# DASH-IF CTS 002 V1.0.0 (2023-06)


# DASH-IF Candidate Technical Specification: DASH-IF Forensic A/B Watermarking

## An interoperable watermarking integration schema

***DASH Industry Forum***

3855 SW 153rd Dr.

Beaverton, OR 97003 - USA

Email : admin@dashif.org

***Important notice***

The present document can be downloaded from:
http://www.dashif.org/guidelines

# Contents

# Disclaimer

This document is a candidate Technical Specification. DASH-IF is publishing this specification initially, but the document has been submitted to ETSI and ETSI is asked to publish this document under the Publicly Available Specifications (PAS) agreement. For details on the PAS process, please refer to ETSI PAS Process Guide: https://www.etsi.org/images/files/ETSI_PAS_Process_Guide.pdf

Note that the ETSI specification may be different, addressing editorial updates.

In addition, in case of any identified issues or bugs, please file issues here: https://github.com/Dash-Industry-Forum/Watermarking/issues.

# Intellectual Property Rights

This is a document made available by DASH-IF. The technology embodied in this document may involve the use of intellectual property rights, including patents and patent applications owned or controlled by any of the authors or developers of this document. No patent license, either implied or express, is granted to you by this document. DASH-IF has made no search or investigation for such rights and DASH-IF disclaims any duty to do so. The rights and obligations which apply to DASH-IF documents, as such rights and obligations are set forth and defined in the DASH-IF Bylaws and IPR Policy including, but not limited to, patent and other intellectual property license rights and obligations. A copy of the DASH-IF Bylaws and IPR Policy can be obtained at http://dashif.org/.

The material contained herein is provided on an "AS IS" basis and to the maximum extent pe mitted by applicable law, this material is provided AS IS, and the authors and developers of this material and DASH-IF hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of workmanlike effort, and of lack of negligence.

In addition, this document may include references to documents and/or technologies controlled by third parties. Those third-party documents and technologies may be subject to third party rules and licensing terms. No intellectual property license, either implied or express, to any third-party material is granted to you by this document or DASH-IF. DASH-IF makes no warranty whatsoever for such third-party material.

Note that technologies included in this document and for which no test and conformance material is provided, are only published as candidate technologies, and may be removed if no test material is provided before releasing a new version of this guidelines document. For the availability of test material, please check https://www.dashif.org.

# Foreword

This Technical Specification (TS) has been produced by the DASH-IF Technical Working Group.

# Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in deliverables except when used in direct citation.

# Executive summary

This document describes proposed architecture and API for supporting forensic watermarking for Over-The-Top (OTT) on content that is delivered in an Adaptive Bitrate (ABR) format. To the possible extend, the proposed solutions do not make assumptions on the ABR technology that is being used, it can be for example, DASH or HLS.

While digital watermarking can be used for different use cases, this document will focus on forensic use cases. In this context, it is used to define the origin of content leakage. the watermarking technology modifies media content in a robust and invisible way in order to encode a unique identifier, e.g., a unique session ID. The embedded watermark provides means to identify where the media content, that has been redistributed without authorization, is coming from. In other words, the watermark is used to forensically trace the origin of content leakage.

# 1 Scope

The present document specifies DASH-IF Forensic A/B Watermarking.

# 2 References

## 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ISO/IEC 23009-1:2021 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.

[2] ISO/IEC 13818-1:2019 Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems.

NOTE: Available at https://www.iso.org/standard/75928.html

[3] IETF Internet Draft draft-pantos-hls-rfc8216bis-12, R. Pantos. HTTP Live Streaming 2nd Edition. Internet Draft.

NOTE: Available at https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis-12

[4] IETF RFC 8949, C. Bormann, P. Hoffman, Concise Binary Object Representation (CBOR), December 2020.

NOTE: Available at https://www.rfc-editor.org/info/rfc8949

[5] IETF RFC 8610, H. Birkholz, C. Vigano, C. Bormann, Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures, June 2019.

NOTE: Available at https://www.rfc-editor.org/info/rfc8610

[6] IETF RFC 8392, M. Jones, E. Wahlstroem, S. Erdtman, H. Tschofenig. CBOR Web Token (CWT). May 2018.

NOTE: Available at https://www.rfc-editor.org/info/rfc8392

[7] IETF RFC 4648, S. Josefsson. The Base16, Base32, and Base64 Data Encodings. October 2006.

NOTE: Available at https://www.rfc-editor.org/info/rfc4648

[8] UHD Forum, Watermarking API for Encoder Integration, version 1.0.1, March 2021.

NOTE: Available at https://ultrahdforum.org/guidelines/

[9] IEEE, Std 1003.1 2018 Edition, The Open Group Base Specifications Issue 7, , 31 January 2018.

NOTE: Available at https://pubs.opengroup.org/onlinepubs/9699919799/

[10] DASH-IF registry of watermarking technology vendors IDs.

NOTE: Available at https://dashif.org/identifiers/watermarking/

[11] IETF RFC 9053, J. Schaad, CBOR Object Signing and Encryption (COSE): Initial Algorithms. August 2022.

NOTE: Available at https://www.rfc-editor.org/rfc/rfc9053.html

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] DASH-IF Live Media Ingest Protocol, URL: https://dashif-documents.azurewebsites.net/Ingest/master/DASH-IF-Ingest.html

# 3 Definition of Terms, Symbols and Abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**Client-driven watermarking**: The action of watermarking content when the user device is performing some actions allowing it to make unique requests for content. The user device embeds a watermarking agent that is integrated with the application.

**Client-side watermarking**: The action of watermarking when the user device is the sole responsible for doing the actual watermarking of content. The user device embeds a watermarking agent that is integrated with the audio-visual rendering engine.

**Server-driven watermarking**: The action of watermarking content when the user device is not performing any other operation than conveying information such as tokens, between servers that are responsible for doing the actual watermarking of content that is delivered to the user device.

**Sequencing**: The action of returning a Variant of a segment when it is requested, based on a watermark token. Typically, this action is performed on a CDN edge server and is thus referred to as “edge sequencing”.

**Variant**: An alternative representation of a given segment of a multimedia asset. Typically, a Variant is a pre-watermarked version of the segment.

**Watermark (WM) pattern**: A series of A/B decisions for every segment that is unique per user device.

## 3.2 Symbols

Void.

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| ABR | Adaptive Bit Rate |
| AES | Advanced Encryption Standard |
| AF | Adaptation Field |
| API | Application Programming Interface |
| AVC | Advanced Video Codec |
| CBOR | Concise Binary Object Representation |
| CDDL | Concise Data Definition Language |
| CDN | Content Delivery Network |
| CMAF | Common Media Application Format |
| COSE | CBOR Object Signing and Encryption |
| CPU | Central Processing Unit |
| CWT | CBOR Web Token |
| DAI | Dynamic Ad Insertion |
| DASH | Dynamic Adaptive Streaming over HTTP |
| DRM | Digital Rights Management |

| | |
|---|---|
| ECDH | Elliptic Curve Diffie-Hellman |
| HEVC | High Efficiency Video Coding |
| HLS | HTTP Live Streaming |
| HMAC | keyed-Hashing for Message AuthentiCation |
| HTTP | Hypertext Transfer Protocol |
| IANA | Internet Assigned Numbers Authority |
| IOP | InterOPerability |
| IP | Internet Protocol |
| ISOBMFF | ISO Base Media File Format |
| JITP | Just In Time Packager |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Token |
| MPD | Media Presentation Description |
| NAL | Network Abstraction Layer |
| OTT | Over The Top |
| RIST | Reliable Internet Stream Transport |
| RTMP | Real-Time Messaging Protocol |
| RTP | Real Time Protocol |
| SEI | Supplemental Enhancement Information |
| SRT | Secure Reliable Transport |
| TS | Transport Stream |
| TV | Television |
| UDP | User Datagram Protocol |
| UHD | Ultra-High Definition |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| UUID | Universally Unique Identifier |
| VOD | Video On Demand |
| WM | Watermark |
| WMID | Watermark Identifier |
| WMT | Watermark Token |

# 4 OTT Watermarking Using Variants

The objective of forensic watermarking is to deliver a unique version of a media asset to the different users consuming the asset. This is somewhat in opposition with media delivery mechanisms that aim at delivering the same asset to all users for efficiency purposes. As a result, in the broadcast era, a typical approach was to perform the watermarking operation at the very last step of the media delivery pipeline, within the end user device e.g., a set-top box. This solution has the virtue of leaving the whole media delivery pipeline unaltered but raises security and interoperability challenges when a large variety of devices owned and operated by the end user shall be supported. This is for instance the case with over-the-top (OTT) media delivery where content is consumed on mobile phones, tablets, laptops, connected TVs, etc. As a result, new forensic watermarking solutions have gained momentum that do not perform security-sensitive and complex operations in the end user realm. While such approaches require minimal changes in the end-user devices, they do mandate the media delivery pipeline to be modified accordingly.

A notable example of such network-side watermarking solutions is OTT watermarking using Variants for adaptive bitrate (ABR) content. In this case, the content is delivered by segments. The baseline idea is then to generate pre-watermarked Variants of each segment and to modify the delivery protocol so that each end user receives a unique sequence of Variants depending on a watermark pattern that has been assigned to the end user. The semantic of this pattern is context dependent and can be, for instance, a device identifier, an account identifier, a session identifier, etc. Figure 1 illustrates a particular case of this strategy, coined as A/B watermarking, where there are two Variants generated for each segment, each Variant containing a watermark that either encodes the information ‘0’ or ‘1’. As a result, the watermarking system will require the transmission of a sequence of Variants as long as the length of the pattern to successfully recover the whole unique identifier.

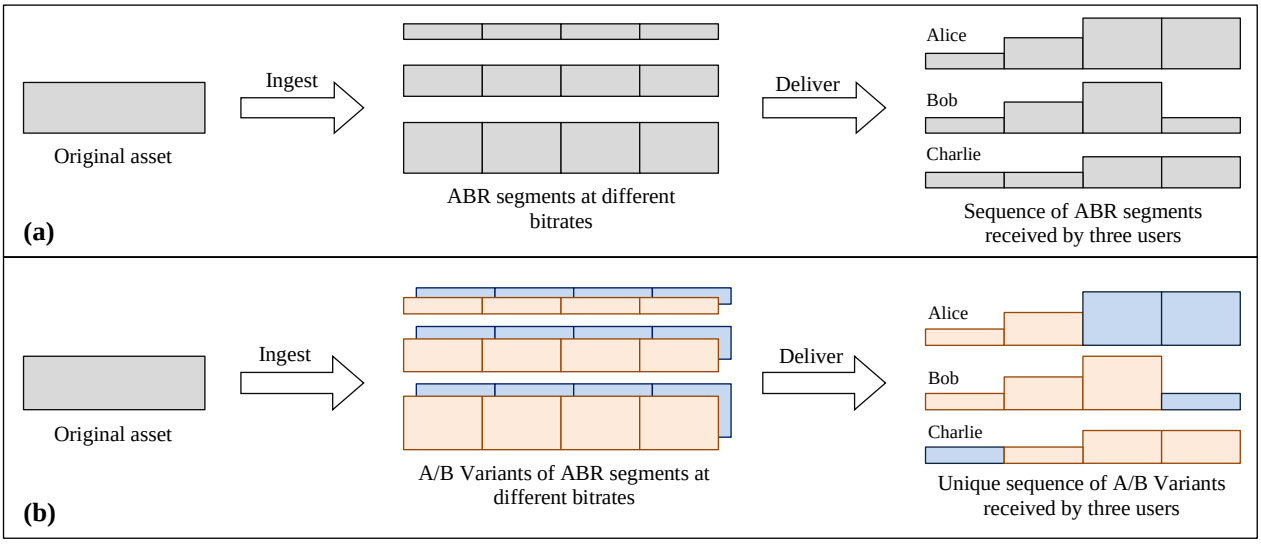


**Figure 1: A/B watermarking concept with (a) ABR content delivery and (b) A/B Variants delivery.**

When using Variants, the serialization process essentially boils down to delivering a unique sequence of Variants to each individual end user. There are two main families of methods to achieve this:

1. Server-driven methods, wherein the client does perform no operation related to watermarking. It simply fetches and forwards a token to the CDN that is responsible for delivering a unique sequence of Variants.
2. Client-driven methods, wherein the client is responsible for the serialization operation. For instance, it relies on some session-based digital object to tamper the URI ABR segments and thereby directly query a unique sequence of Variants from the CDN.

