_size;  // Metadata size
    uint8_t inband_metadata[max_inband_metadata_size]; // SEI to be added
} wmp_mark_description_t;
```

4.3.1 Number of Created Variants

For a given input video frame, the WMP may provide one or several Variants. If the frame is not watermarked, then the nb_variants variable will be set to zero (0); if it is watermarked, then there are several Variants (see examples in Section 6.3.1) and the number of Variants is given by the value of nb_variants, i.e. 1, 2… depending on the configuration. Note that nb_variants cannot take values higher than max_variants defined in WMP_Open. If parallel encoders are used, then each encoder will output one Variant and, in this case nb_variants=1 is the maximum value.

4.3.2 In-band Metadata

The ENC shall examine these two elements in order to determine whether it shall introduce extra metadata in the video bitstream:

1. inband_metadata_size: The effective size of inband_metadata in bytes.
2. inband_metadata: It is the concatenation of the UUID and watermarking_metadata of the payload for a Non-VCL units as described in Annex B depending on the codec.

This is optional, if no metadata is provided, the inband_metadata_size shall be explicitly set to zero (0). It contains vendor-specific data.

---

1 The ‘fragment’ and ‘segment’ terminology in the definition of frag_start and seg_start must be understood as defined in [6]. They are not CMAF fragments/segments.
5 PROCESSING MODES

The following principles apply for the management of the shared buffers:

- The ENC is the owner of the memory buffers, namely frame_description (see section 4.2), representing the input frame, and mark_description (see section 4.3), representing the watermarked Variant. It means that ENC must pre-allocate these buffers before WMP_ProcessFrame is invoked.

- WMP sets the value of the nb_variants variable in mark_description to zero or a positive value (the maximum been defined by max_variants), in order to communicate “non-watermarked vs. number of Variants” frame, respectively – see the explanation of nb_variants in Section 4.3.1.

There are, theoretically, several processing modes that are possible with the parameters of the API defined in Section 6. The processing modes are governed by:

- The use of a callback function or not: If a callback function is used, the memory can only be released when the callback is invoked.

- The value of nb_variant: There are optimization cases allowing to use the frame_description content as replacement for the values for the mark_description and thereby avoid the need to copy full frame buffers.

- The values in delay_description: This allows the WMP to tell the ENC that WMP will add some delay, hence impacting when shared memory can be released and how much memory shall be budgeted by the ENC.

Based on the feedback from vendors with existing implementations, the three following cases are detailed as they represent models to be supported at the time of writing this document:

- Without a callback function and an added delay of n frames (n could be equal to 0) and with the possibility that nb_variant can sometimes be equal to 0.
  - Because of the delay of n frames and the absence of a callback function, the ENC must keep the shared buffer of frame_description for n+1 calls of the processing function. It provides the WMP with an access to the input content as long as necessary for generating the Variants. When nb_variant is equal to 0, the ENC needs to get from the input frame_description the (non-modified) Variants that will be used in the next steps of the processing chain. Shared memory mark_description can always be released after exiting from the processing function and once the ENC has made use of it.

- Without a callback function and an added delay of n frames (n could be equal to 0) and nb_variant cannot be equal to 0.
  - This case is very similar to the previous one, the ENC must keep the shared buffer frame_description for n+1 calls of the processing function. However, since nb_variant is always different from 0, Variants are always provided back in the mark_description that is part of the shared memory. As in the previous case, mark_description can always be released after exiting from the processing function and once the ENC has made use of it.

- With a callback function and an added delay of maximum t milliseconds and nb_variant cannot be equal to 0.
  - The management of the memory is driven by the invocation of the dedicated callback function. When the ENC receives a corresponding callback call, any shared buffer frame_description and mark_description that is an argument of the callback function can be released as soon as the ENC made its own processing with the mark_description content. The ENC always get Variants in mark_description as nb_variant cannot be equal to 0.
6 APPLICATION PROGRAMMING INTERFACE

The WMP functionality is organized in five stateful API calls:

1. `WMP_Open`, which creates a WMP instance to be shared among all subsequent calls – see Section 6.1
2. `WMP_InitContent`, which initializes the earlier created instance with the content description (i.e. what content the instance is supposed to process) – see Section 6.2
3. `WMP_ProcessFrame`, which is called for each frame of the content to be processed by the WMP instance – see Section 6.3
4. `WMP_ClearContent`, which clears the content description of the WMP instance – see Section 6.4
5. `WMP_Close`, which destroys the WMP instance – see Section 6.5

All functions of the API return with one of the status codes defined in Section 6.6.

This document describes version 1.0 of the API.

6.1 WMP Instance Creation

This function is invoked to initialize the WMP. It takes in input several configurations parameters as well as pointers to callback functions that may be invoked by the WMP. If successful, this function returns a pointer to a data structure containing relevant information to operate the created WMP instance.

