CTA Specification

Wave Application Video Ecosystem – Device Playback Capabilities

CTA-5003

December 2018

Consumer Technology Association™

(Formulated under the cognizance of the CTA **WAVE Project** in cooperation with the W3C; for information please see cta.tech/WAVE.)

Published by
CONSUMER TECHNOLOGY ASSOCIATION
Technology & Standards Department
www.cta.tech

Table of Contents

Table of Figures

Forward

This document was developed by the Web Application Video Ecosystem (WAVE) Project of the Consumer Technology Association[1]. The WAVE Project is a broad industry initiative of content, technology, infrastructure and device companies, all working together towards commercial Internet video interoperability based on industry standards.

[1] See https://cta.tech/WAVE

Web Application Video Ecosystem – Device Playback Capabilities

1 Scope

The scope of this document is to define normative requirements around playback of CTA WAVE content, i.e., primarily segmented media content. These requirements will be applicable to HTML 5 based playback of type 1 and type 3 as well as non-HTML 5 devices. Playback of content includes detecting the ability to playback WAVE content and programs (where WAVE programs are sequences of CMAF presentations) and playing back the content in different scenarios (regular, random access, chunked mode, etc.).

2 References

2.1 Normative References

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

These normative references are intended to include corrigenda and amendments available at the time of use.

[WAVE-CON]	*Web Application Video Ecosystem (WAVE) Content Specification*, Consumer Technology Association (CTA), 2018 Edition, April 2018.
[WAVE-WMA]	*Web Media API Snapshot 2017*, Draft Community Group Report 13 February 2018, https://w3c.github.io/webmediaapi/
[CMAF]	ISO/IEC 23000-19, *Information technology — Coding of audio-visual objects — Part 19: Common media application format (CMAF) for segmented media*. https://www.iso.org/standard/71975.html
[CMAF A1]	ISO/IEC 23000-19:2018, *Information technology — Coding of audio-visual objects — Part 19: Common media application format (CMAF) for segmented media, Amendment 1.*

2.2 Informative References

The following documents contain provisions that, through reference in this text, constitute informative provisions of this document. At the time of publication, the editions indicated were valid. All documents are subject to revision, and parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the documents listed here.

[MEDIA-SOURCE]	*Media Source Extensions*, W3C Recommendation 17 November 2016, http://www.w3.org/TR/media-source/
[ENCRYPTED-MEDIA]	*Encrypted Media Extensions*, W3C Recommendation 18 September 2017, http://www.w3.org/TR/encrypted-media/
[MSE-FORMAT-ISOBMFF]	*ISO BMFF Byte Stream Format*, W3C Working Group Note 04 October 2016, http://www.w3.org/TR/mse-byte-stream-format-isobmff/
[HTML51]	*HTML 5.1 2nd Edition*. Steve Faulkner; Arron Eicholz; Travis Leithead; Alex Danilo. W3C. 3 October 2017. W3C Recommendation. https://www.w3.org/TR/html51/
[ECMASCRIPT-5.1]	*ECMAScript Language Specification, Edition 5.1*. Ecma International. June 2011. Standard, http://www.ecma-international.org/publications/standards/Ecma-262.htm
[WEBAUDIO]	*Web Audio API*. Paul Adenot; Chris Wilson; Chris Rogers. W3C. 8 December 2015. W3C Working Draft. https://www.w3.org/TR/webaudio/
[DASH]	ISO/IEC 23009-1: *Dynamic Adaptive Streaming over HTTP: Media Presentation Description and Segment Formats*

3 Document Notation and Conventions

The following terms are used to specify conformance elements of this specification. These are adopted from the ISO/IEC Directives, Part 2, Annex H [ISO-P2H ISO-P2H]. For more information, please refer to those directives.

- SHALL and SHALL NOT indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.
- SHOULD and SHOULD NOT indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.
- MAY and NEED NOT indicate a course of action permissible within the limits of the document.

Terms defined to have a specific meaning within this specification will be capitalized – e.g., “Track”, and should be interpreted with their general meaning if not capitalized.

4 Acronyms

API	Application Programming Interface
AVC	Advanced Video Coding
BMFF	Base Media File Format
CBC	Cypher Block Chaining
CDM	Content Decryption Module
CEA	Consumer Electronics Association
CENC	MPEG Common ENCryption
CMAF	MPEG Common Media Application Format
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CTA	Consumer Technology Association
CTR	CounTeR block cipher mode
DASH	Dynamic Adaptive Streaming over HTTP
DRM	Digital Rights Management
ECMASCRIPT	European Computer Manufacturers Association SCRIPTing
EME	Encrypted Media Extensions
HbbTV	Hybrid broadcast broadband TV
HDCP	High-bandwidth Digital Content Protection
HDMI	High- Definition Multimedia Interface
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEC	International Electrotechnical Commission
ISO	Abbreviated name of the International Organization for Standardization
JSON	JavaScript Object Notation
KID	Key IDentifier
MIME	Multipurpose Internet Mail Extensions
MPD	Media Presentation Description
MSE	Media Source Extensions
NAL	Network Abstraction Layer
PSSH	Protection System Specific Header
UHD	Ultra High Definition
URL	Uniform Resource Locator
UTC	Coordinated Universal Time

WAVE	Web Application Video Ecosystem
XHTML	Extensible HyperText Markup Language

5 Architecture and WAVE Device Reference Model

5.1 WAVE Architecture

The WAVE architecture considers three primary domains: WAVE content, an application that is based on WAVE APIs and the WAVE Device Platform. WAVE specifications are an enabler to support

- Generation of content independently of an application that can be played back on WAVE devices using well defined content formats, playback APIs and device functionalities.
- Implementation of device platforms that enable playback of commonly generated content through well-defined APIs.
- Development of media applications that enable playback of commonly generated content on a broad variety of devices using common APIs.

This architecture enables an ecosystem of independent content generation, app development and device implementations and permits the use of same content within different apps as well as across many different device platforms.

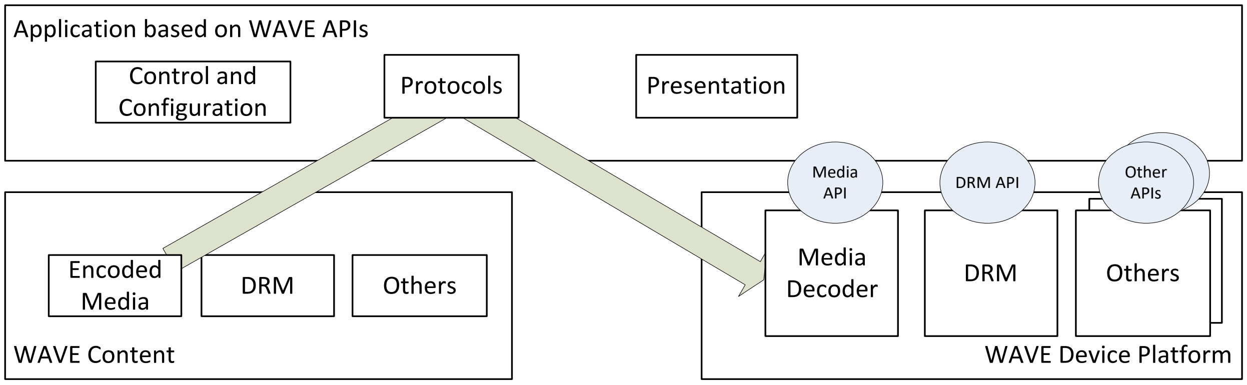


Figure 1: WAVE Architecture Model

This WAVE specification primarily deals with the requirements of a WAVE Device Platform that may be used by an application implementing WAVE APIs to play back WAVE content. The requirements are purposely held abstract in order to support different application and device interface models. Nevertheless, the use of HTML 5 APIs defined in [WAVE-WMA] is one of the prime objectives. The APIs also differentiate between devices supporting different playback variants, primarily Type 1 and Type 3:

- Type 1 Playback: The WAVE device platform receives a manifest and downloads and plays back the contained media based on the information in the manifest. An application may control the playback with limited control features.
- Type 3 Playback: The WAVE application receives a manifest, downloads the media and uses media APIs in order to playback individual tracks of the media experience. The application is in control of the download and playback of the media using track buffers.