This document is describing the server-driven methods. Client-driven methods are not part of this document.

# 5 Server-Driven Architecture and Workflows

## 5.1 Introduction

In the server-driven architecture, the device is unaware that content it consumes is watermarked. The device only exchanges a token with servers allowing these servers, usually CDN edges, to make the decision on which A or B Variant it delivers to the device. In this document, an end-to-end system is presented. It includes the definition of watermarking metadata that limits the need for naming conventions by allowing the encoder to send this metadata all the way to the edge through origins to enable the sequencing of bits. The following goes through the functional architecture and describes the workflows when preparing content and when consuming content.

In the following, it is assumed that the edge is a CDN edge. There are optional architectures, but this does impact the overall functional architecture and workflows. It is also assumed that multi-track content (audio and video multiplexed in one segment) is out of the scope of this document. In addition, all the workflows are only examples of possible implementations.

## 5.2 Functional Architecture

Figure 2 shows the simplified high-level functional architecture and the different interaction between the components that are involved in the flows when a device consumes watermarked content. Note that this also shows that content is encrypted, as watermarking will likely be added for premium content that is also encrypted and protected by a DRM system.

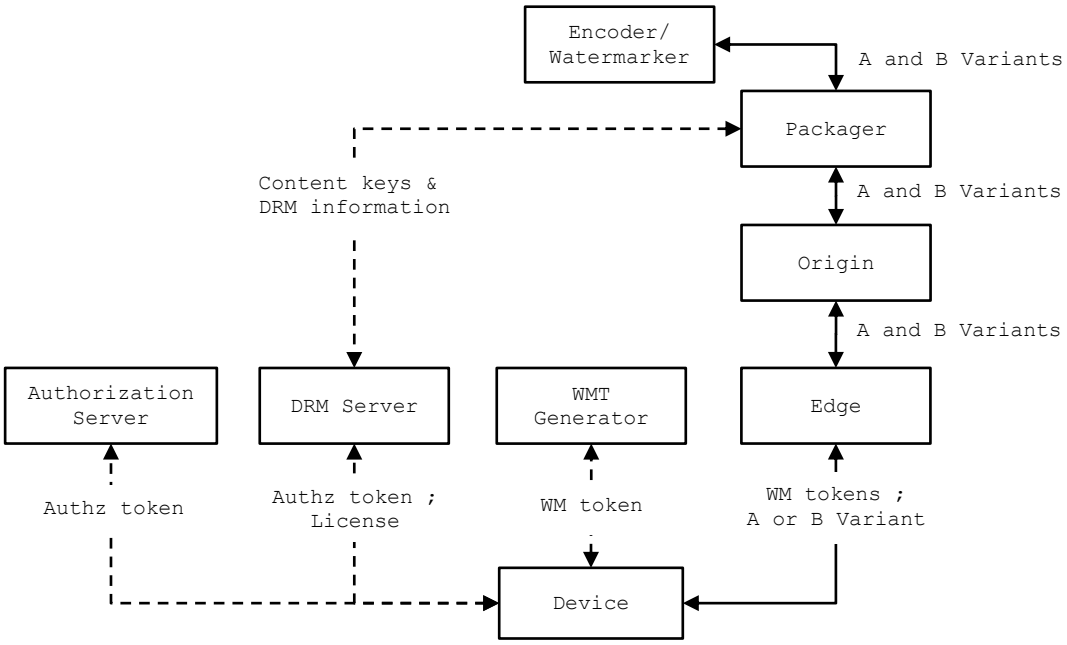


**Figure 2: Functional architecture.**

To consume content, a device needs, at minimum, to have an authorization token (for getting a DRM license) and a WM token that contains a WM pattern, a series of A or B decisions. The device is responsible for obtaining the required data before requesting segments to the CDN.

## 5.3 System Configuration

Enabling or disabling the edge sequencing logic is set through the configuration to the edge. As an example, this can be useful for a service of live sporting events where only premium events require watermarking enforcement. Other moments of the day do not require it. In this case, content is still watermarked but the edge is only configured to sequence during the limited period of time of the premium event. When sequencing is disabled, the edge shall consume segments on the endpoint for Variant A. If this endpoint is not working properly, the origin shall deliver any available Variant on this endpoint.

NOTE: When enabling watermarking, all devices that do not have a WM token will receive an error when requesting content, hence they are then forced to request such token.

NOTE: As an example, enabling and disabling sequencing can be done with an API enable (true/false).

Watermarked objects names shall include a pattern that the CDN can match to differentiate these objects from non-watermarked objects (initialization segments, subtitles, trickplay images). As an example, for a DASH manifest located at https://edge.hostname/path/to/endpoint/index.mpd that references video segments as

```
<SegmentTemplate timescale="60000" media="video_segment_$RepresentationID$_$Time$.mp4"
initialization="video_init_$RepresentationID$.mp4" startNumber="10967120"
presentationTimeOffset="903486496960">
```

the pattern for the differentiation of these objects from non-watermarked objects is **`video_segment_`**.

One of the following identification schemes, referred as `variantId` in this document, shall be used for the identification of the Variants:

- A lower-case letter beginning with ‘a’. Variants are then ‘a’, ‘b’ and so on.
- A number beginning with 0. Variants are then 0, 1 and so on.

When addressing content, `variantId` shall be translated into `variantPath` as follows:

- `variantPath` = `${variantId}` followed by ‘/’ with the exception, that if `${variantId}` is ‘a’ or ‘0’ then `${variantPath}` may be empty

## 5.4 WM Token

A WM token provides a WM pattern which is unique (for example per streaming session or per user). This pattern allows the sequencing of A/B Variants.

Two tokenisation schemes are defined in this document. The first, named direct, embeds the WM pattern in the token and can be opened and interpreted by an edge irrespective of the underlying WM technology and provider. The second, named indirect, requires integration of a WM technology provider's edge sequencing software at the edge.

The following are requirements on the WM token:

- The token shall be a CWT token, the basic structural requirements are defined in [6].
- The token shall be with integer keys in “deterministically encoded CBOR” as specified in [4] clause 4.2.
- Recipients shall process claims listed in [6] clause 3.1 when they are present. `exp` and `iat` shall be present.
- The token shall include either a WM pattern (direct mode) or data for deriving the WM pattern (indirect mode). Absence of a `wmpattern` claim implies that the token is in indirect mode.
- Recipients shall support direct mode and may support indirect mode.
- The token shall be signed as described in clause 7 of [6]. Recipients shall support the HMAC 256/256 (kty number 5) and ES256 (kty number -7) algorithms.
- The token shall be base64url-encoded as described in clause 5 of [7].

The following claims are defined and Table 1 provides the integer claim keys.

```
wmtoken = {
  wmver-label ^ => wmver-value,
  wmvnd-label ^ => wmvnd-value,
  wmpatlen-label ^ => wmpatlen-value,
  ? wmsegduration-label ^ => wmsegduration-value,
  wmtoken-direct // wmtoken-indirect,
  * wmext-label => any
}

wmver-value = uint .size 1
wmvnd-value = uint .size 1
wmpatlen-value = uint .size 2
wmsegduration-value = [(wmtimeticks : uint, wmtimescale : uint)]
wmext-label = int

; direct mode
wmtoken-direct = {
  wmpattern-label ^ => wmpattern-value
}
wmpattern-value = COSE_Encrypt0 // COSE_Encrypt // bytes

; indirect mode
wmtoken-indirect = {
  wmid-label ^ => wmid-value
  wmopid-label ^ => wmopid-value
  wmkeyver-label ^ => wmkeyver-value
}
wmid-value = text
wmopid-value = uint
wmkeyver-value = uint
```

**Table 1: Integer Claim key values for the WM token.**

| Claim label | Integer key |
|---|---|
| wmver-label | 300 |
| wmvnd-label | 301 |
| wmpatlen-label | 302 |
| wmsegduration-label | 303 |
| wmpattern-label | 304 |
| wmid-label | 305 |
| wmopid-label | 306 |
| wmkeyver-label | 307 |

`wmver`

The version of the WM Token. Recipients shall support this claim. This document describes version 1.

`wmvnd`

The WM technology vendor. Recipients shall support this claim. This provides the context for the key material needed for signature verification. In the direct mode, it also provides the context for the key material needed for

decrypting wmpattern if needed. In the indirect mode, it identifies the vendor specific core to use. A list of WM technology vendor identifiers is available at [10].

wmpatlen
The length in bits of the WM pattern. Recipients shall support this claim.

wmpattern
The WM pattern. Recipients shall support this claim in direct mode. It is recommended to encrypt the pattern. Recipients shall support ECDH-SS+A128KW (key type -32) as defined in [11].

wmsegduration
The nominal duration of a segment. This claim is optional. Recipients may support this claim. When WMPaceInfo data is not available, this may allow the edge to define the index to be considered in the WM pattern. If WMPaceInfo is available, this claim shall be ignored. The array contains exactly 2 values. The first value is a duration in time ticks where its base unit is defined by the second value. The second value is the scale in number of time ticks per second. As an example, [60'000, 10'000] means that the segments are 60'000 ticks long while the scale is 10'000 ticks per second, wmsegduration is then equal to 6 seconds.

wmid
Used as input to derive the WM pattern for indirect mode. Recipients shall support this claim in indirect mode. The derivation algorithm is not defined in this document and is vendor specific.

wmopid
Used as additional input to derive the WM pattern for indirect mode. Recipients shall support this claim in indirect mode.

wmkeyver
The key to use for derivation of the WM pattern in indirect mode. Recipients shall support this claim in indirect mode.

Once the WM pattern is obtained from the token (either directly, decrypted or calculated), the CDN edge shall enforce big-endian convention to address a single bit in it when using the value of position (defined in clause 5.5.2).

The following is an example with a WM pattern equal to 0x0A0B0C0D.

| Byte | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| bit offset | 01234567 | 01234567 | 01234567 | 01234567 |
| binary | 00001010 | 00001011 | 00001100 | 00001101 |
| hex | 0A | 0B | 0C | 0D |

For a value of position equal to 3, the bit to consider is highlighted in green (equal to 0). This is not any other bit, especially, those highlighted in red.

For the indirect mode, there is a vendor specific core (identified by wmvnd). It is recommended that, performance-wise and software-stack-wise, it is comparable with the direct case. In other words, the vendors specific core should be based on the crypto operations which are used in the direct mode, and its performance should be equivalent. For example, the direct mode relies on one decryption operation when wmpattern is encrypted, the vendor specific core should be consisting of the similar operations to preserve the quantity of operations comparable between these two modes.

# 5.5 WMPaceInfo

## 5.5.1 Introduction

When a device requests a segment, the edge sequencing logic needs to know which bit in the unique WM pattern to consider for retrieving either A or B Variant of the requested segment before delivering it to the device. WMPaceInfo contains this mapping in addition to some data needed for content preparation. It is transmitted from the encoder (that is combined with the watermarking pre-processor) to the following servers that may need it (packager, origin, or edge).

## 5.5.2 WMPaceInfo Data

WMPaceInfo is as shown in Table 2,

**Table 2: WMPaceInfo data.**

| Attribute | Producer | Consumers | Purpose |
|---|---|---|---|
| variant | Encoder | Edge | Integration, debugging |
| position | Encoder | Edge | Bit position in the WM pattern |

| | | | |
|---|---|---|---|
| `firstpart` | Encoder | Packager, Origin | Egress packaging |
| `lastpart` | Encoder | Packager, Origin | Egress packaging |

where

- `variant` gives the Variant identification, 0, 1 and so on. This information can be useful up to the edge for verifying that the right Variant has been obtained.
- `position` is the index in the WM pattern to consider for this segment. Positions are zero-based. When it is equal to -1, the corresponding segment is not watermarked. For example, `position`=33 indicates that this segment refers to position 34 of the WM pattern.
- `firstpart` informs whether this segment is the first one with this `position` value. It is equal to true if this is the case, otherwise it is equal to false. See clause 5.6.2 for further details.
- `lastpart` informs whether this segment is the last one with this `position` value. It is equal to true if this is the case, otherwise it is equal to false. See clause 5.6.2 for further details.

## 5.5.3 Conveying WMPaceInfo

### 5.5.3.1 Introduction

`WMPaceInfo` is delivered from the encoder to other servers. There is no unique mechanism for this. This document does not recommend one preferred option applicable for all protocols, Table 3 only present some possible options for conveying `WMPaceInfo` with a preferred option for some protocols (in bold in the table). The following goes through these different options.