```c
wmp_status_t WMP_Open(uint32_t init_data_size,
                        const void  *init_data,
                        uint32_t   wm_id_length,
                        uint32_t   max_variants,
                        uint32_t   use_EBP_signal,
                        void  *log_message_callback,
                        void  *mark_ready_callback,
                        void **instance)
```

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_data_size</td>
<td>Size in bytes of the blob of data init_data.</td>
</tr>
<tr>
<td>init_data</td>
<td>Blob of data containing configuration parameters for the WMP. The exact structure of this blob of data depends of the watermarking vendor and could include, for example, the path for the file to be created by the WMP output (see Annex B). The ENC is responsible for allocating and filling this data structure with relevant information. This structure is unaltered by the function. The ENC obtains the data for filling this blob in a WMP vendor-specific manner.</td>
</tr>
<tr>
<td>wm_id_length</td>
<td>Number of symbols composing the watermarking identifier to be embedded at a later stage. This may be present in init_data, in this case, this must be the same value.</td>
</tr>
<tr>
<td>max_variants</td>
<td>Maximum number of Variants that this WMP instance shall create. This may be present in init_data, in this case, this must be the same value.</td>
</tr>
<tr>
<td>use_EBP_signal</td>
<td>Set to 1 if the EBP signalling in frame_description is used. Set to 0 otherwise. This may be present in init_data, in this case, this must be the same value.</td>
</tr>
<tr>
<td>log_message_callback</td>
<td>Callback function used by the WMP to escalate logs to the ENC.</td>
</tr>
</tbody>
</table>
mark_ready_callback | Optional callback function required for the WMP to operate in case Variants are not provided when WMP_ProcessFrame() returns. It shall be set to NULL if it is not used.
---|---
instance | The created WMP instance will be returned via this parameter. It will be set to NULL in case of error.

### 6.1.1 Log Callback Function

This callback function is invoked by the WMP to provide log information (error, warning, info, debug) to the ENC. The ENC shall incorporates these log messages as part of its general logs. This callback function shall return one of the status code defined in Section 6.6 to indicate whether or not the log message has been successfully processed by the ENC or not.

```c
typedef wmp_status_t (*log_message_callback)(wmp_logs_level_t log_level,
                                          const char *log_message)
```

The `log_level` values are in descending order, meaning that the lower the value is, the higher the importance of the log information is. The different log level values are:

```c
typedef enum {
    ERROR = 0,
    WARNING,
    INFO,
    DEBUG,
    MAX_LOGS
} wmp_logs_level_t;
```

### 6.1.2 Mark Ready Callback Function

This callback function is invoked when the Variants generated by the WMP are not provided when WMP_ProcessFrame() returns. In that case, the WMP invokes this callback to notify the ENC that a frame has been fully processed. The ENC shall then push the generated Variants further downstream. Moreover, invoking this callback indicates that the WMP will no longer reference the original input frame as well as the watermarked frames and that the ENC can reuse the underlying shared memory buffers. This callback function shall return one of the status code defined in Section 6.6 to indicate whether or not the ENC has successfully processed the provided frames and Variants.

```c
typedef wmp_status_t (*mark_ready_callback)(wmp_frame_description_t *frame_description,
                                           wmp_mark_description_t *mark_description)
```

### 6.2 Content Initialization

This function takes in input the description of the video content to be processed and attaches it to a created WMP instance. The WMP returns information about its processing delay and the amount of in-band metadata that may need to be incorporated to the output video content.