5.2 WAVE Device Playback Reference Model

5.2.1 Overview

Based on the architecture in clause 5.1, this clause defines abstracted device models. A device model for Type 3 playback is shown in Figure 3 and a device model for Type 1 playback is shown in Figure 4.

The focus of this initial specification is on Type 3 playback – i.e., the application has access to the streaming manifest and parses and processes the manifest.

This specification follows the HTML 5 and MSE model as shown in Figure 2. A Media Element provides an output and control environment for the playback of media data. A Media Source object represents a source of media data that can be addressed by an application. The *MediaSource* combines a list of *SourceBuffer* objects that can be used to add media data to the presentation. MediaSource objects are created by the application. The application uses the SourceBuffer objects to add media data to this source. In addition, Encrypted Media Extensions (EME) and a Content Decryption Module (CDM) support APIs and functionalities for decryption.

While this specification attempts to abstract from the concrete instantiation in HTML Media element and MSE implementation, in case of ambiguities the terminology for the HTML 5/MSE instantiation applies.

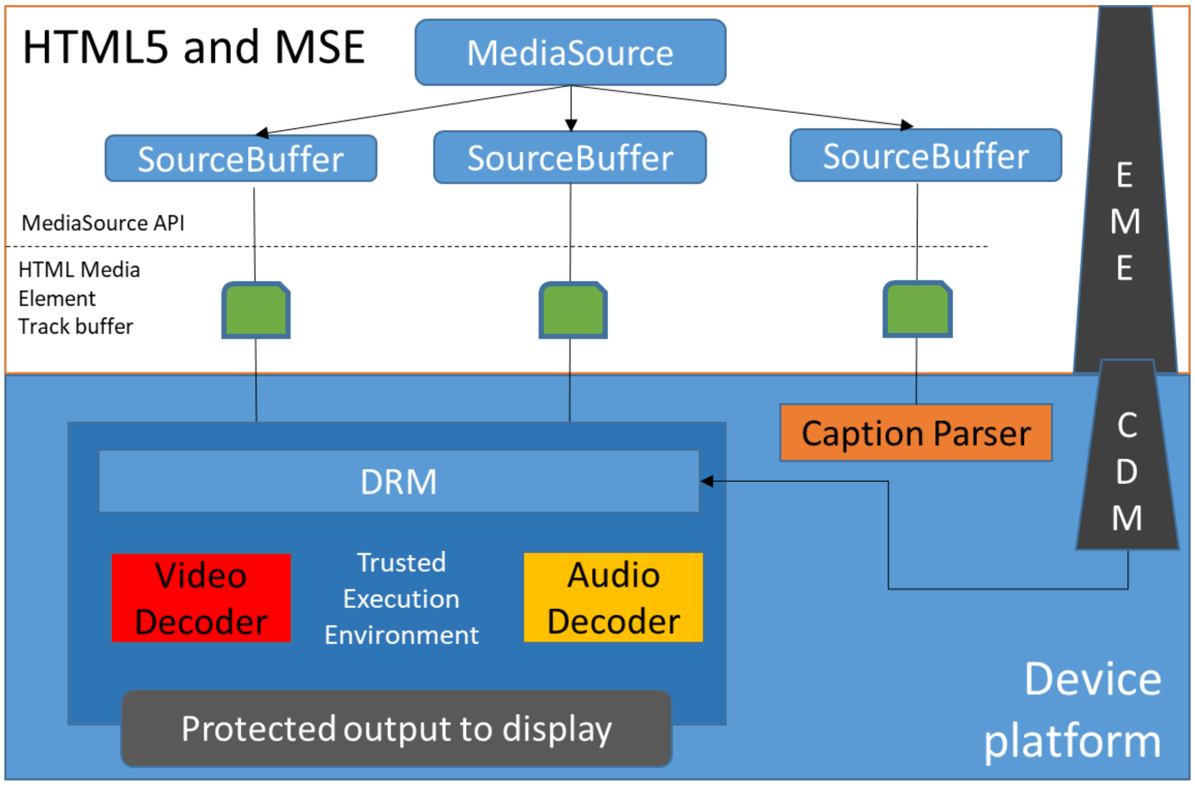


Figure 2: HTML 5 and MSE based Media Source model

In the context of this specification, as CTA WAVE content relies on CMAF content, and CMAF content only uses non-multiplexed tracks, a source buffer is directly mapped to a single track buffer. Hence, these two terms are used synonymously and interchangeably.

An application that receives a manifest referring to CTA WAVE content is expected to have access to two primary high-level APIs:

- Control API: This API is primarily responsible for capability discovery of the device platform, establishing and tearing down media source objects and source/track buffers for specific media types, as well as to control the playback of media.
- Media API: This API consists of one or multiple track/source buffers where the source buffers can be dynamically established and removed. The track/source buffers enable playback of WAVE content by the device platform.

In the case of a streaming application, the application deals with manifest updates as well as with providing/downloading the segments that are then forwarded to the device platform for playback using the media APIs.

The device platform is expected to provide rendering capabilities. A video display region and an audio device are available as output. The final control of the playback of the media may be handled by the application or it may be part of the device platform. However, it is assumed that a device-oriented mode exists for which playback is primarily the task of the device.

Such a mode may be used in a testing environment such that the device's video and audio output are used to observe if the device platform is capable of fulfilling a set of playback requirements. The application is out of scope for this specification but is assumed that any application that supports the methods and functions of the control and media API can use the device to playback content.

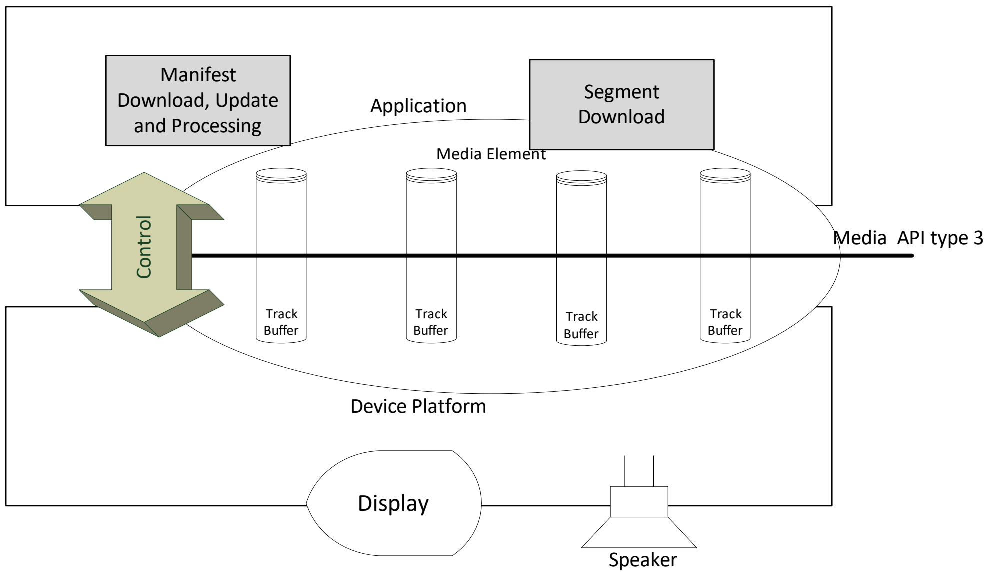


Figure 3: Abstracted device model for Type 3 playback

For type 1 playback, the application has access to the URL of a streaming manifest, but the URL is provided to the device platform through a different type of API. The downloading and processing of the manifest as well as the downloading of the segments is native to the device platform. Application to device platform communication is again split into two primary APIs:

- The control API: This API is primarily responsible for capability discovery of the device platform, establishing and tearing down video elements buffers for specific codecs, as well as to control the playback of media through video elements.
- The media API: this API consists of one or multiple video elements where the video elements can be dynamically established and removed. Each video element enables playback of WAVE content made available through a proper manifest.