**Table 3: Possible options for conveying WMPaceInfo information.**

| Ingest protocol | WMPaceInfo delivery options |
|---|---|
| RTMP | SEI |
| RTP/UDP/RIST/SRT | SEI, TS adaptation field |
| HLS/TS over HTTP POST | HTTP header, SEI |
| CMAF-based protocols/formats (HLS/fMP4, DASH) over HTTP POST | **HTTP header**, ISOBMFF box, SEI |
| File access protocol | ISOBMFF box, SEI, **sidecar file** |

### 5.5.3.2 Sidecar File

When segments (discrete files or byteranges) are delivered with a file transfer protocol, it may be convenient to have `WMPaceInfo` data in a sidecar file. For efficiency, the `WMPaceInfo` data is not copied directly as some would be included multiple times.

The sidecar file is of the following format (using CDDL representation [5]), following recommendation of clause 5 of [4] and shall be encoded using deterministically encoded CBOR as specified in [4] clause 4.2 with integer keys.

```
;----------------------------------------+
; Maps Integer Keys
version      = 1
segments     = 2
fileSize     = 3
startRange   = 4
segmentRegex = 5
position     = 6
firstpart    = 7
lastpart     = 8
;----------------------------------------+

discrete-segment = {
  ?segmentRegex : text,
  position :      int .size 2 .ge -1,
  ?firstpart :    bool,
  ?lastpart :     bool
}

byterange-segment = {
  startRange : uint .size 8,
  position :   int .size 2 .ge -1
}
```

```
sidecar-discrete = {
  version :  uint .size 1,
  segments : [+ discrete-segment]
}

sidecar-byterange = {
  version :  uint .size 1,
  fileSize : uint .size 8,
  segments : [+ byterange-segment]
}

sidecar = (sidecar-byterange // sidecar-discrete)
```

When segments are discrete files:

- `sidecar` shall contain only `sidecar-discrete` elements.
- `version` is set to 1 for sidecar files compliant to this document.
- `segmentRegex` is a POSIX extended regular expression as described in clause 9 of [9]. It allows to define the filename of the segments for which the data applies. `segmentRegex` is optional.
- `position`, `firstpart` and `lastpart` are defined in clause 5.5.2. `firstpart` and `lastpart` are optional.

NOTE: Using regular expressions and file naming conventions allows reducing the number of required side car files. The same side car file could be used for all renditions for example. This allows the origin to reduce the number of sidecar files, but the edge will always receive several copies of the same data as caching is done on the exact filename. It is recommended to balance the advantages and disadvantages of regular expressions, because of its Central Processing Unit (CPU) load on the origin.

The following is an example for a set of segments where the filenames satisfy the `segmentRegex` expression. In this example, the filenames are in the form of `video_segment_[repID]_123.mp4`, `video_segment_[repID]_124.mp4` and so on, allowing to have one sidecar file for all Representations (for DASH).

```
sidecar (
  /version/ 1,
  /segments/ [{/segmentRegex/ "video_segment_ .*?_123.mp4", /position/ 21},
              {/segmentRegex/ "video_segment_ .*?_124.mp4", /position/ 22}]
)
```

When segments are byteranges:

- `sidecar` shall contain only `sidecar-byterange` elements.
- `version` is set to 1 for sidecar files compliant to this document.
- `fileSize` is the size of the track in bytes.
- `startRange` defines the position of the first byte in the byterange. This expressed as a byte offset from the beginning of the track `sidecar-byterange` elements in the array shall be ordered in increasing `startRange` values.
- `position` is defined in clause 5.5.2.

NOTE: The first byterange of a track contains the initialisation segment. When segments are delivered with byteranges, it is not possible to differentiate the request for this part of the file from a request for a media segment when using a pattern as described in clause 5.3.5. The initialisation segment is not watermarked, therefore `position` equal -1 for this segment.

The following is an example of a file with an initialisation segment part of the byterange from 0 to 1117 and two segments.

```
Sidecar (
  /version/ 1,
  /fileSize/ 262445216,
  /segments/ [{/startRange/ 0, /position/ -1},
              {/startRange/ 1118, /position/ 0},
              {/startRange/ 1701212, /position/ 1},
              …
              {/startRange/ 261083393, /position/ 118},
```

```
                {/startRange/ 262073936, /position/ 119}]
)
```

#### 5.5.3.3 HTTP Header

When content is pushed, in the request header, under the `WMPaceInfoIngest` HTTP header field, the following JSON object is added:

```
WMPaceInfoIngest : {
  "version":   version,
  "variant":   variant,
  "position":  position,
  "firstpart": firstpart,
  "lastpart":  lastpart
}
```

where

`version` is set to 1 for `WMPaceInfoIngest` compliant to this document.

`variant`, `position`, `firstpart` and `lastpart` are defined in clause 5.5.2.

When content is pulled, in the response header, under the `WMPaceInfoEgress` HTTP header field, the following CBOR object, base64url-encoded as described in clause 5 of [7], is added:

```
WMPaceInfoEgress : <sidecar-discrete>
```

where

- `sidecar-discrete` is defined in clause 5.5.3.2 and contains exactly one `discrete-segment` object with data for that segment.

Below is an example of the JSON element added in a `WMPaceInfoIngest` header field where the payload of the HTTP request contains the full segment of Variant A.

```
{
  "version": 1,
  "variant": 0,
  "position": 33,
  "firstpart": true,
  "lastpart": true
}
```

#### 5.5.3.4 ISOBMFF Box

The format of `WMPaceInfo` class shall be

```
class WMPaceInfo {
  unsigned int(8)  version;
  unsigned int(8)  variant;
  unsigned int(1)  emulation_1;
  unsigned int(15) position;
  unsigned int(1)  emulation_2;
  unsigned int(1)  firstpart;
  unsigned int(1)  lastpart;
  unsigned int(5)  reserved;
}
```

Where

- `version` is set to 1 for `WMPaceInfo` compliant to this document.
- `variant`, `position`, `firstpart` and `lastpart` are defined in clause 5.5.2.
- `emulation_1`, and `emulation_2` are set to 1.

Within an ISOBMFF file, the `WMPaceInfo` class shall be carried in the following box:

Box Type: `'wmpi'`
Container: Top level box
Mandatory: No
Quantity: Zero or one

```
aligned(8) class WMPaceInfoBox extends Box('wmpi')
{
  WMPaceInfo();
```

```
}
```

This box should be inserted only at the beginning of a segment, after the `styp` box and before the `moof` box, in order to facilitate content manipulation when padding it (see clause 5.7.5.1).

#### 5.5.3.5 SEI Message

SEI messages are inserted in the stream with a specific syntax depending on the codec. [8] provides the syntax for AVC, HEVC and AV1 video codecs in Annex B. In these messages:

- The UUID shall be equal to `0xbec4f824-170d-47cf-a826-ce008083e355`
- The watermarking metadata is the `WMPaceInfo` data with the format defined for the class `WMPaceInfo()` in clause 5.5.3.4.

This message should be inserted for the first frame of a segment to facilitate content manipulation when padding it (see clause 5.7.5.1).

#### 5.5.3.6 TS Adaptation Field

Following clause U of [2], the format of the private adaptation field descriptor carrying the `WMPaceInfo` data is defined in Table 4.

**Table 4: WMPaceInfo descriptor.**

| Syntax | No. of bits | Mnemonic |
|---|---|---|
| `temi_WMPaceInfo_descriptor {` | | |
| `  af_descr_tag` | `8` | `uimsbf` |
| `  af_descr_length` | `8` | `uimsbf` |
| `  WMPaceInfo()` | `40` | `uimsbf` |
| `}` | | |

where

- `af_descr_tag` is an 8-bit field that identifies this AF descriptor. It is equal to 0xDF.
- `af_descr_length` is an 8-bit field specifying the number of bytes of the AF descriptor immediately following `af_descr_length` field.
- `WMPaceInfo()` is a 40-bit field that carries the information defined for the class `WMPaceInfo()` in clause 5.5.3.4.

This message should be inserted for the first frame of a segment to facilitate content manipulation when padding it (see clause 5.7.5.1).

## 5.6 Content Preparation

### 5.6.1 Introduction

Content preparation means the generation of A/B Variants of the segments followed by the push of content on the origin. It is under a workflow manager responsibility in case of VOD and fully automated for Live content. The encoder generates the different Variants of the adaptive content. The encrypted segments, the DASH manifest and HLS playlists are generated by the packager and pushed to the origin. A simplified flow is shown in Figure 3 for the case of Live content if the DASH-IF ingest protocol is used [i.1] (note that content protection steps are omitted for clarity). For encrypted content, Variants of every segment part of the same Representation may be encrypted using the same encryption method and with the same content key, meaning the same DRM license allows decrypting the A and B Variants. In addition to the Variants, the encoder also pushes `WMPaceInfo` that contain information allowing the packager and the origin to properly associate the pieces of Variants that are pushed to a bit position on the WM pattern. In such flow, the packager can aggregate multiple ingest segments into one egress segment, with the limitation that only ingest segments carrying the same `position` value can be aggregated together.

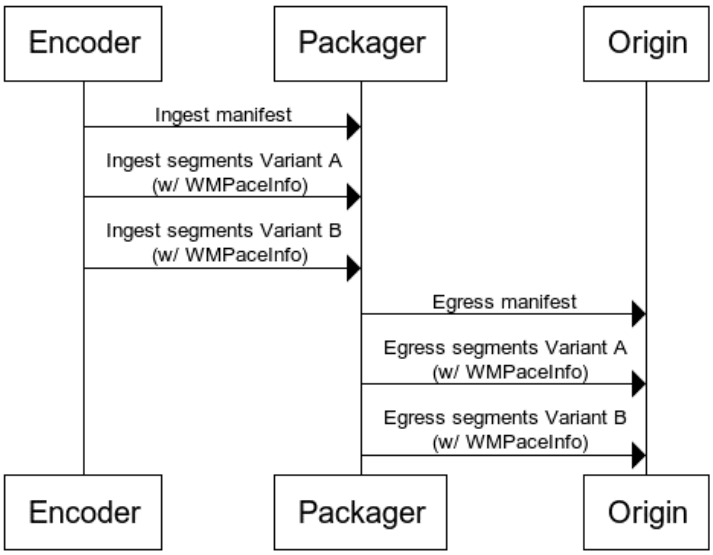


**Figure 3: Example of Live DASH content preparation workflow using the DASH-IF ingest protocol.**

### 5.6.2 Encoding Recommendations

This clause contains recommendation when encoding content. The goal is to facilitate the creation and management of A and B Variants in the delivery chain.

When segments are requested as byteranges in a file or when chunks are requested as byteranges in a segment, the segments and chunks in A and B Variants shall have the same size as the player receives only one DASH manifest or HLS playlist and will get byterange lengths from one `sidx` box only. How this is achieved in out of the scope of this document (as an example, bit stuffing in the encoder is an option).

NOTE: This solution does not allow creating aligned segment when content is delivered with HLS in the form of MPEG-2 TS segments encrypted with AES sample encryption, because start code emulation prevention is re-applied over the entire NAL unit after encryption with MPEG-2 TS.

NOTE: An alternative solution is either to not use segments requested as byteranges, but to use discrete files (in these cases, there is no need to align Variant A and B of the same segment) or use CMAF segments with HLS where start code emulation prevention is not re-applied after encryption.

### 5.6.3 Delivering Content and WMPaceInfo from the Encoder to the Packager

Only one option for conveying `WMPaceInfo` information from the encoder to the origin shall be used. Multiple concurrent formats are not allowed.

NOTE: When `WMPaceInfo` is delivered in TS adaptation field, ISOBMFF box, or SEI, it adds overhead in the delivery from the CDN to devices. The sidecar file and HTTP header delivery methods do not.

The encoder is sending part of segments to the packager, as the output of the encoder is not necessarily aligned on the segment length. Furthermore, when multiple streaming formats are used, it may happen that segments generated by the packager are not of the same size for every streaming protocol (for example, 2 seconds segments for DASH and 4 seconds segments for HLS). The encoder then needs a mechanism for announcing which parts of the Variants it sends can be aggregated in segments. This is achieved by using the `firstpart` and `lastpart` within `WMPaceInfo`.

NOTE: Where an encoder delivers additional metadata to instruct the packager how to aggregate the content into segments, the encoder ensures that metadata and `firstpart` and `lastpart` fields are consistent.

For example, the encoder could output the series of content elements of 1 second length with `WMPaceInfo` as shown in Figure 4.