```c
wmp_status_t WMP_InitContent(const wmp_content_description_t *content_description,
                              wmp_delay_description_t *delay_description,
                              uint16_t *max_inband_metadata_size,
                              void *instance)
```
content_description Pointer to a data structure containing the description of the video content to be processed by the WMP instance (see Section 4.1). The ENC is responsible for allocating and filling this structure. It is unaltered by the function.

delay_description Pointer to a data structure containing information about the processing delay introduced by the WMP (see below). The ENC is responsible for allocating this structure and the WMP fills it with relevant information.

max_inband_metadata_size This is the maximum size in bytes of inband_metadata part of the mark_description structure. This value is returned by the WMP so that the ENC can properly pre-allocate the memory buffers of Variants.

instance Pointer to a created WMP instance that shall process the frames of content_description.

6.2.1 Processing delay

When the WMP processes video content, it may introduce some delay in the video processing pipeline. Such delay may be dependent on some characteristics of the video content, e.g. the resolution, the video codec, the video bitrate, etc. To notify the ENC of this processing delay, the WMP fills a data structure previously allocated by the ENC.

typedef enum {
    NO_DELAY = 0, // No delay is added (delay value shall be set to 0)
    TIME     = 1, // Time delay expressed in milliseconds
    FRAME    = 2  // Frame delay expressed in number of frames
} wmp_delay_type_t;

typedef struct {
    wmp_delay_type_t delay_type; // Type of delay
    uint32_t     value;  // Effective delay introduced (ms or frames)
} wmp_delay_description_t;

The data structure wmp_delay_description_t first indicates if the introduced delay will be expressed in milliseconds (time delay) or in number of frames (frame delay). Usually, when the WMP provides the Variants when WMP_ProcessFrame() returns, the delay is a number of frames whereas it is rather expressed in milliseconds if a callback function is used. Such information is expected to help the ENC to properly provision buffers that will be shared with the WMP.

If the WMP does not introduce any delay, the type of delay shall be set to NO_DELAY and the value set to 0.

6.3 Frame Processing

This method is called repeatedly to process successive frames of the video content and produce Variants. For baseband watermarking technologies, frames should be passed in presentation order. Conversely, for compressed domain watermarking technologies, frames shall be passed in decoding order.

wmp_status_t WMP_ProcessFrame(const wmp_frame_description_t *frame_description, wmp_mark_description_t *mark_description, void *instance)
Data structure containing the description of the watermarked frames generated by the WMP (see Section 4.3). The ENC is responsible for allocating this structure. WMP fills it with relevant information. It is provided for every frame.

Instance of the WMP to be used for processing the input frame.

### 6.3.1 Examples of WMP Parameters Settings

The following are examples of setting some parameters in the frame\_description and mark\_description structures.

**Example 1:** An ENC receives one frame and outputs two Variants created by the WMP. Optionally, the WMP may decide to output no Variant or only one Variant if there is no difference between A/B (for example if adding only a time stamp).

- **max\_variants = 2**  
  // General parameter of the instance.
- **nb\_variants = 0, 1 or 2**  
  // If equal to 0, the WMP does not watermark the frame. If equal to 1, the WMP produced a single Variant and the ENC can overlook the content of variants[1].
- **encoder\_id = 1**

**Example 2:** An encoder receives one frame and outputs three Variants created by the WMP. Optionally, the WMP may decide to output no Variant or only one Variant if there is no difference between A/B/C.

- **max\_variants = 3**  
  // General parameter of the instance.
- **nb\_variants = 0, 1 or 3**  
  // If equal to 0, WMP does not watermark the frame. If equal to 1, the WMP produced a single Variant and the ENC shall overlook the content of variants[1] and variants[2].
- **encoder\_id = 1**

**Example 3:** Two ENCs are used in parallel, both receive the same frame and each encoder outputs one Variant.

-
For each transcoder:

- `max_variants = 1`
  
  // General parameter of the instance.

- `nb_variants = 0` or `1`
  
  // If equal to 0, the WMP does not watermark the frame. Each transcoder outputs only one Variant.

- `encoder_id = 1` or `2`
  
  // Each encoder has a different ID.

**Example 4:** Three ENCs are used in parallel; each one receives the same frame and outputs one Variant.

For each transcoder:

- `max_variants = 1`
  
  // General parameter of the instance.

- `nb_variants = 0` or `1`
  
  // If equal to 0, the WMP does not watermark the frame. Each transcoder output only one Variant.

- `encoder_id = 1` or `2` or `3`
  
  // Each encoder has a different ID.

### 6.4 Content Purge

The ENC invokes this function to purge a WMP instance prior to its termination or its reconfiguration to process another video content having possibly different characteristics. Once this function has returned, the WMP will never invoke the `mark_ready_callback` function if a callback function is defined and will never reference shared buffers provided previously. Before exiting, several callback could be triggered, if used, in order to flush the shared buffer in the pipeline. If no callback function is used, it is the responsibility of the ENC to release shared memory, as a consequence, few frames may not be marked at the end of the content.