Any interfaces between the device platform and external displays or speakers is out of scope of this specification.

In this specification, the primary focus is on type 3 based playback – i.e. it is assumed that the application does manifest download and processing and can establish and playback WAVE content using a set of track buffers. The application of the requirements in this specification to type 1 playback may be addressed in a future version of the specification.

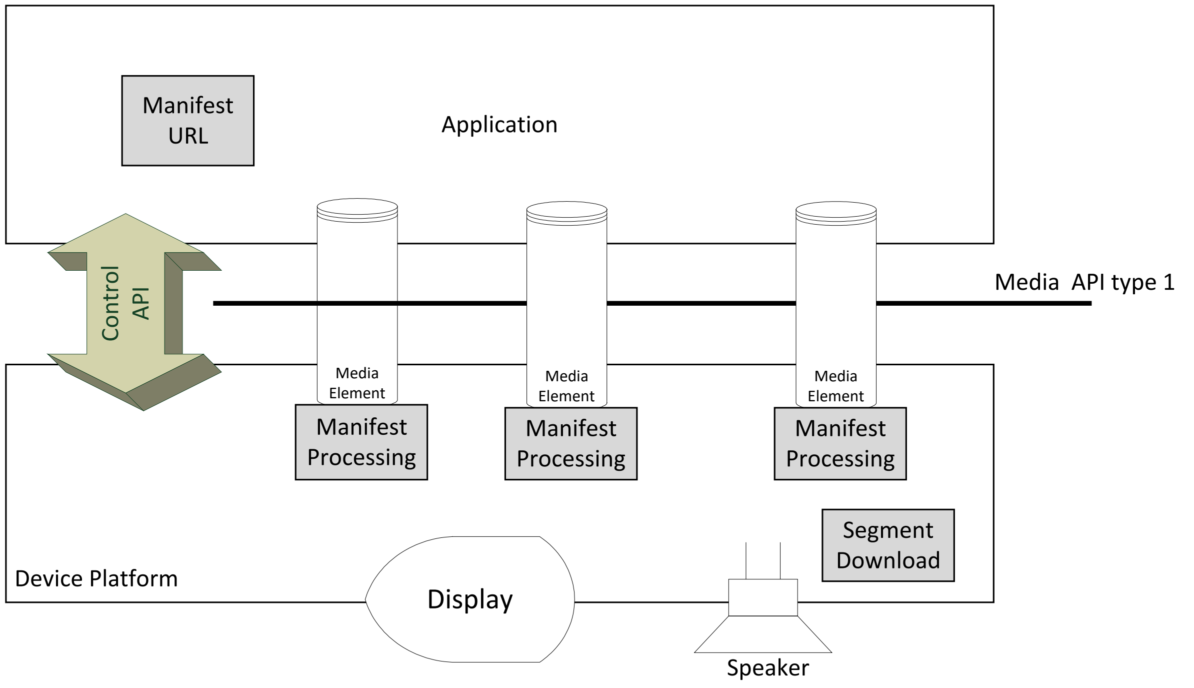


Figure 4: Abstracted device model for Type 1 playback

5.2.2 Wave Device Platform APIs

5.2.2.1 Introduction

Following Figure 3, the usage of APIs enables abstraction to the device and playback model. The differentiation between control APIs and media APIs relates to the two complementary WAVE specifications. The control API focuses on the application functions, whereas the media API focuses on playback of WAVE content.

5.2.2.2 Control API

The control API includes all functions to establish, maintain and control media playback. Control APIs are typically used for the initialization phase or at the change of configurations.

5.2.2.3 Media API

The media API includes all functions to feed and playback WAVE content.

5.2.2.4 Video and Audio Output

With setting up a source buffer for a video media type, either:

- a pre-determined display window that matches the aspect ratio and size of the content (height and width) is established, or
- a full-screen display is established.

With setting up a source buffer for an audio media type, an audio output is established.

5.2.3 Web Media API-based Playback Model

This specification is primarily written with the Web Media API-based Playback Model [WAVE-WMA] in mind.

The following two core specifications are mandated in [WAVE-WMA]

- HTML 5.1 [HTML51], devices acting as Web browsers that support the HTML syntax and the XHTML syntax, scripting, and the suggested default rendering.
- ECMAScript Language Specification, Edition 5.1 [ECMASCRIPT-5.1].

For Media playback, the following three specifications are mandated in [WAVE-WMA]:

- Encrypted Media Extensions [ENCRYPTED-MEDIA].
- Media Source Extensions [MEDIA-SOURCE].
- Web Audio API [WEBAUDIO] with some exceptions.

It is expected that any device playback tests testing the requirements in this specification rely on the APIs mentioned above.

This specification provides an overview of the key functions and methods that are available for applications, both for testing and for playback. The specification is written in a way such that the functions and methods provided by the above specifications can directly be used for testing and playback.

5.3 WAVE Content

5.3.1 Overview

The WAVE Content [WAVE-CON] specification defines WAVE content. As indicated in clause 3 of [WAVE-CON], WAVE content is based on CMAF [CMAF]. This document also introduces models on the CMAF content model.

WAVE Content is offered as WAVE Programs. A WAVE Program is a sequence of CMAF Presentations, played back sequentially. Sequential playback means that the timing of the playback is controlled by the application, including the ability to create smooth playback without interruptions.

Clause 5.3.2 introduces the CMAF Content Model and clause 5.3.3 introduces the WAVE Content Model based on CMAF.

5.3.2 CMAF Content Model

5.3.2.1 Overview

The CMAF content model is shown in Figure 5.

This specification uses CMAF defined terms according to the CMAF definitions. The key terms the reader of this specification should be familiar with are:

- CMAF Presentation
- CMAF Switching Set
- CMAF Track
- CMAF Header, CMAF Chunk, CMAF Fragment, CMAF Segment
- Decode times of samples
- Presentation times of samples

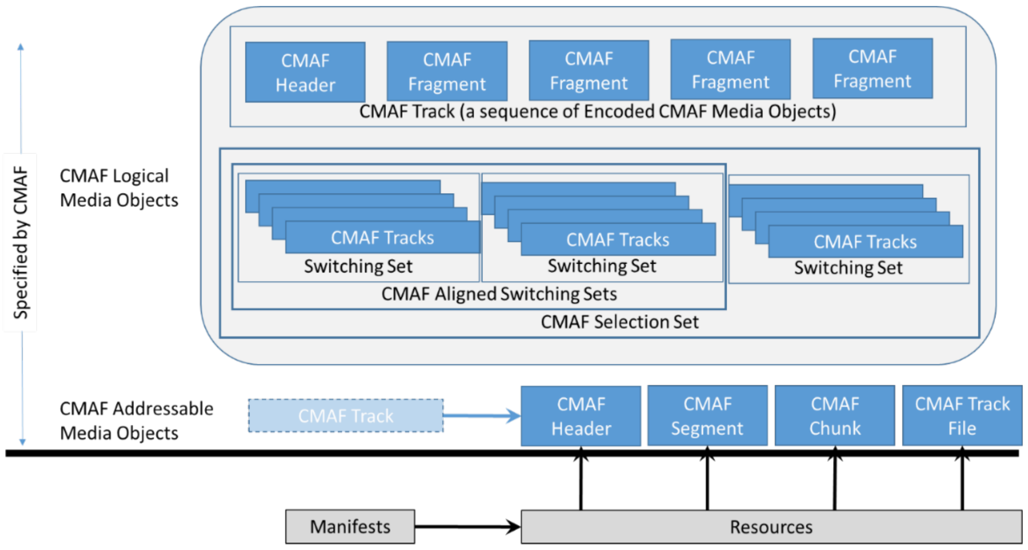


Figure 5: CMAF Content Model

A media sample is media data in a CMAF track associated with a single decode start time and duration.