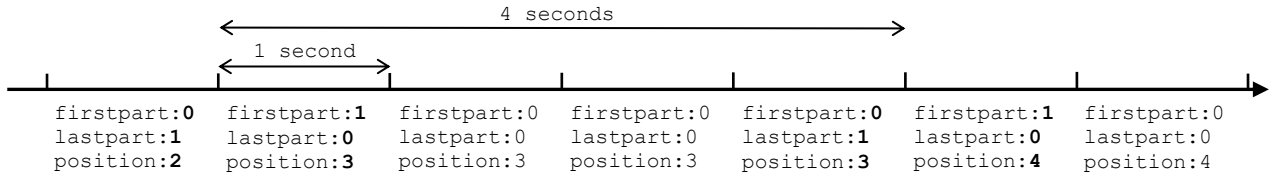


**Figure 4: Example of output of an encoder.**

If the encoder pushes over HTTP these elements, each one should carry a `WMPaceInfoIngest` HTTP header with the relevant data. Every server keeps the information within the header associated to the ingested segment. In some cases, for example when the origin does additional packaging, the header may be updated. The packager can then prepare segments according to the streaming protocol. From the example above, it can create segments of 2 or 4 seconds keeping the consistency of the watermarking.

NOTE: In this case, 2 consecutive segments of 2 seconds carry the same `position` value, hence a larger piece of content is required to retrieve an identifier compared to the case where 2 consecutives segments carrying different `position` values.

Other options are to carry `WMPaceInfo` in a sidecar file or SEI or ISOBMFF box or TS adaptation field. For cases where the origin can perform additional manipulation of the content, `WMPaceInfo` may be carried within the content instead providing it is overwritten as specified in clause 5.6.5.

## 5.6.4 Segment Ingress Path Structure on the Origin

### 5.6.4.1 Introduction

The DASH manifest [1] and HLS playlist [3] served to the devices are “neutral”, meaning that

- The same playlist or manifest is served to all devices of all end-users.
- It does not expose different names for A and B Variants of a given segment.

### 5.6.4.2 Locating the Variants

Egress DASH manifests and HLS playlists shall be neutral, but ingest DASH manifests and HLS playlists include information about the A and B Variants being ingested, this is

- The ingest path
- Some signalling elements to describe if a DASH Adaptation Set includes the A or B Variants, or if an HLS media playlist includes A or B Variants.

The ingest of A and B Variants shall use specific ingest paths that include a Variant identification (`${variantId}`).

DASH Ingest manifests shall include an **`AdaptationSet`** per Variant. The contents of the **`AdaptationSet`** shall be identical for every Variant apart from an **`EssentialProperty`** element that indicates the `variantId` and that the Variants are grouped (i.e., they reference the same media). It has the `@schemeIdUri` attribute equal to `http://dashif.org/guidelines/watermarking_variant#${variantId}` where `${variantId}` identifies the Variant with which this **`EssentialProperty`** element is associated and `@value` attribute identifies the group to which the Variant belongs. If there are additional Variants (A, B and C for example), the `@schemeIdUri` attribute is different for each Variant, for example, for Variant C, `@schemeIdUri` attribute shall be equal to `http://dashif.org/guidelines/watermarking_variant#c`, if the schema with lower case letters is used.

The following is an example of a DASH ingest manifest with two Variants, A and B. The watermarking signalling is highlighted in bold. **`EssentialProperty`** elements indicate that Variant A and Variant B belong to the same group ("tv1"). In this example, lower case letters are used for `variantId`.

NOTE: Segment file naming with template based on segment `$number` or `$time` are possible.

```
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1" bitstreamSwitching="true">
  <EssentialProperty schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a"
value="tv1"/>
  <SegmentTemplate timescale="60000"
media="a/video_segment_$RepresentationID$_$Time$.mp4"
initialization="a/video_init_$RepresentationID$.mp4" startNumber="10967120"
presentationTimeOffset="903486496960">
```

```
    <SegmentTimeline>
      <S t="903487696960" d="240000"/>
      <S t="903487936960" d="186000"/>
    </SegmentTimeline>
  </SegmentTemplate>
  <Representation id="27" width="1920" height="1080" frameRate="30/1"
bandwidth="5000000" codecs="avc1.4D4028"/>
  <Representation id="24" width="1280" height="720" frameRate="30/1"
bandwidth="3000000" codecs="avc1.4D401F"/>
  <Representation id="26" width="640" height="360" frameRate="30/1"
bandwidth="1499968" codecs="avc1.4D401E"/>
</AdaptationSet>
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1" bitstreamSwitching="true">
  <EssentialProperty schemeIdUri="http://dashif.org/guidelines/watermarking_variant#b"
value="tv1"/>
  <SegmentTemplate timescale="60000"
media="b/video_segment_$RepresentationID$_$Time$.mp4"
initialization="b/video_init_$RepresentationID$.mp4" startNumber="10967120"
presentationTimeOffset="903486496960">
    <SegmentTimeline>
      <S t="903487696960" d="240000"/>
      <S t="903487936960" d="186000"/>
    </SegmentTimeline>
  </SegmentTemplate>
  <Representation id="27" width="1920" height="1080" frameRate="30/1"
bandwidth="5000000" codecs="avc1.4D4028"/>
  <Representation id="24" width="1280" height="720" frameRate="30/1"
bandwidth="3000000" codecs="avc1.4D401F"/>
  <Representation id="26" width="640" height="360" frameRate="30/1"
bandwidth="1499968" codecs="avc1.4D401E"/>
</AdaptationSet>
```

For HLS ingest playlists, the multivariant playlist shall include all the A and B Variants with a custom attribute specifying the Variant (using `${variantId}` identification as defined in clause 5.3). The attribute is `WATERMARKING-VARIANT`. A combination of both audio and video watermarking can therefore be used in a single streamset. In the media playlists, the only specific signalling is the segments paths that reflects on which ingest path the Variants are ingested. The sub-paths in the media playlists shall use the same convention that the `${variantId}`.

The following is an example of HLS ingest playlists, the watermarking signalling is highlighted in bold (this theoretical example, both the video and audio are watermarked). In this example, lower case letters are used for `variantId`.

**Multivariant playlist**

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-STREAM-INF:BANDWIDTH=5227200,AVERAGE-
BANDWIDTH=3511200,CODECS="avc1.4d401f,mp4a.40.2",RESOLUTION=1280x720,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2719200,AVERAGE-
BANDWIDTH=1861200,CODECS="avc1.77.30,mp4a.40.2",RESOLUTION=640x360,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=8571200,AVERAGE-
BANDWIDTH=5711200,CODECS="avc1.4d4028,mp4a.40.2",RESOLUTION=1920x1080,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_3.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=5227200,AVERAGE-
BANDWIDTH=3511200,CODECS="avc1.4d401f,mp4a.40.2",RESOLUTION=1280x720,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_4.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2719200,AVERAGE-
BANDWIDTH=1861200,CODECS="avc1.77.30,mp4a.40.2",RESOLUTION=640x360,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_5.m3u8
```

```
#EXT-X-STREAM-INF:BANDWIDTH=8571200,AVERAGE-
BANDWIDTH=5711200,CODECS="avc1.4d4028,mp4a.40.2",RESOLUTION=1920x1080,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_6.m3u8
#EXT-X-IMAGE-STREAM-INF:BANDWIDTH=55649,AVERAGE-
BANDWIDTH=23579,RESOLUTION=308x174,CODECS="jpeg",URI="trickplay_7.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,LANGUAGE="eng",NAME="Stadium
ambiance",AUTOSELECT=YES,DEFAULT=YES,GROUP-
ID="program_audio",URI="audio_8.m3u8",WATERMARKING-VARIANT="a"
#EXT-X-MEDIA:TYPE=AUDIO,LANGUAGE="eng",NAME="Stadium
ambiance",AUTOSELECT=YES,DEFAULT=YES,GROUP-
ID="program_audio",URI="audio_9.m3u8",WATERMARKING-VARIANT="b"
```

NOTE: While it is a legal signalling in HLS to have multiple `EXT-X-MEDIA` tags with the same `GROUP_ID` value, each tag shall have a different `NAME` value. As these playlists are not for devices to consume and to minimize the processing on the playlists, the ingest playlists do not follow this rule and multiple `EXT-X-MEDIA` share the same `NAME` value.

**Media playlist (A Variant)**

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:11352692
#EXT-X-MAP:URI="video_init_1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2021-09-15T00:48:38.933Z
#EXTINF:6.000,
a/video_segment_1_11352692.mp4
#EXTINF:6.000,
a/video_segment_1_11352693.mp4
#EXTINF:6.000,
a/video_segment_1_11352694.mp4
#EXTINF:6.000,
a/video_segment_1_11352695.mp4
#EXTINF:6.000,
a/video_segment_1_11352696.mp4
```

**Media playlist (B Variant)**

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:11352692
#EXT-X-MAP:URI="video_init_1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2021-09-15T00:48:38.933Z
#EXTINF:6.000,
b/video_segment_1_11352692.mp4
#EXTINF:6.000,
b/video_segment_1_11352693.mp4
#EXTINF:6.000,
b/video_segment_1_11352694.mp4
#EXTINF:6.000,
b/video_segment_1_11352695.mp4
#EXTINF:6.000,
b/video_segment_1_11352696.mp4
```

When the ingested content is not watermarked anymore, then

- For DASH content, the **`EssentialProperty`** elements shall be removed from the ingest manifest and a new **`Period`** shall be created with a single **`AdaptationSet`**. The path to the segments shall be updated, removing any information on the Variant location (in the example above, the `a/` shall be removed from the `@media` value of the **`SegmentTemplate`** element).
- For HLS content, the encoder shall create a new multivariant playlist that does not include `WATERMARKING-VARIANT` attributes. It also stops delivering the additional media playlists for the B Variant and others if present. The path to the segments in the media playlist delivered to devices shall be updated, removing any information on the Variant location (in the example above, the `a/` shall be removed from the media playlist).

NOTE: Stopping watermarking content is different from toggling edge sequencing logic (see clause 5.3).

#### 5.6.4.3 Locating the Sidecar File

The sidecar file is part of the ingest with the DASH manifest or HLS playlist, the link to this file is added in different places depending on the format.

DASH ingest manifests shall include an **`EssentialProperty`** element at the **`Representation`** level with a `@schemeIdUri` attribute equal to `http://dashif.org/guidelines/watermarking_wmpaceinfo` and `@value` attribute equal to the pointer to the sidecar file. The pointer is relative to the ingest manifest.

The following is an example of a DASH ingest manifest where the watermarking signalling is highlighted in bold. In this example, the absolute path for the sidecar file for the first representation is equal to https://dash.edgesuite.net/dash264/TestCases/1a/ElephantsDream_H264BPL30_0100.264.dash_wm_pace_info.

NOTE: This example also includes the signalling defined in clause 5.6.2 (for one Variant A). In this case, the **`EssentialProperty`** elements are added in the **`Representation`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 DASH-MPD.xsd"
  type="static"
  mediaPresentationDuration="PT654S"
  minBufferTime="PT4S"
  …
  <AdaptationSet mimeType="video/mp4" codecs="avc1.42401E" subsegmentAlignment="true"
subsegmentStartsWithSAP="1" contentType='video' maxWidth="480" maxHeight="360"
maxFrameRate="24" par="4:3">
    <Representation id="2" bandwidth="150000" width="480" height="360" frameRate="24">
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a" value="tv1"/>
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_wmpaceinfo"
value="ElephantsDream_H264BPL30_0100.264.dash_wm_pace_info"/>
      <BaseURL>a/ElephantsDream_H264BPL30_0100.264.dash</BaseURL>
      <SegmentBase indexRange="984-11244">
        <Initialization range="0-983"/>
      </SegmentBase>
    </Representation>
    <Representation id="3" bandwidth="250000" width="480" height="360" frameRate="24">
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a" value="tv1"/>
      <EssentialProperty
schemeIdUri="https://dashif.org/guidelines/watermarking_wmpaceinfo"
value="ElephantsDream_H264BPL30_0175.264.dash_wm_pace_info"/>
      <BaseURL>a/ElephantsDream_H264BPL30_0175.264.dash</BaseURL>
      <SegmentBase indexRange="984-11245">
        <Initialization range="0-983"/>
      </SegmentBase>
    </Representation>
  …
  </AdaptationSet>
</MPD>
```

HLS ingest playlists shall include in the media playlist a custom tag specifying the pointer to the sidecar file. The pointer is relative to the ingest manifest. The tag is `#EXT-X-WMPACEINFO:<attribute-list>` where the defined attribute is `URI`, a quoted-string that gives the relative pointer to the sidecar file. In the media playlist for each Variant (A, B, C …), the sidecar file referenced by the `#EXT-X-WMPACEINFO` tag is the same as the `variant` value shall not be considered.