```c
wmp_status_t WMP_ClearContent(void *instance)
```

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Instance of the WMP for which the content descriptor is reset.</td>
</tr>
</tbody>
</table>

### 6.5 WMP Instance Termination

The ENC invokes this function to terminate a running WMP instance.

```c
wmp_status_t WMP_Close(void *instance)
```

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Instance of the WMP to be terminated.</td>
</tr>
</tbody>
</table>
6.6 API Return Status Codes

The following status code are defined as part of this API:

typedef enum {
    WMP_SUCCESS = 0,
    WMP_E_MEM_ALLOC = 1,  // Error during the memory allocation
    WMP_E_MEM_FREE = 2,  // Error during the memory release
    WMP_E_INIT_FORMAT = 3,  // Error during the processing of init_data
    WMP_E_CONTENT_FORMAT = 4,  // Error during the processing of content_description
    WMP_E_WM_ID_LENGTH_RANGE = 5,  // wm_id_length is out of supported range
    WMP_E_MAX_VARIANTS_RANGE = 6,  // max_variants value is out of supported range
    WMP_E_MARK_READY_CALLBACK_MISSING = 7,  // Callback pointer is mandatory
    WMP_E_INIT = 8,    // Error other than those specified above during init
    WMP_E_FRAME_FORMAT = 8,  // Error occurred during reading frame_description
    WMP_E_GENERIC = 99  // Generic error code
} wmp_status_t;

Additional error codes can be defined by any WMP provider. These error codes must be above 99. These error codes are then specific to this WMP provider and to its implementation.
ANNEX A  CALL SEQUENCE DIAGRAM

A.1  INTRODUCTION
The following exemplifies two reference WMP flows and associated sequence calls with main parameters. Both examples assume that an A/B version of the input video content is expected and that a single WMP instance is created for creating these Variants (example 1 in the previous section). The flow presented in section A.2 is typically for a WMP working in the baseband space while the flow presented in section A.3 is for a WMP working in the compressed domain.

A.2  WITHOUT A CALLBACK FUNCTION

ENC

```
WMP_Open(wmp_instance)

WMP_InitContent(content_desc, wmp_instance)

WMP_ProcessFrame(frame_desc_1, mark_desc_1, instance)

WMP_ProcessFrame(frame_desc_2, mark_desc_2, instance)

WMP_ProcessFrame(frame_desc_3, mark_desc_3, instance)

WMP_ProcessFrame(frame_desc_4, mark_desc_4, instance)

WMP_ProcessFrame(frame_desc_N, mark_desc_N, instance)

WMP_ClearContent(content_desc, wmp_instance)

WMP_Close(wmp_instance)
```
Figure 4 is a typical flow when no callback function is defined. The watermarked frames are provided back when the function `WMP_ProcessFrame()` returns. In

For example, with a delay of two frames, `mark_desc_3` contains the watermarked frames for the frame provided in `frame_desc_1`. Assuming that the content is SDR 1080p at 24 fps, the relevant parameters in the API calls would be:

```
WMP_Open()
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init_data_size</code></td>
<td>Variable</td>
</tr>
<tr>
<td><code>init_data</code></td>
<td>Variable</td>
</tr>
<tr>
<td><code>wm_id_length</code></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>max_variants</strong></td>
<td>2 - creation of A/B Variants</td>
</tr>
<tr>
<td><strong>use_EBP_signal</strong></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>log_message_callback</strong></td>
<td>Pointer to a function</td>
</tr>
<tr>
<td><strong>mark_ready_callback</strong></td>
<td>NULL - no callback function</td>
</tr>
<tr>
<td><strong>instance</strong></td>
<td>Pointer to WMP</td>
</tr>
</tbody>
</table>