5.3.2.2 CMAF Addressable Objects

This section provides an overview on CMAF addressable objects, namely

- CMAF Track Structure in [CMAF].

- CMAF Chunk Structure in [CMAF].
- CMAF Fragment Structure in [CMAF].
- CMAF Segment Structure in [CMAF].

For details, please refer to the CMAF specification [CMAF].

In the remainder of this specification, it is assumed that for test purposes the application can access CMAF Headers, CMAF Fragments, continuous byte ranges of a single CMAF Fragment of arbitrary size and CMAF Chunks as units and hand those to the media API. The access may for example be through HTTP or any other protocol but could also access the data from a local file storage.

Note that in practical applications, CMAF Fragments and CMAF Chunks may be embedded in Segments (e.g., DASH or HLS). Normally segments are the addressable units, i.e. units with an assigned URL. However, for the purpose of testing, it is assumed that CMAF Fragments are addressable (e.g., by a 1-to-1 mapping of CMAF Fragments to Segments) and CMAF Chunks are accessible or addressable (e.g., by a 1-to-1 mapping to DASH delivery units or by using a chunk index as part of Segment).

5.3.2.3 CMAF presentation timing model

There are multiple timelines involved in playout and rendering CMAF tracks within a presentation.

Each track is a sequence of timed samples. Each sample has a decode time and may also have a composition (display) time offset. Edit lists may be used to over-ride the implicit direct mapping of the media timeline, into the timeline of the overall movie. The movie timeline is used to synchronize CMAF Tracks in a CMAF presentation and also serves as the synchronization source for playback in an HTML 5 media element and the media source.

In addition, each CMAF Track may have assigned an anchor wall-clock time – e.g., UTC time. The wall clock time may be used to relate the relative presentation time of the track to a wall-clock time, for example expressing the time when the corresponding sample was captured, encoded, or packaged.

In summary, three timelines exist and the signaling of the timeline in CMAF is summarized as follows:

- Decode time: The decode time of each CMAF chunk is provided as the baseMediaDecodeTime in a TrackFragmentBaseMediaDecodeTimeBox. This provides the decode time of the first sample in the CMAF chunk and the remaining decode times are derived by the sample durations in the 'traf' box.
- Presentation time: The presentation time of each sample in a fragment is determined by the decode time of the sample and, if present, the composition offset (in the sample table) and the track edit list (in the track header). The earliest presentation time in a CMAF Fragment is important for synchronization and switching. Note that the earliest

presentation time of a CMAF Fragment may not be the presentation time of the first sample of the CMAF Fragment. In the remainder of the document, presentation time is also referred to as *media time*.
- Wall-clock time: By the use of a ProducerReferenceTimeBox ('prtf'), the sample with a specific decode time can be mapped to wall-clock time.

5.3.2.4 CMAF Track Model for this Specification

For the specification, the following definitions are relevant for CMAF Tracks. For each CMAF Track *k (k=1,...,K)* in a CMAF Switching Set, the following features are defined:

- CMAF Header `CH[k], k=1,…,K`
- CMAF Fragments `CF[k,i], i = 1,2,3,… N`
    - Position in the CMAF track `i`
    - Earliest presentation time: `tf[k,i]`
    - CMAF Fragment duration: `df[k,i] = tf[k,i+1]-tf[k,i]`
    - Wall-clock time assigned to the earliest presentation time of CMAF Fragment: `twc[k,i]`
    - CMAF Chunks `CC[k,i,j], j = 1,2,3,…, C[i]`
        - Position in the fragment `j`
        - Earliest decode time `tc[k,i,j]`
        - Chunk duration in decode times `dc[k,i,j]`
- An edit list EL[*k*] that may be present in the CMAF header describing the difference between the composition time and the presentation for this track in the CMAF Presentation.
- The earliest presentation time of the first fragment, i.e. `tf[k,i=1]` (presentation time offset)
- The duration of the CMAF Track is defined as `td[k]`
- The CMAF Track has an assigned media profile, which includes:
    - CMAF media profile brand
    - Suitable MIME Type string providing
        - Media type
        - Codecs parameter
        - Profiles parameter <reference>
- The CMAF Track has samples `sample[k,s]` with s=1, …, S, each with nominal presentation time `T[k,s]`.

5.3.2.5 CMAF Switching Set Model for this Specification

The following defines a Switching Set:

- A set of CMAF Tracks conforming to the conditions in clause 5.3.2.4.
- The CMAF Switching Set may contain a single CMAF Header for all CMAF Tracks or an individual CMAF Header for each CMAF Track.

- A Master CMAF Header CH*. Either the single header or a Master CMAF Header assigned to the Switching Set `CH*`.

From the definition of a CMAF Switching Set [CMAF], the following holds:

- All CMAF Tracks in a Switching Set conform to one media profile.
- There exists one CMAF Header that is used to initialize the playback of the Switching Set. This header referred as Master CMAF Header CH*.
- The CMAF Header for each track in a Switching Set is defined such that appending it to the source buffer does not result in a reinitialization of the decoding and rendering platform.
- Each CMAF Track in a Switching Set has the same number of CMAF Fragments.
- The earliest decoding time of each CMAF Fragment at the same position `i` in different CMAF Tracks of a CMAF Switching Set are identical.
- The earliest presentation time of each CMAF Fragment at the same position `i` in different CMAF Tracks of a CMAF Switching Set are identical.
- The fragment duration of each CMAF Fragment at the same position in different CMAF Tracks or a CMAF Switching Set are identical.

Note that the equalities above only hold for CMAF Fragments, not necessarily for CMAF Chunks.

5.3.3 WAVE Content Model

5.3.3.1 Overview

WAVE defines a program model and continuous switching sets.

5.3.3.2 WAVE Presentations and Programs

WAVE Programs are a sequence of WAVE Presentations that are played consecutively [WAVE-CON][CMAF]. WAVE presentations in one WAVE program may follow certain constraints and restrictions.

A WAVE Presentation is defined as a collection of WAVE Selection Sets. For each Selection Set, it is expected that one Source Buffer is established using the associated CMAF master header and media type. Typically, at least one Selection Set for audio and one Selection Set for video is available.

Each Selection Set contains one or multiple Switching Sets as defined in clause 5.3.2.5.

All tracks in a presentation follow the presentation timing model of clause 5.3.2.3.

5.3.3.3 WAVE Splice Constraints

WAVE Splice constraints [WAVE-CON] are defined to ensure seamless playback across presentation boundaries.

5.3.3.4 WAVE Continuous Switching Sets

Continuous switching sets permit to continue playout media without timeline updates.

- For each track *k* in a switching set *s* that is a continuation of a switching set *r*, the following features are defined and requirements hold:
    - For CMAF Header `CH[s,k] = CH[r,k]`.
    - Decode time of first fragment of *s* equals decoding time of last fragment plus fragment duration.

The earliest presentation time of the first fragment of the switching set *s* equals the earliest presentation time of the last fragment of *r* plus the last fragment duration of *r*: `tf[s, k, i=1] =  tf[r, k, i=N] + df[s, k, i=N]`.