The following is an example of a HLS media playlist, the watermarking signalling is highlighted in bold. Note that the multivariant playlist remains unmodified.

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-WMPACEINFO:URI="main_wm_pace_info"
#EXT-X-MAP:URI="main.mp4",BYTERANGE="1118@0"
```

```
#EXTINF:7.98333,
#EXT-X-BYTERANGE:1700094@1118
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1789481@1701212
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1777588@3490693
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1752144@5268281
a/main.mp4
#EXTINF:7.26667,
#EXT-X-BYTERANGE:1563219@7020425
a/main.mp4
```

### 5.6.5 Packaging Recommendations

This clause contains requirements where packaged content is served to devices. The goal is to facilitate the creation and management of A and B Variants in the delivery chain. These requirements apply even if no re-packaging process exists.

NOTE: This implies that an encoder working against a completely passive receiver (e.g. interface 2 of [i.1]) publishes egress versions of the content directly.

The minimum segment duration should consider the embedding capabilities of the WM technology in order to ensure that a segment contains only information for A or B Variant. A segment carrying only one bit of information (Variant A or B) allows to match a segment to a bit value in the WM pattern.

As described in clause 5.6.3, a re-packaging process may aggregate received parts of content. It builds a segment beginning with the part of content with `firstpart=true` and then aggregates until `lastpart=true` for creating a segment until the targeted length has been reached. It shall begin creating a new segment if a part of content with `firstpart=true` is received before reaching the targeted length. The packager shall not aggregate segments that have inconsistent metadata, more precisely, only ingest segments carrying the same `position` value shall be aggregated together.

The transformation of ingest manifest into egress manifests requires the following actions:

- All `watermarking_wmpaceinfo` and `watermarking_variant` **`EssentialProperty`** elements in DASH manifests and `EXT-X-WMPACEINFO` tags in HLS playlists shall be removed from the egress manifests.
- HLS media playlists of a given rendition in HLS shall be merged into a single, neutral version of it (without `${variantPath}`).
- DASH manifests shall be made neutral (without `${variantPath}`).

While the manifests are made neutral when delivered to devices, the content shall remain stored with the structure defined in the ingest manifests. Doing so, when the CDN edge requests a Variant for a given segment in applying the logic defined in clause 5.7.5, the origin has a direct access to the requested Variant.

Additionally, when translating from ingress to egress, a re-packaging process shall:

- overwrite `WMPaceInfo` when carried as SEI messages, TS adaptation fields or ISOBMFF boxes. Overwriting shall prevent start code emulation. It is recommended to overwrite with `0xFF`.
- remove `firstpart`, `lastpart`, `segmentRegex` from `sidecar-discrete` elements.

## 5.7 Content Playback

### 5.7.1 Introduction

The flow for content playback is shown in the following clauses. The origin received content as explained in clause 5.5. It has access to the A/B Variants and the `WMPaceInfo` data.

This clause describes only the case where the WM token is used in direct mode and does not consider the value of `wmsegduration` (hence using `WMPaceInfo`).

This clause is also not considering the case of download of content for later offline playback. Usually, content available for download is available in the form of byteranges and the device requests large byteranges that overlap those

announced in the MPD or HLS playlists. When content is watermarked, this is not possible as only announced byteranges are addressable (see clause 5.7.5.4). The device shall therefore either use the announced byteranges only or a proxy shall ensure that the edge receives requests that are for announced byteranges.

Content playback is divided in three actions:

- Acquiring the WM token, the DASH manifest, or the HLS playlists
- Acquiring the initialisation segment
- Acquiring media segments

While the first action is common to all type of content, the other ones have variations depending on the packaging and delivery mode of the content. Variation is, for example on the difference between content delivered as byterange or discrete segments. Another possible variation appears when HLS low latency is used for the chunks requested at the edge of live.

The following goes through the different actions by providing the expected workflows.

## 5.7.2 Dynamic Ad Insertion

In case of Dynamic Ad Insertion (DAI), the break may happen at any time. As every segment carries watermarking information allowing to perform the detection, there shall not be segments carrying conflicting data. While some techniques may recover from this mix of data, it will, in all cases, impact the length of content needed for retrieving the unique identifier.

For Live content, assuming that an ad replacement period is defined, then from the device perspective, the following consumption modes are possible.

- The device consumes ads from an alternative edge for the full duration of the ad break
- The device consumes ads from an alternative edge for a duration shorter than the replacement period
- The device consumes the original content as no replacement ad is proposed

Devices may therefore consume content differently during the ad break.

For VOD content, ads will be inserted or stitched with ad break (cue in/out points for example) markers. The device should consume them from an alternative edge for the full duration of the ad break.

The encoder shall watermark ads part of the original content for Live content. The watermarking technology shall remain consistent between all these options. Some devices may receive the original content if no ad can be found for replacement. One consequence is that these devices receive content that is meant to be watermarked following the rules of this document.

Devices receiving an ad for replacement shall receive it from a different edge that does not enforce watermarking. Such edge will then gracefully ignore the WM token.

The WM token is expected to be present in all playback requests during the session. In presence of a DAI manifest manipulator, depending on its behaviour, it may be necessary to tweak the configuration of the delivery pipeline to guarantee the propagation of the WM token. For instance, it may be required to perform some manifest manipulation at the edge to re-introduce the WM token in the response, e.g., when the token is transported as a query parameter and the DAI manifest manipulator is not piggybacking incoming query parameters in the rewritten manifest/playlist. Another case is when the watermark token is incorporated to the virtual path, stripped at the edge on its way to the DAI manifest manipulator (that remains therefore unaware of the WM token) which returns a manipulated playlist that contains absolute URLs.

## 5.7.3 WM Token, DASH Manifest and HLS Playlists Acquisition

The device acquires the WM token in an implementation specific manner. It may be retrieved directly from a WM token server, or it may be provided in a response from another server as part of other data required for playing back content.

The WM token may be added as part of the virtual path of the requested object, as a query string attribute or as part of the HTTP header when the device requests content to the edge. It is recommended to use the virtual path.

The WM token may be added by the device for requesting DASH manifest and HLS playlists. While these objects are not watermarked (the pattern in the name allows the edge to know this), the edge may validate or not the token and refuse to serve these objects if the token is not valid. The edge may also gracefully ignore the token. The origin cleans

the served objects, removing any property related to location of objects (see clause 5.6.5). The manifest and playlist are neutral. This is summarized in Figure 5.

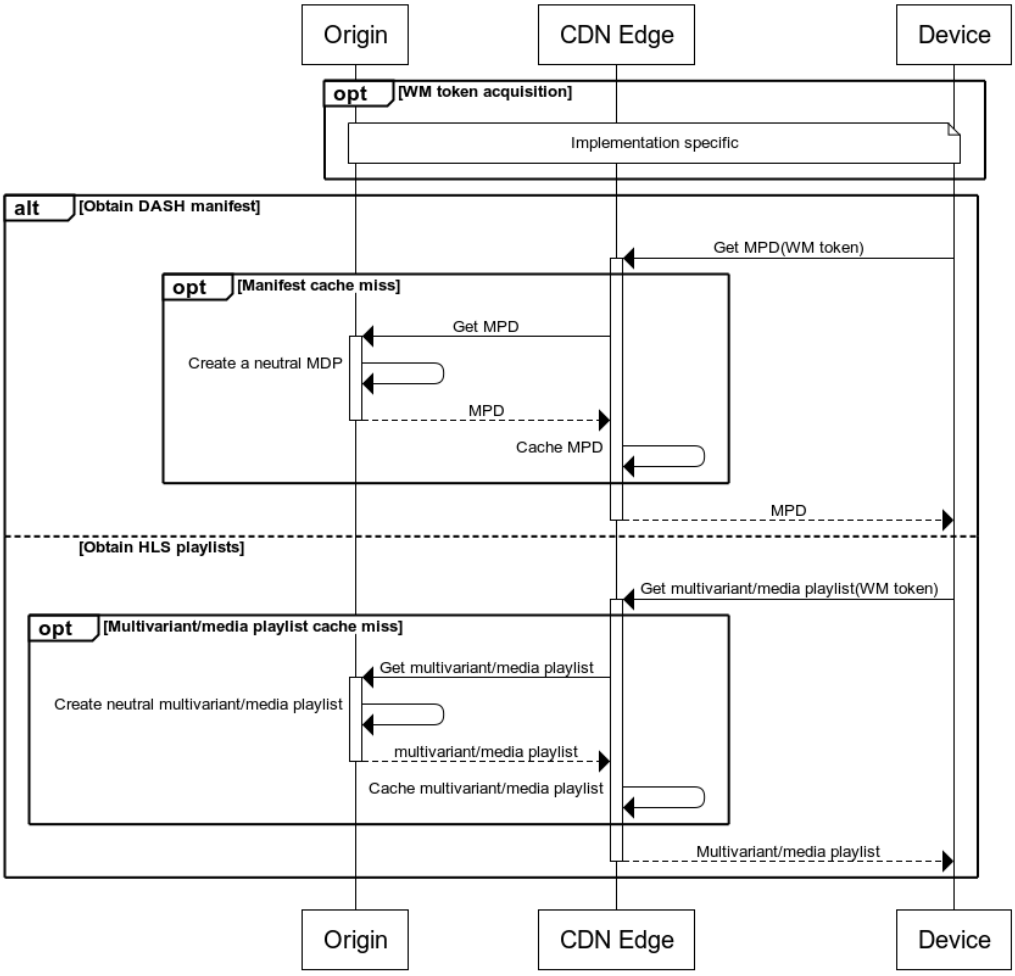


**Figure 5: Token, DASH manifest and HLS playlist acquisition.**

## 5.7.4 Initialisation Segment Acquisition

When content is delivered as byteranges, as the initialisation segment is within the file, the token shall be added in the request as the requested file has a name that matches the pattern for watermarked content. The edge will then apply the exact same logic it applies for a media segment, it retrieves the sidecar file and extracts the `WMPaceInfo` for the first part of the track that contains the initialisation segment (as defined in clause 5.7.5). It can then deliver the initialisation segment to the device. As `position` is equal to -1 (not watermarked), it shall deliver the initialisation segment from Variant A. One or several Variants may become unavailable on the origin for any reason, such as a lost connection with the encoder for these encoding pipelines. Such situation will result in a failed playback if Variant A is the one that is not available. The origin shall deliver to the edge the initialisation segment from any available Variant in this case on the endpoint for Variant A.

NOTE: The token is evaluated and validated as the edge cannot make a difference between the initialisation segment and a media segment.

When content is delivered as discrete segments, the name of the initialisation segment shall not match the pattern for watermarked content as written in clause 5.3. The WM token may be added by the device for requesting the initialisation segment. The edge may validate it or not and may refuse to serve these objects if it is not valid. The edge may also gracefully ignore it.