**WMP_InitContent()**

| **content_description** | width=1920;  
height=1080;  
stride=1920;  
yuv_format=1;  
format=2,  
hdr_mode=1;  
fps_num=24;  
fps_den=1;  
codec=99;  // Irrelevant parameter for baseband  
seg_min=48;  
seg_max=48;  
mode=2;  |

| **delay_description** | delay_type=2;  
value=2;  // mark_desc_(n+2) contains watermarked frames for the frame provided in frame_desc_(n)  |

| **max_inband_metadata_size** | Variable |
| **instance** | Pointer to WMP |

**WMP_ProcessFrame()**

| **frame_description** | All parameters are different per frame |
| **mark_description** | All parameters are different per frame |
| **instance** | Pointer to WMP |
Figure 4: Illustration of the calls sequence diagram when the WMP operates without a callback function.

A.3 WITH A CALLBACK FUNCTION

Figure 5 depicts a similar flow when a dedicated callback function is used to output watermarked frames.

In Figure 5, the function WMP_ProcessFrame() returns almost immediately and ENC can rapidly push several video frames to WMP if needed. WMP then invokes the callback mark_ready_callback() to notify ENC that it produced Variants for an input frame and that it will no longer reference the specified frame_description and mark_description. ENC can then reuse the memory allocated to these structures once watermarked content is pushed further in the content preparation pipeline.

In this example, assuming content is SDR 1080p at 24 fps encoded with AVC, the relevant parameters in the calls would be

WMP_Open()
<table>
<thead>
<tr>
<th>init_data_size</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_data</td>
<td>Variable</td>
</tr>
<tr>
<td>wm_id_length</td>
<td>Variable</td>
</tr>
<tr>
<td>max_variants</td>
<td>2 - creation of A/B Variants</td>
</tr>
<tr>
<td>use_EBP_signal</td>
<td>Variable</td>
</tr>
<tr>
<td>log_message_callback</td>
<td>Pointer to a function</td>
</tr>
<tr>
<td>mark_ready_callback</td>
<td>Pointer to a function</td>
</tr>
<tr>
<td>instance</td>
<td>Pointer to WMP</td>
</tr>
</tbody>
</table>

**WMP_InitContent()**

| content_description | Width=1920; Height=1080; Stride=1920; yuv_format=1; format=2, hdr_mode=1; fps_num=24; fps_den=1; codec=1; seg_min=48; seg_max=48; mode=2; |

| delay_description | delay_type=1; value=50; // The callback function provides watermarked data with 50 ms delay |

| max_inband_metadata_size | Variable       |
| instance                | Pointer to WMP |

**WMP_ProcessFrame()**

| frame_description | All parameters are different per frame |
| mark_description  | While been present and memory is allocated, watermarked data is not provided in this call |

**mark_ready_callback()**

| frame_description | All parameters are different per frame |
| mark_description  | All parameters are different per frame |
Figure 5: Illustration of the calls sequence diagram when the WMP operates with a callback function.
ANNEX B IN-BAND METADATA

B.1 INTRODUCTION

Encoders produce formatted streams in order to be compliant with the ingest format of the next in-line components or systems. In case of OTT, these next in-line can be Origins or JiT (Just-in-Time) Packagers. Encoders also make some initial formatting decisions (e.g. EBP, RAP, IDR, etc.). To avoid two information channels between an upstream component and a downstream component in the Live flows, where one channel is the video stream and the other channel is out-of-band metadata with locations, Encoders can introduce in-band metadata located within the encoded video bitstream.

When an Encoder calls WMP_ProcessFrame, it submits the frame_description structure with the metadata such as: frame_number or seq_number. In order to enable proper switching between Variants in the downstream components, an out-of-band embed locations file can be constructed. While this may work for VOD deployments, it is risky and introduces significant (repetitive) overhead in terms of out-of-band file copies for Live deployments.

The same effect can be achieved by an in-band artifact collocated with the actual video frames, where the switching between Variants can be recovered by quick scan of the final (formatted) stream while avoiding out-of-band synchronization challenges. Therefore, instead of having two channels and potential misalignments between them in Live deployments, a single channel carries all necessary information and metadata “piggy-backs” the existing infrastructure.

B.2 AVC METADATA INCLUSION

Metadata is added in the stream as non-VCL NALUs (user data unregistered SEI NAL) as described in Section 7.3.1 and Annex D of [3]. These NALUs are interleaved with VCL NALUs, which are carrying video frames data. Watermarking metadata can be seamlessly carried with other video bitstream Non-VCL data.