5.3.3.5 WAVE Splice Conditioned Switching Sets

From the definition of a CMAF WAVE Splice conditioned Switching Sets [WAVE-CON][CMAF] , the following holds for two switching sets *r* and *s*:

- All CMAF Tracks in both Switching Sets conform to one media profile.
- There exists one CMAF Header that is used to initialize the playback of both Switching Sets. This header referred as Master CMAF Header. Note that this master header may be the master header of one of the two Switching Sets or may be superset of the those.
- The CMAF Header for each track in each Switching Set is such that appending it to the source buffer does not result in a reinitialization of the decoding and rendering platform.

5.3.4 Content Model Format

In order to test and define content, this specification defines a content model format for properly referencing content. This content model format document follows the concept of a DASH Media Presentation Description (MPD) [DASH] but simplifies the description.

The first version does not yet define a content model, but an update is expected in the next version once the first test content is provided.

5.4 Scope of This Specification

5.4.1 Introduction

This specification deals with providing requirements following the description in Figure 6. Devices with capabilities defined in this specification can play back WAVE content using well-defined APIs following the model from above. Playback includes different aspects, and for each of these aspects the APIs and the units of WAVE content are used in different ways to stimulate the playback.

If a device supports certain capabilities (that can possibly be queried by an external application), then using well-defined API calls as well as WAVE content when stimulating media track buffers enables playback of media content.

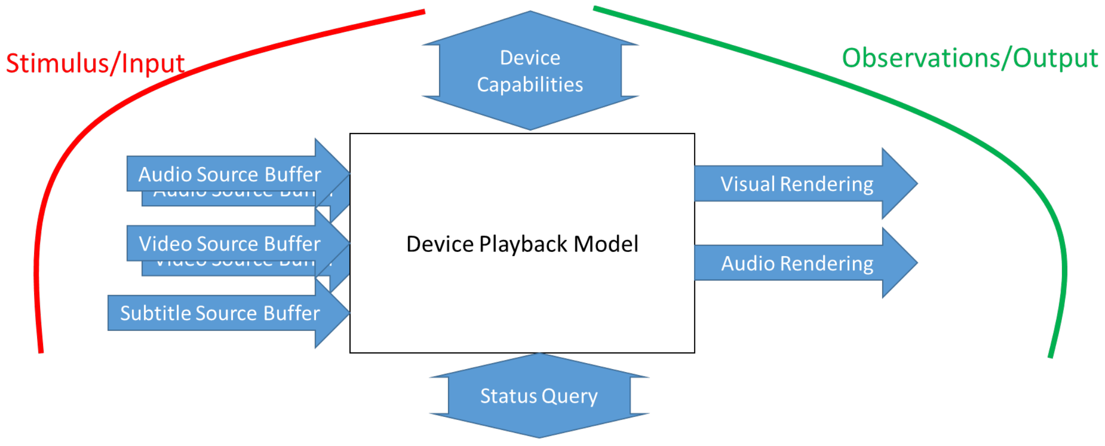


Figure 6: Device playback model

5.4.2 Conformance Aspects and Interoperability

This specification defines different conformance and interoperability points for devices.

- **Media Profile Playback**: Enables ***playback*** of a CMAF Media Profile as part of a WAVE Presentation.
- **Device Core Profiles**: Enable ***playback*** of WAVE Programs conforming to a ***WAVE Program "Profile".***
    - A full set of media profile playback capabilities that permit playback of WAVE Presentations.
    - Capability of transition across different WAVE Presentations.
    - Additional requirements such as media synchronization, etc.
- **Device Extension Profiles**: Enable ***playback*** of enhanced experiences beyond a Device Core Profile if the content provides the extension and the device supports the extensions.

    - Examples are media profiles, or, for example, seamless switching across codecs, etc.
- **Configurations:** Enable ***playback*** of a Device Core Profile or Device Extension Profile with a specific configuration by supporting at least one configuration.

The initial conformance point provided in this specification is media profile playback conformance.

It is expected that in later versions, the primary conformance points for this specification will be the device core profiles. Device core profiles collect a set of requirements, and a device conforms to a device profile if it conforms to all requirements that are associated to that device profile. A device core profile supports a full audio-visual experience to play back a WAVE Presentation Profile.

5.4.3 Tests, Performance and Performance Requirements

This specification is written such that the tests can be defined. Tests describe the usage of playback APIs and the expected resulting observations. Such observations may be documented at a high-level, but it is preferable that well-defined and measurable performance requirements are defined. Whereas an ideal performance may be desirable and should be documented, in certain case a degradation of the ideal performance may also be acceptable. In this case not only the ideal performance is documented, but also minimum performance requirements for a device.

5.4.4 Existing and New Devices

This specification is written primarily for the development and testing of new devices. Device platform manufacturers are expected to take into account the requirements when developing new devices. In this case the device maker should target to meet the device playback performance requirements as documented in this specification.

However, this specification also permits testing existing devices for their performance and to document the performance and suitability for WAVE content playback. In this case, even if a device does not fulfill the full performance requirements, the device may still be used for playback as long some minimum performance requirements are fulfilled.

This specification can be used for both cases.

6 Media Playback Model

6.1 Introduction

For an application to make use of the device platform for playback of a CTA WAVE content, it requires the following principle actions:

1. Establishing a Media Element that enables the control and presentation of media data.
2. Establishing a Media Source that serves as the source of media data for a Media element.
3. Checking if the device is capable of establishing a Source Buffer for playback of a specific CMAF media profile.
4. Establishing a Source Buffer that enables the playback of the WAVE through a well-defined set of APIs.
5. Using the source buffers and the media element for controlled playback.

A device should only establish a Source Buffer for a media type if the device is capable of playing back at least one of the included WAVE media profiles for each media type in the WAVE content offerings.

Once a source buffer for a media type is established, this source buffer is used for the playback of the media type.

6.2 Media Element and Source Establishment

6.2.1 General

A Media Element presents an output and control environment for WAVE content playback. The Media Source represents as a source of WAVE content that is consumed by a Media element. Source Buffers are added for specific media playback. A Media element is established for playback of a WAVE Presentation and a Media Source is attached to it to dynamically add WAVE content. The media element permits playback control functions such as playback, pause, mute, etc.

6.2.2 Web Media API-based Media Element and Source Establishment

The video element is a media element whose media data is ostensibly video data, possibly with associated audio data. The src, preload, autoplay, loop, muted, and controls attributes are the attributes common to all media elements. For details see https://www.w3.org/TR/html51/semantics-embedded-content.html#the-video-element.

The Media Source Extensions [MEDIA-SOURCE] Specification defines the MediaSource object. The MediaSource object represents a source of media data for an HTMLMediaElement. It also includes a list of SourceBuffer objects to add media data to the presentation. MediaSource objects are created by the web application and then attached to an HTMLMediaElement.

- For details, see https://www.w3.org/TR/media-source/#mediasource

6.3 Media Element and Media Source Control

6.3.1 General

For a media element, at least the following functions are expected to be available:

- Play: Initiates playback of the media added to the media element from the start of the buffer.
- Seek: Seeks to a media time of the media included in the buffer.
- Pause: Pauses the playback of the media in the buffer.

For a media element, it is expected that an application can at least observe the following information

- buffered: Provides the buffered media time ranges of the media in the media source.
- currentTime: Indicates the current playback media time.

6.3.2 Web Media API-based Media Element and Media Source Control

In the Web Media API [WAVE-WMA**Error! Reference source not found.**] context, a subset of properties and methods are available. For convenience and completeness of this spec, this is document in Annex B.2.

6.4 Device Capability

6.4.1 General

A first requirement is the establishment of a source buffer for each relevant media type of the WAVE Presentation. In order to establish such a source buffer, two aspects are relevant:

- Support of the generic baseline CMAF content format.
- Support of a specific media profile.

In the context of this specification it is assumed that a device supports playback of the CMAF content format.

In addition, WAVE content may be offered with different options that an application can choose from based on its device capabilities. The WAVE content specification defines multiple media profiles for each media type. Hence, an application should determine the device capabilities and based on this select the proper content options. Device capability discovery is an essential feature for an Internet media-based ecosystem.