## 5.7.5 Media Segments and WMPaceInfo Acquisition

### 5.7.5.1 General Requirements

For the media segments, a token shall be attached to the HTTP requests. If not present, the edge shall reject the request and shall not deliver the segment. The edge shall validate the WM token (that can include checking signed data or

decrypting some claims) which is attached to the requests and extracts the WM pattern so that the correct Variant can be sequenced.

Watermarked objects shall include in the sub-path in the edge forward requests to the origin the value of identifying Variants that is part of the configuration described in clause 5.3. A request received at the CDN edge for https://edge.hostname/path/to/endpoint/video_segment_5_8353305.mp4 shall be translated into a forward request for https://origin.hostname/path/to/endpoint/`${variantPath}`video_segment_5_8353305.mp4 where the value of `${variantPath}` depends on the value extracted from the WM pattern for this segment. The same logic applies if the watermarking is done through audio segments.

The connection between the origin and the edge shall be restricted to legitimate requests. How this is achieved is out of the scope of this document.

NOTE: A static secret (a shared key), dynamic signatures or access lists (based on IP addresses) are examples of tools for restricting the access.

There may be the need to disable watermarking within or upstream of the packager at any time, for example, one or several Variants may become unavailable on the origin for any reason, such as a lost connection with the encoder for these encoding pipelines. As devices request all Variants, this situation will result in intermittent black screens when requesting the affected Variants. In such case, `position` shall be set to -1 in `WMPaceInfo`, effectively announcing to the edge sequencing logic that segments are not watermarked. The edge shall then consume segment on the endpoint for Variant A. If this endpoint is not working properly, the origin shall deliver any available Variant on this endpoint.

NOTE: This is breaking the watermarking detection. The period when such contingency measure is applied is not to be used for detection. How the end-to-end system is synchronized is out of the scope of this document. As an example, the origin can raise an alarm.

#### 5.7.5.2 WMPaceInfo Acquisition

For each device request for `/pathname/filename`, the edge shall retrieve from the origin egress `WMPaceInfo` data associated to this object. The origin presents this information differently whether segments are discrete or byteranges:

- For byterange segment, the origin shall have a dedicated endpoint for delivering `WMPaceInfo` information as a sidecar file. For a segment requested by a device at `/pathname/filename`, the origin shall have an endpoint `/pathname/WMPaceInfo/filename` that makes the sidecar file available. The response payload shall contain the sidecar file (as defined in clause 5.5.3.2 for byterange segments). The origin shall not extract data and only provide the sidecar file to the edge. The `Content-Type` for this object is `application/cbor`.
- For discrete segment, the origin
  - Shall have a dedicated endpoint `/pathname/WMPaceInfo/filename` for delivering `WMPaceInfo` for the requested segment. The response payload shall contain a sidecar file that contain a single `WMPaceInfo` object. The `Content-Type` for this object is `application/cbor`.
  - Shall add `WMPaceInfo` in the response header (as defined in clause 5.5.3.3) under the `WMPaceInfoEgress` header field when the edge requests the segment.
- It is the edge that defines which endpoint it uses.

If WMPaceInfo was delivered to the origin in ingress form (as part of the HTTP request headers, SEI message, ISOBMFF box, TS adaptation field or a sidecar file per track), that data shall be extracted and made available in egress form to the edge as both a HTTP header and dedicated endpoint.

Any direct request from a device with `/pathname/WMPaceInfo/filename` shall receive an error code 403.

Table 5 gives examples of content flows as ingest to the origin and egress of the origin to the edge.

**Table 5: Examples of content flows.**

| | Live content | VOD content |
|---|---|---|
| **Ingest of the origin** | No sidecar file, data is delivered as part of HTTP headers, SEI messages, ISOBMFF boxes or TS adaptation field. | For both discrete segments and byteranges, one sidecar file per track. |
| **Egress of the origin** | One sidecar file per segment (note the special case of HLS low latency with byterange where multiple chunks are be linked to the | For discrete segments, one sidecar file per segment and HTTP header.<br>For byterange, one sidecar file per track. |

| | | |
|---|---|---|
| | same sidecar file, see clause 5.7.5.4) and HTTP header. | |

There are then three endpoints on the origin:

- `WMPaceInfo`: /pathname/WMPaceInfo/filename
- Variant A: /pathname/`${variantPath}`filename
- Variant B: /pathname/`${variantPath}`filename

Where `${variantPath}` is as defined in clause 5.3.

NOTE: Adding Variants creates additional endpoints.

### 5.7.5.3 Discrete Files

For the media segments delivered as discrete files, the flow is shown in Figure 6. The edge sequences the A or B Variant of a segment based on the WM pattern contained in the token. It has two options to know the position of the segment within the WM pattern:

- First make a request to the origin to retrieve the `WMPaceInfo` data. This is done with a GET request using the path `/pathname/WMPaceInfo/filename`. The origin provides the `WMPaceInfo` from the Variant A in the payload of the response as a sidecar file.
- Once, the data in `WMPaceInfo` is interpreted in conjunction with the WM pattern, the edge can request to the origin the right Variant corresponding to the position in the WM pattern that matches the value of `position` in `WMPaceInfo` and then deliver it to the device.
- Make a request for the A and B Variants, extract the `WMPaceInfo` from one response header and once, the data in `WMPaceInfo` is interpreted in conjunction with the WM pattern, the edge can deliver the right Variant to the device.

NOTE: There is a high probability that the edge will request both A and B Variants, hence adding `WMPaceInfo` to the response header allows avoiding an extra request to the origin.

The edge caches the Variants of a given segment with different cache keys and it should prevent the cache keys to be revealed through debug headers.

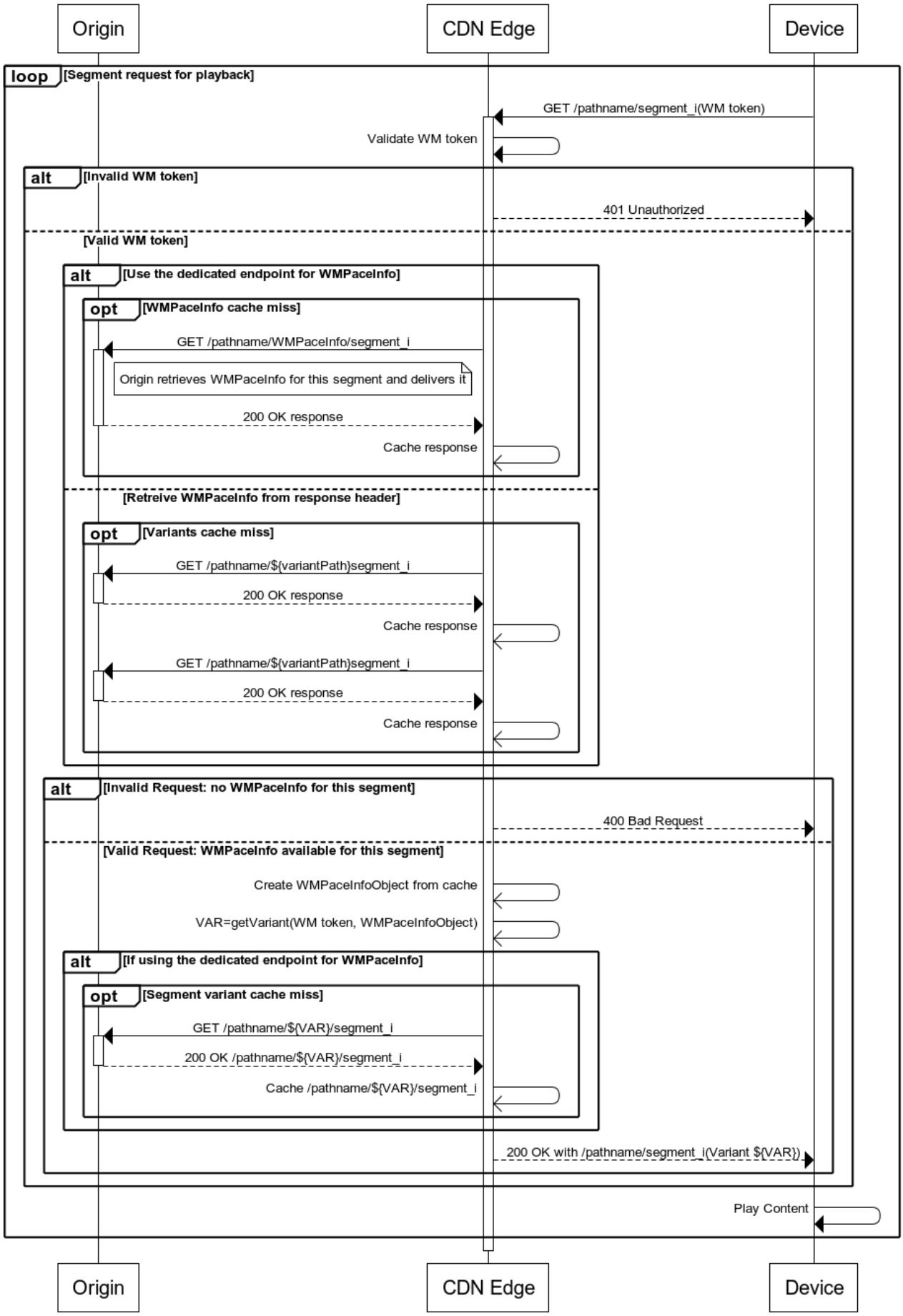


**Figure 6: Media segment, as discrete file, acquisition.**

#### 5.7.5.4 Byterange

For the media segments delivered as byteranges, the flow is shown in Figure 7. The edge delivers the A or B Variant of a segment based on the WM pattern contained in the token. To know which position in the WM pattern it has to consider, it needs to retrieve the sidecar file associated to this track. It first makes a HTTP GET request to the origin in order to retrieve the sidecar file.

Whilst sub ranges within segments, such as chunks, are allowed, the edge shall not deliver byteranges overlapping several segments with different `position` values in `WMPaceInfo`.

NOTE: An example is content delivered with HLS using the `EXT-X-PART` tag are byterange requests within a discrete segment. When the edge receives the request for this partial segment, it will request `WMPaceInfo` to the origin and will receive a sidecar file with only one `WMPaceInfo`. This allows the edge to know that it shall not enforce byterange validation for these requests).

NOTE: Only byteranges overlapping valid ranges are problematic, requests for byteranges included in an allowed range are not breaking the WM pattern that is created by the A/B Variants and thus can be served.

Once the data in `WMPaceInfo` is interpreted in conjunction with the WM pattern, the edge can deliver the correct Variant corresponding to the position in the WM pattern that matches the value of `position` in `WMPaceInfo`.

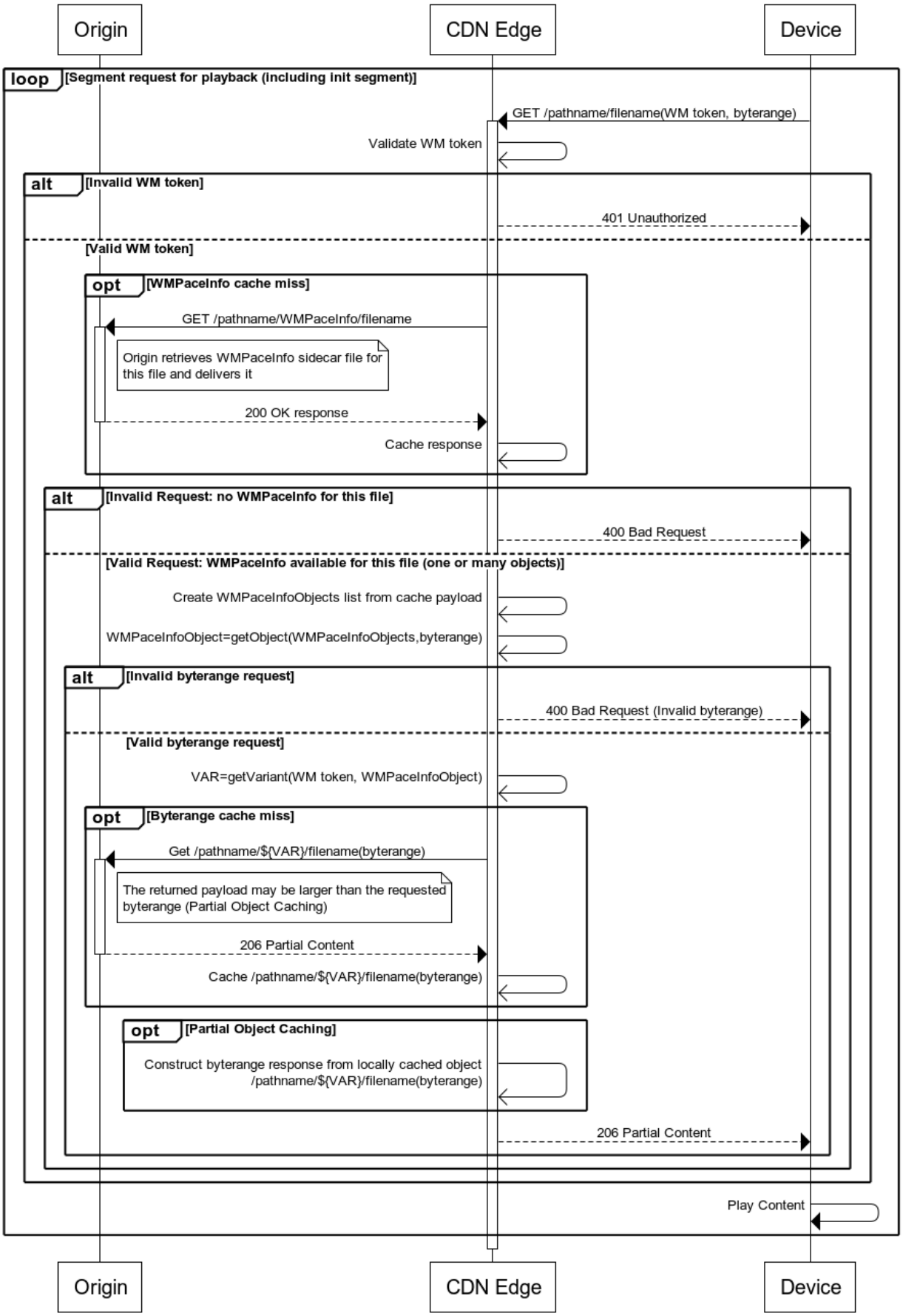


**Figure 7: Media segment, as byterange, acquisition.**

## 5.8 Monitoring and Watermark Detection

If content is found, a detection of a WM pattern can be performed. A video acquisition that includes valuable content (no commercial breaks for example) is performed. As the unique ID is obtained by extracting information from segments (0 or 1 in every segment), the acquired content shall be of several minutes (the longer the segments are, the longer the acquired video is). The video is then processed by the watermarking provider in order to extract the unique ID. This ID is then provided to the relevant entity that can match it to a device, user or streaming session and take the desired actions.

How the detection is performed, and the revocation of the WM token is performed are out of the scope of this document.

# Annex A: (normative)
# Vendor Specific Core API

## A.1 Introduction

In case of a token in indirect mode, it is expected that a vendor specific core (identified by `wmvnd`) generates the WM pattern (referred as `wmpattern`). This means that this requires some interaction between the edge and this vendor specific core. To facilitate this integration, the following defines the API made available by the vendor specific core.

## A.2 Edge-Vendor Specific API

It is assumed that:

- The call to the API function is blocking and the edge waits for the vendor specific core to end its processing.
- The verification of the token is done before the call to the function. Verification includes the validation of the signature.

The inputs are the values of the claims of the token that are relevant for the generation of the WM pattern.