Table 1 gives the format of the SEI NALU that carries watermarking metadata.

<table>
<thead>
<tr>
<th>Name</th>
<th>Descriptor</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forbidden_zero_bit</td>
<td>f(1)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>nal_ref_idc</td>
<td>u(2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>nal_unit_type</td>
<td>u(5)</td>
<td>6</td>
<td>SEI NAL</td>
</tr>
<tr>
<td>payloadType</td>
<td>u(8)</td>
<td>5</td>
<td>user_data_unregistered</td>
</tr>
<tr>
<td>payloadSize</td>
<td>u(8x)</td>
<td></td>
<td>SEI payload size</td>
</tr>
<tr>
<td>uuid_iso_iec_11578</td>
<td>u(128)</td>
<td></td>
<td>UUID</td>
</tr>
<tr>
<td>user_data_payload_byte</td>
<td>u(n)</td>
<td></td>
<td>Watermarking metadata</td>
</tr>
</tbody>
</table>

uuid_iso_iec_11578: It shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A. Its value is vendor dependent e.g. 0x4062b69958c7442081fa091040882077.

B.3 HEVC METADATA INCLUSION

Metadata is added in the stream as non-VCL NALUs (user data unregistered SEI NAL) as described in Section 7.3.1.1 and Annex D of [4]. As in the case of the AVC codec, these NALUs are interleaved with VCL NALUs, which are carrying video frames data. Watermarking metadata can be seamlessly carried with other video bitstream Non-VCL data.

Table 2 gives the format of the SEI NALU that carries watermarking metadata.

<table>
<thead>
<tr>
<th>Name</th>
<th>Descriptor</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Watermarking metadata SEI NAL for HEVC.
uuid_iso_iec_11578: It shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A. Its value is vendor dependent.

### B.4 AV1 METADATA INCLUSION

The AV1 codec is packetized in so called “Open Bitstream Units” (OBU), which purpose matches the one of NAL units in codecs such as AVC and HEVC. Moreover, the purpose of Reserved OBU (obu_type = 0) is similar to the purpose of SEI and user data. According to the Alliance for Open Media, AV1 codec and Section 5.4 of [7]: “Reserved OBUs do not have a defined syntax. The obu_type reserved values are reserved for future use. Decoders should ignore the entire OBU if they do not understand the obu_type. Ignoring the OBU can be done based on obu_size.” Consequently, watermarking metadata can be added in the stream as the payload of a Reserved OBU.

**Table 3: Watermarking metadata and Reserved OBU for AV1.**

<table>
<thead>
<tr>
<th>Name</th>
<th>Descriptor</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obu_forbidden_bit</td>
<td>f(1)</td>
<td>0</td>
<td>Always zero for all OBUs</td>
</tr>
<tr>
<td>obu_type</td>
<td>f(4)</td>
<td>0</td>
<td>Reserved OBU enumeration</td>
</tr>
<tr>
<td>obu_extension_flag</td>
<td>f(1)</td>
<td>0</td>
<td>No extensions for this instance</td>
</tr>
<tr>
<td>obu_has_size_field</td>
<td>f(1)</td>
<td>1</td>
<td>Read “leb128” encoded size</td>
</tr>
<tr>
<td>obu_reserved_bit1</td>
<td>f(1)</td>
<td>0</td>
<td>Reserved bit set to 0 for all OBUs</td>
</tr>
<tr>
<td>obu_size</td>
<td>leb128</td>
<td>e.g. 0x1e</td>
<td>Watermark metadata size as leb128</td>
</tr>
<tr>
<td>uuid_iso_iec_11578</td>
<td>u(128)</td>
<td></td>
<td>UUID</td>
</tr>
<tr>
<td>metadata</td>
<td>u(n)</td>
<td></td>
<td>Watermarking metadata</td>
</tr>
</tbody>
</table>

AV1 stream must include the obu_size inside the Reserved OBU, and it must be encoded as unsigned integer represented by a variable number of little-endian bytes – for the explanation see Section 4.10.5 of [7]. The obu_size includes both uuid_iso_iec_11578 and metadata.

uuid_iso_iec_11578: It shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A. Its value is vendor dependent.