However, based on the discussion in Annex A, this specification does not mandate a specific device capability discovery mechanism in this initial version of the specification. Device capabilities are inconsistently implemented today in devices, so multiple approaches are discussed, and it is expected that implementations will evolve over time. However, at this stage an application should be prepared to run multiple steps for capability discovery.

Based on the discussions in Annex A, it is recommended that new devices support the Media Profile capability discovery as defined in Annex A.2.2. However, devices may not implement the handling of the profile MIME subparameter and/or the details of the CMAF media profiles, and hence the response of the device for such an API may be `unknown`. In particular, existing devices may be not supporting such an API at all.

If a media profile wants to enable playback on devices that do not support the media capability API, then it is recommended that the media profile documents other means for capability detection for such a media profile, in particular using one of the following:

- MIME Subparameters as documented in Annex A.2.3.
- Media Capability approach as documented in Annex A.2.4.

Each media profile should provide sufficient information on how to use APIs for capability discovery in order to ensure the playback of the media profile following the requirements in this specification. Specifically, suitable capability discovery for existing devices is recommended to be added.

Media Profiles document how to use the Media Capability API with their media profile to identify options and parameters in their profile.

If none of the above methods are applicable or sufficiently conclusive, it is recommended that a source/track buffer with the media type is established as defined in clause 6.5, and the successful establishment is used as an indication for successful playback.

6.4.2 Web Media API-based Capability Discovery

If a media source is to be created, the `MediaSource.isTypeSupported(type)` method may be used.

- For details, see: https://www.w3.org/TR/media-source/#dom-mediasource-istypesupported.
- The method is expected to return "`false`" or "`true`".

In the context of this specification, the MIME-types for this specification must conform to the rules outlined for "audio/mp4" and "video/mp4" in RFC 6381, i.e. the ISO BMFF Byte Stream format is used as defined in [MSE-FORMAT-ISOBMFF].

The proper `type` for each media profile is defined in the WAVE content specification [WAVE-CON]. If the device responds to this method with `true`, the device claims to support playback of a specific WAVE media profile, then this specification defines the requirements that need to be fulfilled.

As an alternative to the `isSupportedType` method, a CMAF Header may be appended once the Source Buffer is established (see clause 6.5 for details) and then the source buffer monitored for any error.

6.5 Source Buffer Management

6.5.1 General

To make use of the platform for media playback, the source buffer needs to be added and initialized properly.

A WAVE device shall support an API such that the application can create a new source buffer for a specific media type and add the buffer to the media source buffer list.

A device compliant to WAVE shall allow to establish at least one source buffer for media type `video` and one for media type `audio`. A WAVE device should support the establishment of a source buffer for the playback for `subtitles`. A WAVE device may support the establishment of multiple source buffers for each media type.

A WAVE device shall support an API to establish a source buffer for CMAF content playback.

A WAVE device shall support an API such that the application can remove a source buffer from the media source buffer list.

For all media types, a WAVE device shall support an API such that the application can update the configuration of the source buffer. It is recommended that the source buffer is only updated if the update follows the requirements documented in the remainder of this specification. Otherwise, it is recommended to remove the source buffer and re-establish a new one with the new configuration.

A source buffer is expected to be established for each media type individually by:

1) Adding a source buffer providing the media type, possibly augmented with MIME sub - parameters.
2) If 1 is successful, initialize the source buffer with an appropriate CMAF Master header.
3) Create a proper output environment for each established source buffer.
    a. For video a pre-determined display window that matches the aspect ratio and either:
        i. default to the size of the content (height and width) of the CMAF Master Header is established,
        ii. or as an option a fullscreen mode may be used as well.
    b. For audio, no specific provisioning is done unless the media profile defines a specific output environment.

In addition, successful establishment of the source buffer also implies that the set of media APIs documented in clause 6.6 are supported and can be used by the application for media playback.

6.5.2 Web Media API-based Source Buffer Management

The Media Specification defines the `MediaSource.addSourceBuffer(type)` method.

- For details, see https://www.w3.org/TR/media-source/#dom-mediasource-addsourcebuffer.
- The method returns a source buffer.

The type shall follow the requirements in ISO BMFF Byte Stream Format [MSE-FORMAT-ISOBMFF], clause 2.

The source buffer is further initialized by appending a CMAF header (CH) to the `SourceBuffer` by using the `MediaSource.appendBuffer(CH)`. For a Switching Set, the CMAF Master Header shall be used.

The source buffer may be updated/re-initialized by appending a CMAF header (CH) to the `SourceBuffer` by using the `MediaSource.appendBuffer(CH)`.

The source buffer is further managed by the `MediaSource.removeSourceBuffer(type)` method.

- For details, see https://www.w3.org/TR/media-source/#dom-mediasource-removesourcebuffer.
- The method does not return any value.

The display window is established by using width and height from the CMAF Header.

6.6 Device Playback Model for a Single Source Buffer

6.6.1 Introduction

In this clause, a set of media playback APIs are introduced that permit the playback of WAVE content using these APIs. This specification does not provide requirements for the functional availability of these media APIs but assumes that these functions are available. For instance, these media APIs can be realized with W3C Media Source Extensions [MEDIA-SOURCE] as listed in clause 6.2.2.

Furthermore, it is assumed that a source buffer is established with some configuration parameters according to the requirements in clause 6.5. In addition, the capability discovery in clause 6.4 resulted in an acknowledgement that the WAVE content collected in the media profile can be played back.

Functions and properties described in the following are:

- Methods and parameters to support the playback.
- Observations that permit to query the state of the playback.

These functions and methods may be general or may be specific for specific media types. A simplified playback model for a single media type is provided in Figure 7.

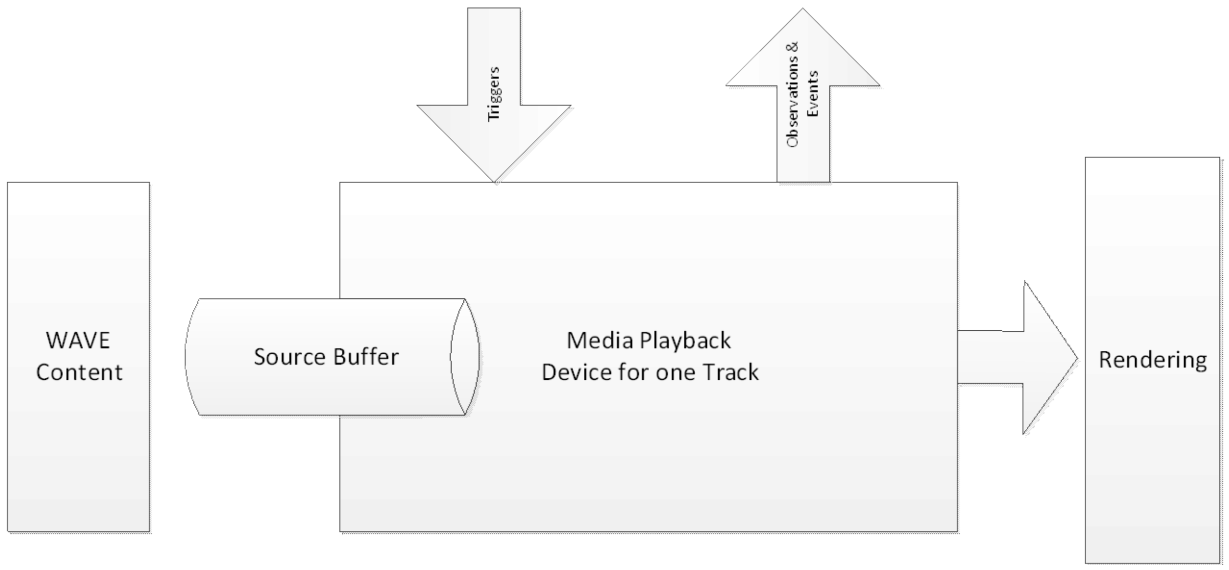


Figure 7: Simplified playback model

6.6.2 General

This clause outlines the source buffer model for this specification. It describes the various rules and behaviors associated with appending data to an individual SourceBuffer. At the highest level, the web application creates SourceBuffer objects and appends sequences of CMAF data to update their state. The media element pulls media data out of the MediaSource object, plays it, and fires events. The application is expected to monitor these events, for example to determine when it needs to append more CMAF media data.