```
const crypto = require('crypto');

function generate_wmpattern (token.wmpatlen, token.wmkeyver, token.wmid, token.wmopid)
{
  /* vendor specific processing */
  return wmpattern;
}
```

# Annex B: (informative) Examples of Workflows

## B.1 Introduction

This annex takes the DASH-IF ingest protocol [i.1] as a reference. There are two interfaces defined:

- Interface 1, where the combination of packager and origin is able to perform additional re-packaging hence the structure of ingest and egress may differ. Each POST/PUT contains one CMAF segment. This is often referred to an active receiving entity as a Just in Time Packager (JITP)
- Interface 2, where the combination of packager and origin does not perform additional re-packaging, the structure of ingest and egress may be the same. The receiving entity is "passive", the source produces all objects in form that devices can consume. Each POST/PUT implicitly refers to one addressable object in an MPD or playlist.

Therefore, the receiving entity is either active (interface 1) or passive (interface 2) and this leads to the following possibilities:

- CMAF ingest, **active** receiving entity (JITP)
- HLS/DASH ingest, **active** receiving entity (JITP)
- HLS/DASH ingest, **passive** receiving entity

Given all the options for carrying `WMPaceInfo` (see clause 5.5.3), the following describes some example flows for Live and VOD content.

## B.2 Live Content Flows

For an **active receiving entity** (JITP), the grouping is non-trivial (as defined in [i.1] clause 6.2), therefore, as described in clause 5.6.4, the manifests are sent. The JITP may aggregate ingress segments according to (`firstpart`, `lastpart`) and `WMPaceInfoEgress` will reflect the aggregated result. In addition, evidence of WM process (such as the essential properties) is removed from egress playlists.

If using the `WMPaceInfoIngest` header field on interface 1, the flow from the encoder to the edge is shown in Figure 8.

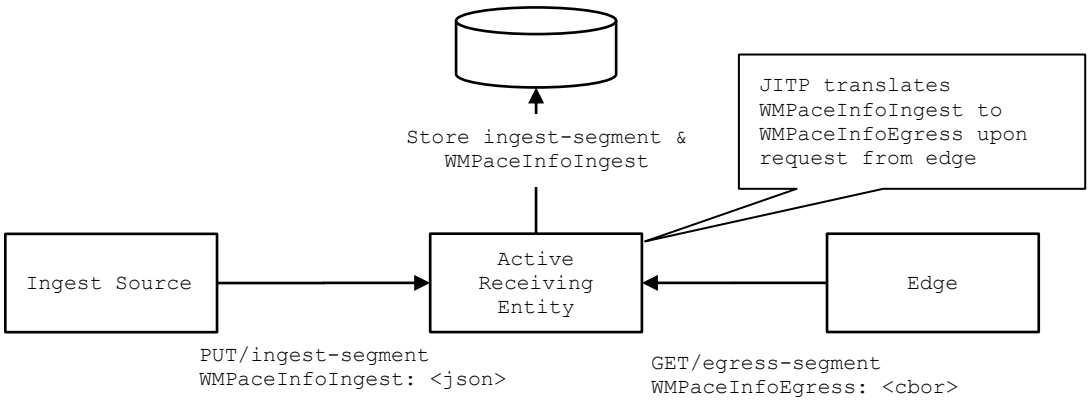


**Figure 8: Flow when using WMPaceInfoIngest and WMPaceInfoEgress header fields.**

Another possible option is using sidecar file, this leads to the flow shown in Figure 9.

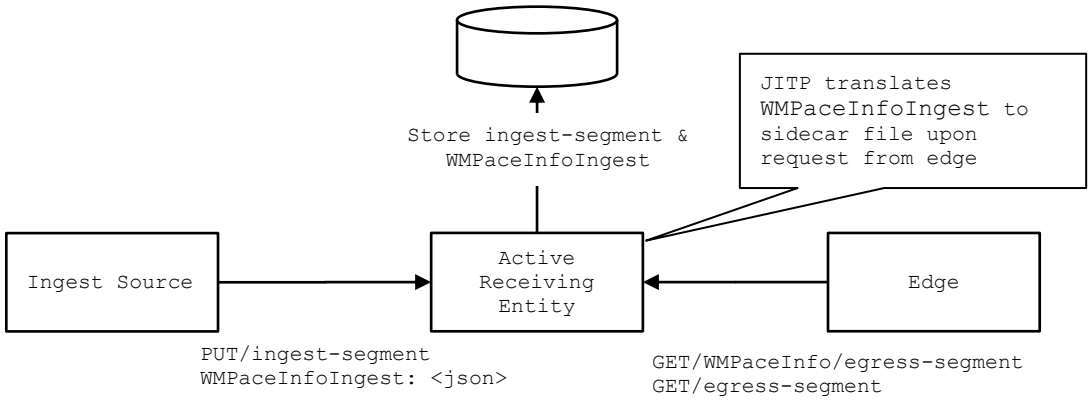


**Figure 9: Flow when using WMPaceInfoIngest header field and sidecar file.**

Another option is using SEI data. In this case, the receiving entity, either leaves `WMPaceInfo` in segment when storing and then overwrites it when serving after translating to `WMPaceInfoEgress` header or overwrites it before storing and saves the `WMPaceInfo` data somewhere else. the flow shown in Figure 10.

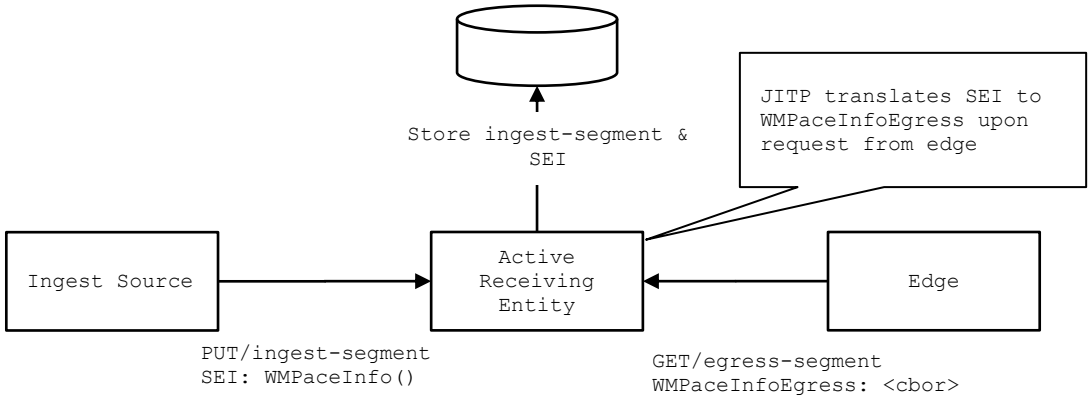


**Figure 10: Flow when using SEI data and WMPaceInfoEgress header field.**

With a **passive receiving entity**, there is no media manipulation downstream of ingest source, therefore transferring `WMPaceInfo` data within the media is not an option, as it is not possible to overwrite it. Figure 11 shows a possible flow with sidecar files.

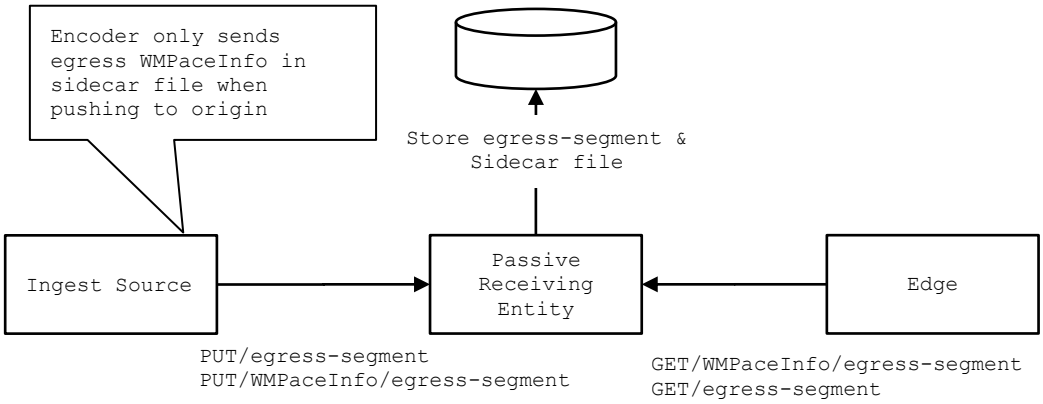


**Figure 11: Flow when using sidecar files.**

# B.3 VOD Content Flows

If VOD content is prepared using live profile, then the permutations presented in clause B.2 are applicable. In addition, another option is that a single sidecar can describe all segments using regex for `segmentRegex`. This latter case leads to the flow shown in Figure 12.

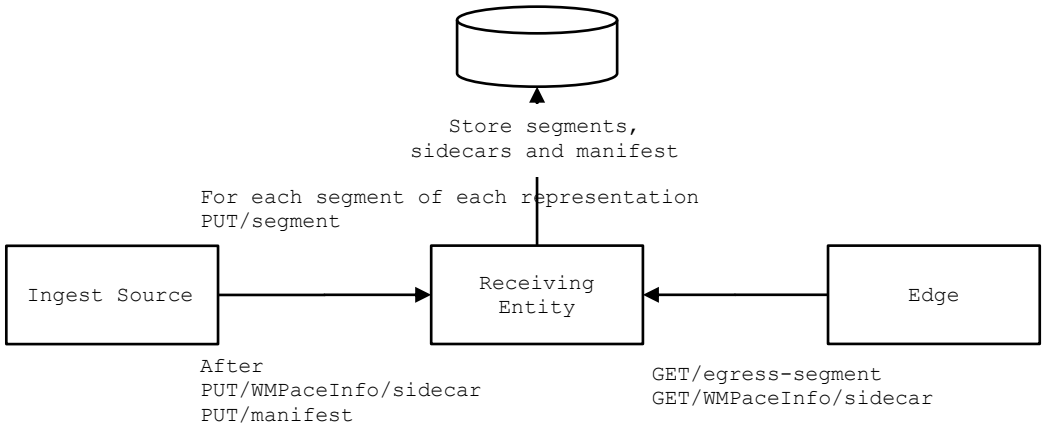


**Figure 12: Flow when using sidecar files for VOD live profile.**

If VOD content is prepared using on-demand profile, then the sidecar file is the only mechanism available to deliver `WMPaceInfo` data. This leads to the flow shown in Figure 13.

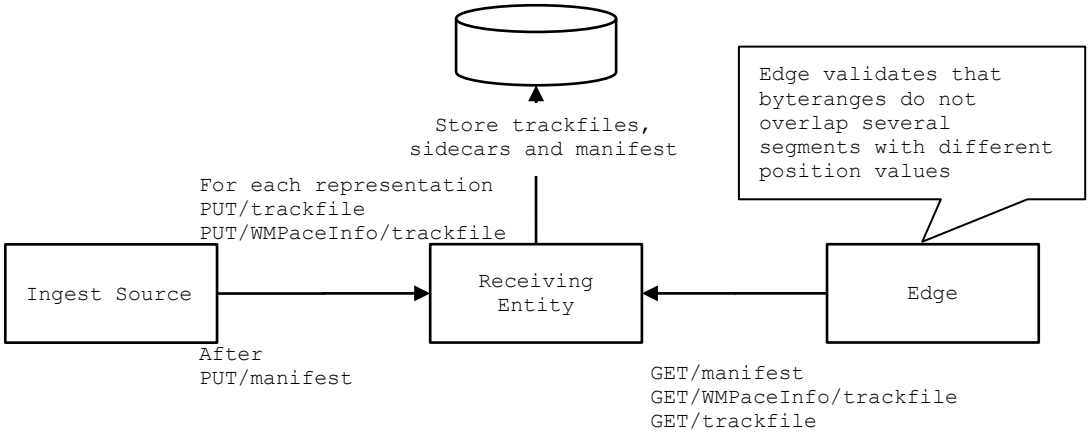


**Figure 13: Flow when using sidecar files for VOD on-demand profile.**

# Annex C: (normative) Registration Requests

## C.1 General

This section contains the registration requests for IANA (token claims) and MP4RA (4CC code).

## C.2 IANA Considerations

This specification requests IANA to register the following claims in the following registry: https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry.

**Version Claim**

- Claim Name: wmver
- Claim Description: The version of the WM Token.
- JWT Claim Name: wmver
- Claim Key: 300
- Claim Value Type: unsigned integer
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Technology Vendor Claim**

- Claim Name: wmvnd
- Claim Description: The WM technology vendor.
- JWT Claim Name: wmvnd
- Claim Key: 301
- Claim Value Type: unsigned integer
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Pattern Length Claim**

- Claim Name: wmpatlen
- Claim Description: The length in bits of the WM pattern.
- JWT Claim Name: wmpatlen
- Claim Key: 302
- Claim Value Type: unsigned integer
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Segment Duration Claim**

- Claim Name: wmsegduration
- Claim Description: The nominal duration of a segment.
- JWT Claim Name: wmsegduration
- Claim Key: 303
- Claim Value Type: map
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Pattern Claim**

- Claim Name: wmpattern
- Claim Description: The WM pattern.
- JWT Claim Name: wmpattern
- Claim Key: 304
- Claim Value Types: COSE_Encrypt0 or COSE_Encrypt or byte string
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**ID Claim**

- Claim Name: wmid
- Claim Description: Used as input to derive the WM pattern for indirect mode.
- JWT Claim Name: wmid
- Claim Key: 305
- Claim Value Type: text string
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Operator ID Claim**

- Claim Name: wmopid
- Claim Description: Used as additional input to derive the WM pattern for indirect mode.
- JWT Claim Name: wmopid
- Claim Key: 306
- Claim Value Type: unsigned integer
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

**Key Version Claim**

- Claim Name: wmkeyver
- Claim Description: The key to use for derivation of the WM pattern in indirect mode.
- JWT Claim Name: wmkeyver
- Claim Key: 307
- Claim Value Type: unsigned integer
- Change Controller: DASH-IF
- Specification Document(s): Section 5.4 of this document

## C.3 MP4RA Registration

This specification requests MP4RA to register the following 4CC code.

1. The name, address, and URL of the organization requesting the code-point.

   DASH-IF
   3855 SW 153rd Dr., Beaverton, OR 97003, USA
   https://dashif.org/

2. The kind of code-point you wish to register (please choose from the set of registered types).

   Boxes (Atoms)

3. For all except object-type registrations, the suggested identifier (four-character code). Note that four-character codes use four 8-bit printable characters, usually from the first 128 Unicode characters (commonly thought of as plain ASCII), but at most from the first 256 Unicode characters.

   `wmpi`

4. The specification in which this code-point is defined, if possible. A copy of the specification would be appreciated, as it enables the authority to understand the registration better. If you are requesting a 'codec' code-point, a reference to the definition of the coding system itself, if separate from the definition of its storage in these files, would also be appreciated.

   Available from here (https://dashif.org/docs/IOP-Guidelines/DASH-IF-CTS-00XX-AB-Watermarking-0.9.pdf).

5. A brief 'abstract' of the meaning of the code-point, perhaps ten to twenty words (see examples on this site).

   wmpi stands for WaterMarkPaceInfo. It carries A/B forensic watermarking information within the ISOBMFF file.

6. Contact information for an authorized representative for the code-point, including:

a. Contact person's name, title, and organization.

   Thomas Stockhammer, DASH-IF Interoperability WG Chair

b. Contact email.

   tsto@ @qti.qualcomm.com

7. Date of definition or implementation (if known) or intended date (if in future).

   To be published by March 31, 2023, as stated here https://dashif.org/guidelines/others/#candidate-technical-specification-dash-if-forensic-a-b-watermarking

8. Statement of an intention to apply (implement) the assigned code-point.

   Expected to be implemented as part of DASH-IF conformance and reference tools according to the boilerplate in the specification

# Annex D: (informative)
# Code for Web Sequence Diagram

## D.1 Introduction

This Annex provides is the code for generating all workflows shown in figures 6 to 9 to be used on https://websequencediagrams.com

## D.2 Figure 6

```
Participant Encoder
Participant Packager
Participant Origin

# STEP 1: Ingest from the encoder to the packager
# For instance, the segmentation is 1s long
Encoder -> Packager: Ingest manifest
Encoder -> Packager: Ingest segments Variant A\n (w/ WMPaceInfo)
Encoder -> Packager: Ingest segments Variant B\n (w/ WMPaceInfo)

# STEP 2: Ingest from the Packager to the Origin (e.g., 2S long segments)
# The Packager has to aggregate several DASH segments to produce the distributed
segment
Packager-> Origin: Egress manifest
Packager-> Origin: Egress segments Variant A\n (w/ WMPaceInfo)
Packager-> Origin: Egress segments Variant B\n (w/ WMPaceInfo)
```

## D.3 Figure 7

```
Participant Origin
Participant CDN Edge
Participant Device

# STEP 1: Acquire a WM token
opt WM token acquisition
    note over Origin,Device: Implementation specific
end

# STEP 2: Get the DASH manifest or HLS playlist for the viewing session
alt Obtain DASH manifest
    Device->+CDN Edge: Get MPD(WM token)
    opt Manifest cache miss
        CDN Edge->+Origin: Get MPD
        Origin->Origin: Create a neutral MPD
        Origin-->-CDN Edge: MPD
        CDN Edge->CDN Edge: Cache MPD
    end
    CDN Edge-->-Device: MPD
else Obtain HLS playlists
    Device->+CDN Edge: Get multivariant/media playlist(WM token)
    opt Multivariant/media playlist cache miss
        CDN Edge->+Origin: Get multivariant/media playlist
        Origin->Origin: Create neutral multivariant/media playlist
        Origin-->-CDN Edge: multivariant/media playlist
        CDN Edge->CDN Edge: Cache multivariant/media playlist
    end
    CDN Edge-->-Device: Multivariant/media playlist
end
```

# D.4 Figure 8

```
Participant Origin
Participant CDN Edge
Participant Device

loop Segment request for playback
    Device->+CDN Edge: GET /pathname/segment_i(WM token)
    CDN Edge->CDN Edge: Validate WM token
    alt Invalid WM token
        CDN Edge-->Device: 401 Unauthorized
    else Valid WM token
        alt Use the dedicated endpoint for WMPaceInfo
            opt WMPaceInfo cache miss
                CDN Edge->+Origin: GET /pathname/WMPaceInfo/segment_i
                note right of Origin
                    Origin retrieves WMPaceInfo for this segment and delivers it
                end note
                Origin-->-CDN Edge: 200 OK response
                CDN Edge ->> CDN Edge: Cache response
            end
        else Retreive WMPaceInfo from response header
            opt Variants cache miss
                CDN Edge->+Origin: GET /pathname/${variantPath}segment_i
                Origin-->-CDN Edge: 200 OK response
                CDN Edge ->> CDN Edge: Cache response
                CDN Edge->+Origin: GET /pathname/${variantPath}segment_i
                Origin-->-CDN Edge: 200 OK response
                CDN Edge ->> CDN Edge: Cache response
           end
        end
        alt Invalid Request: no WMPaceInfo for this segment
            CDN Edge-->Device: 400 Bad Request
        else Valid Request: WMPaceInfo available for this segment
            CDN Edge ->> CDN Edge: Create WMPaceInfoObject from cache
            CDN Edge ->> CDN Edge: VAR=getVariant(WM token, WMPaceInfoObject)
            alt If using the dedicated endpoint for WMPaceInfo
                opt Segment Variant cache miss
                    CDN Edge->+Origin: GET /pathname/${VAR}/segment_i
                    Origin-->-CDN Edge: 200 OK /pathname/${VAR}/segment_i
                    CDN Edge ->> CDN Edge: Cache /pathname/${VAR}/segment_i
                end
            end
            CDN Edge-->Device: 200 OK with /pathname/segment_i(Variant ${VAR})
        end
    end
    Device->Device: Play Content
End
```

# D.5 Figure 9

```
Participant Origin
Participant CDN Edge
Participant Device

loop Segment request for playback (including init segment)
    Device->+CDN Edge: GET /pathname/filename(WM token, byterange)
    CDN Edge->>CDN Edge: Validate WM token
    alt Invalid WM token
        CDN Edge-->Device: 401 Unauthorized
    else Valid WM token
        opt WMPaceInfo cache miss
            CDN Edge->+Origin: GET /pathname/WMPaceInfo/filename
            note right of Origin
                Origin retrieves WMPaceInfo sidecar file for
                this file and delivers it
            end note
```

```
            Origin-->-CDN Edge: 200 OK response
            CDN Edge ->> CDN Edge: Cache response
        end
        alt Invalid Request: no WMPaceInfo for this file
            CDN Edge-->Device: 400 Bad Request
        else Valid Request: WMPaceInfo available for this file (one or many objects)
            CDN Edge ->> CDN Edge: Create WMPaceInfoObjects list from cache payload
            CDN Edge ->> CDN Edge: WMPaceInfoObject=getObject(WMPaceInfoObjects,
byterange)
            alt Invalid byterange request
                CDN Edge-->Device: 400 Bad Request (Invalid byterange)
            else Valid byterange request
                CDN Edge ->> CDN Edge: VAR=getVariant(WM token, WMPaceInfoObject)
                opt Byterange cache miss
                    CDN Edge->+Origin: Get /pathname/${VAR}/filename(byterange)
                    note right of Origin
                        The returned payload may be larger than the requested
                        byterange (Partial Object Caching)
                    end note
                    Origin-->-CDN Edge: 206 Partial Content
                    CDN Edge ->> CDN Edge: Cache /pathname/${VAR}/filename(byterange)
                end
                opt Partial Object Caching
                    CDN Edge->>CDN Edge: Construct byterange response from locally
cached object\n/pathname/${VAR}/filename(byterange)
                end
                CDN Edge-->Device: 206 Partial Content
            end
        end
    end
    Device->Device: Play Content
End
```

# Annex (informative): Change History

| Date | Version | Information about changes |
|---|---|---|
| 2022-03-23 | 0.8.0 | Version published for first community review. |
| 2023-02-02 | 0.9.0 | Version published for second community review. |
| 2023-02-09 | 0.9.1 | Added IANA and MP4RA registration annexes. |
| 2023-05-02 | 0.9.2 | (editorial) Corrections on broken automatic references and few formatting issues.<br>(editorial) Changed Master to Multivariant (HLS).<br>(IANA) Updated the sidecar file integer keys and claim keys with final values.<br>Removed remaining “must” from the text.<br>Added `COSE_Encrypt` option for wmpattern.<br>Updates on the `variantPath` construction options (removed the ‘.’ possibility).<br>Deleted examples for token claims.<br>Changed the encryption algorithm for the pattern (align with CTA WAVE CAT).<br>Clarified the storage paths for the Variant on the origin.<br>Clarified the order of the bits in the WM pattern. |
| 2023-05-03 | 0.9.3 | Version published for IOP Review with some small editorial updates |
| 2023-05-09 | 0.9.5 | Version created for IPR Review and ETSI Submission |
| 2023-06-12 | 1.0.0 | Version published on DASH-IF Website |