The SourceBuffer is expected to basically implement algorithms aligned to those defined in [MEDIA-SOURCE], clause 3.5. It also inherits the methods from the Media Element or Media Source as defined in clause 6.7.

The key method is the `appendBuffer` functionality. In the context of CTA WAVE specification, this method has the following parameters:

- `data`: The appended data is a CTA WAVE addressable resource. The appended data is added to the buffer by including the data in the timeline and possibly overwriting existing data in the buffer.
- `offset`: Optional parameter to signal the offset that needs to be applied to the presentation time in order to map it to the media timeline. If not provided, the parameter is assumed to be 0.

Once this method is invoked, but the buffer is unable to accept data, the device is expected to run proper methods. For example:

- Append Error Algorithm as defined in [MEDIA-SOURCE], clause 3.5.3.
- Coded Frame Eviction Algorithm as defined in [MEDIA-SOURCE], clause 3.5.10.

If invoked and the buffer is able to accept data, the underlying platform is expected to run the following steps to process the appended data (note that this is a high-level description on what is defined in [MSE]):

- The appended data `data` is parsed.
    - If invalid data not conforming to CTA WAVE content is found, an appropriate error algorithm is run and an event is fired. The appended data `data` is ignored and not appended.
    - Any bytes that the byte stream format specifications require to be ignored are removed from the start of the input buffer. The byte stream specification format for CMAF data follows the mp4 ISO BMFF Byte Stream Format [MSE-FORMAT-ISOBMFF].
    - If the parsed data `data` is a CMAF Header:
        - Run the CMAF Header Append received function.s
    - If the parsed data `data` is a CMAF Chunk or CMAF Fragment:
        - Apply the offset.
        - Add the data to the buffer.
    - If the parsed data `data` is any other part of a CMAF Fragment:
        - Apply the offset.
        - Add the data to the buffer.

The CMAF Header append function:

- If the CMAF Header does not conform to the CMAF Header, ignore the data and stop.
- If the CMAF Header has a non-recognized media profile, ignore, fire an event and stop.
- If the CMAF Header contains a duration, set or update the duration of the media.
- If this is not the first CMAF Header that is added and processed (always the case as the SourceBuffer Establishment adds CMAF Master Header), check the following:
    - Is the same media type received?
    - Does the codecs and all other parameters announced in the updated CMAF Header match was specified in the CMAF Master Header?
    - If not, create an event that informs application about an incompatible header.
- If all successful, add track descriptions to track buffers:
    - Track description is byte stream specific.
- Set the need for a random-access point for the media.

CMAF Media Append function:

- If the Source Buffer is not initialized with a CMAF Header (should never be the case as the SourceBuffer Establishment adds header), then ignore the data.
- If the appended data `data` contains one or more coded frames/samples, then add each sample to the track buffer as follows
    - Let `dts`, `pts` and `dur` be the decoding time stamp, presentation time stamp and duration of the sample.
    - Adjust to `pts` and to `dts` with the `offset` value.
    - If random access is needed and sample is no random access point, drop frame
    - If an overlapped frame is detected.
        - Run overlap algorithm, depending on media type.
    - Remove existing coded frames from track buffer with non-matching presentation time
    - Remove all frames that depend on any frames removed in the previous two steps.
    - Add coded frame/sample with pts, dts and dur to track buffer.
    - Set highest time stamp for track and also set group end timestamp.
- Extend the duration of the track buffer.

The append function allows *set the timestamp offset* to control the offset applied to timestamps inside media segments that are subsequently appended to the SourceBuffer.

6.6.3 Web Media API-based Playback

This clause documents the methods and functions of the media source. The details will be added in a future version of the specification.

NOTE: It is expected that the development of the test environment will support this documentation.

6.7 Device Playback Model for a Media Element

6.7.1 General

For each media type contained in the WAVE content, a Source Buffer is established.

Each source buffer is then consuming the media of the corresponding type.

6.7.2 Web Media API-based Playback

This clause documents the methods and functions of the media source. The details will be added in a future version of the specification.

NOTE: It is expected that the development of the test environment will support this documentation.

7 DRM Protected Media

7.1 Introduction

Content security is accomplished in a device by two functions:

1. Key management by a digital rights management system (DRM), which authenticates a user or their device and authorizes decryption and playback of a Track on that device under control of the DRM client and under specific conditions – e.g. output protection, rental period, hardware root of trust, etc.
2. Decryption of a Track encrypted with Common Encryption using either ‘cenc’ or ‘cbcs’ encryption scheme.

Each DRM system has a proprietary trust infrastructure for DRM client private and public key pairs and encrypted media keys. DRM systems normally use a proprietary license format that contains encrypted keys and playback conditions specified in a rights expression language. In the case of FairPlay, the key server only passes an encrypted key and must rely on server authorization rules (such as checking a valid rental period) and built-in device functionality (such as requiring output protection). These systems cryptographically bind authorization to a specific device and content (with one or more Tracks encrypted with that media key), which prevents a license from being used with unintended content or on unintended devices.

The operation, implementation, and robustness of each DRM system is considered out of scope for WAVE. DRM systems are assumed to be actively managed and tested to maintain content security by testing client implementation security, identifying and repairing breaches, revoking compromised keys, denying keys to compromised devices, securely storing licenses, maintaining a secure processing environment and memory, integrating with security hardware to protect keys and decrypted media samples, etc. A WAVE device can select one or more DRM systems to implement based on commercial factors such as DRM features, robustness, availability, popularity, provider support, cost, content publisher requirements, etc.

What is in scope for WAVE is correct parsing and decryption of Tracks encrypted with ‘cenc’ or ‘cbcs’ encryption schemes specified in ISO/IEC 23001-7 Common Encryption, and the use of Encrypted Media Extensions (EME) APIs to request and download a device supported DRM license or key to test decryption, decoding, and rendering of protected content. Different DRM systems support different functions, such as persistent licenses, hierarchical licenses, license rotation, key rotation, multiple keys per license, sample variants, stream counting, secure stop, higher security for UHD content, etc. The primary focus of WAVE testing is the basic functions common to all DRM systems with the assumption that DRM-specific tests can be provided by each DRM system.

Note that the W3C EME specification also defines “Clear Key” decryption in browsers using Common Encryption, which is useful for authorization but not content protection. If content is encrypted and a Clear Key offered, a device can request authorization and download of that key, and the authorization server can possibly identify who made the request, how many authorizations, where or when they were requested, what types of devices, etc. Because Clear Keys and decrypted content are exposed in main memory, both content and keys can be redistributed to any other player or browser for unauthorized playback, so Clear Key does not provide content protection. Because Clear Key exposes keys, it must not be used with a protected key used by one or more DRM systems.

7.2 Media Profiles and Encryption Schemes

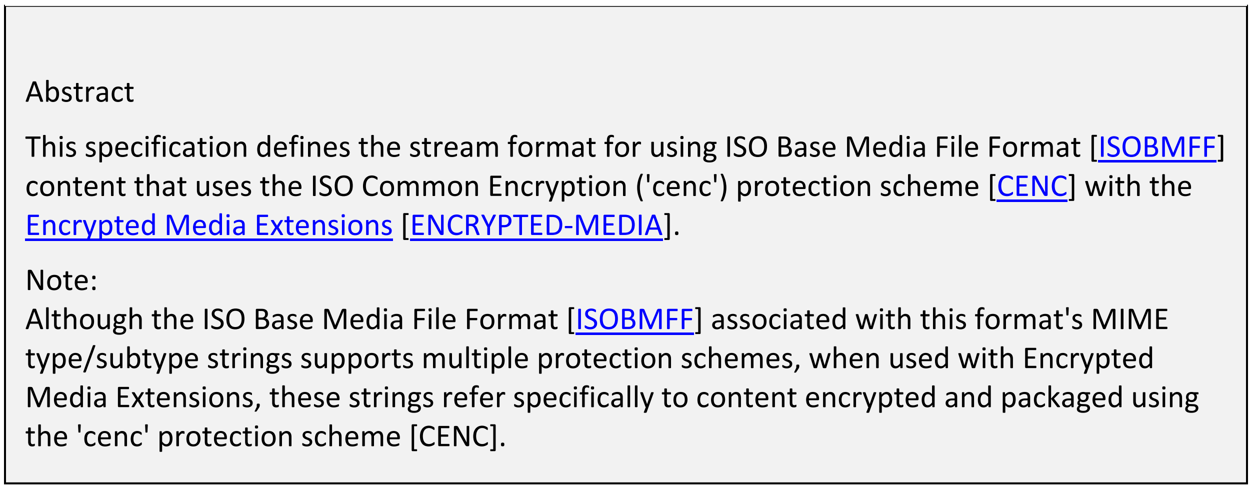
7.2.1 Introduction

The WAVE Content specification allows Common Encryption using either the ‘cenc’ or ‘cbcs’ scheme. Both schemes specify partial sample encryption for NAL structured video, and full sample encryption for other media types. In the case of ‘cbcs’, “pattern encryption” is used, which only encrypts 10% of the cipher blocks in a protected byte range using a ten block pattern. “Full sample encryption” in the case of ‘cbcs’ means the entire sample is protected, not that all blocks in the pattern are encrypted.

Common Encryption can be applied to any Media Profile using the appropriate sample format. Partial sample encryption only encrypts the video data contained in specific NAL units so parsing, display management using header information, NAL manipulation, etc. can be processed with standard software and CPU before video is passed to secure decryption, decoding, and rendering that prevents access to the decrypted portion of the samples. All other Media Profiles use full sample encryption, unless a subsample encryption mapping, like the mapping to NAL structured video, is specified.

The media and encryption formats supported by W3C Encrypted Media Extensions are specified here: https://www.w3.org/TR/eme-stream-registry/ and https://www.w3.org/TR/eme-stream-mp4/.

Figure 8: EME ISOBMFF Stream Format Spec



Abstract

This specification defines the stream format for using ISO Base Media File Format [ISOBMFF] content that uses the ISO Common Encryption ('cenc') protection scheme [CENC] with the Encrypted Media Extensions [ENCRYPTED-MEDIA].

Note:
Although the ISO Base Media File Format [ISOBMFF] associated with this format's MIME type/subtype strings supports multiple protection schemes, when used with Encrypted Media Extensions, these strings refer specifically to content encrypted and packaged using the 'cenc' protection scheme [CENC].

NOTE: As quoted above, the W3C EME ISOBMFF stream specification only includes the ‘cenc’ scheme and it only references CENC 2nd edition.

The 3rd edition of CENC (ISO/IEC 23001-7:2016) added 'cbcs' scheme and manifest signaling. Hence the W3C EME ISO BMFF stream format needs to be updated.

To apply the existing specification for 'cbcs' scheme, replace each instance of the 4CC code 'cenc' with 'cbcs', "CTR mode" with "CBC mode", and reference the 2016 edition of CENC. Section 4 recommends including a 'pssh' box in every Header in the Common SystemID and PSSH box format, but that should be deprecated. See CENC 3rd edition for signaling DRM SystemID, encryption scheme, and default_KID in manifests rather than in each Header.

7.2.2 License Acquisition using the EME API

Overview of license acquisition process and test runner using the EME API

1. When a test playlist signals that a test may contain encrypted content, the test runner will query what key systems and configurations are supported using *requestMediaKeySystemAccess(*`DOMString` *`keySystem`*, `sequence<MediaKeySystemConfiguration>` *`supportedConfigurations`*)

   The sequence of `MediaKeySystemConfiguration` configurations are tried in order. The first element with a satisfiable configuration is used. A configuration is an audio/video content type, optionally including the 'codecs' subparameter.

2. The test runner calls *createMediaKeys()* to create a new *MediaKeys* object for *keySystem.* This object has a *sessionId*, collects events, exposes the *MediaKeyStatusMap*, which lists KIDs known to the session and the status of each key.

3. The test runner calls *createSession()* to start a *MediaKeySession* that groups all calls, messages, and events for this key. There can be multiple simultaneous sessions, e.g. for different audio and video keys, subsequent presentations, etc.

4. When a license request is signaled, the test runner calls *generateRequest(*`DOMString` *`initDataType`* `BufferSource` *`initData`*) on the key session so the CDM will format a license request from DRM *initData*, as specified in https://www.w3.org/TR/eme-initdata-registry/ and https://www.w3.org/TR/eme-initdata-cenc/

   Note that initData for ISOBMFF is specified as the payload of a 'pssh' box and represented in a DASH manifest as an element containing a base64 encoded 'pssh' box and an attribute equal to the SystemID of the DRM system. Although a 'pssh' box can be included in a file header for each DRM system supported (useful for downloaded files without a manifest), for streaming it is preferable to signal *initData* in the manifest, so it can be processed once and the necessary licenses can be identified and downloaded in advance of attempting to play the encrypted media.

5. Application forwards CDM formatted license request to a license server for the selected key system, usually with an access token. A license server must encrypt a license or key with a key bound to the CDM under test, so dynamic generation and download of licenses is required during testing.

6. The downloaded license is sent to the CDM by the test runner using *update(*`BufferSource` *`response`).*
7. The runner may need to create additional sessions and license requests for additional key IDs used by other audio or video Switching Sets and start playback when all the necessary keys have been updated on the CDM.

8 Single-Track Media Playback Requirements

8.1 Introduction and Content Model

This clause documents typical functionalities required by an application in the playback using a source model as documented above for a single track.

A single CMAF track is assumed.

8.2 Sequential Track Playback

8.2.1 Background

Sequential track playback refers to the case that a CMAF/WAVE track is played from the beginning by providing CMAF fragments to the source buffer after initialization.

8.2.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1]=0`.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5, including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.2.3 Parameters and Variants

The playback has the following parameters:

- `min_buffer_duration`: expresses the minimum buffer that the Source Buffer maintains in the playback.
- `TSMax`: The maximum permitted startup delay set to 120ms.

  NOTE: This constraint is defined as a first approach but may be refined after running some initial tests.

8.2.4 Stimulus

For a track buffer that supports a media profile, Sequential Playback of a CMAF Track `k`, consisting of a sequence of CMAF Header and CMAF Fragments, refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set time offset to `tf[k,i=1]`.
- Append CMAF Fragments `CF[k,i]` in order starting from `i=1` .
- Load as many CMAF fragments `CF[k,i]` starting from fragment 1 such that the buffer duration is at least `min_buffer_duration`.
- Once reached, initiate play-back on the media source and observe:
    - The measured time when playback is initiated is `Ti`.
    - The measured time when the first sample is rendered at time `TR[k,s=1]`.
    - The measured time when sample s is rendered is time `TR[k,s]`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append next fragment `CF[k,i]`.
- Stop at the end of the last CMAF Fragment in the buffer.

8.2.5 Required Observation

8.2.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration of the playback matches the duration of the CMAF Track, i.e. TR[k, S] = TR [k, 1] + `td[k]`.
- The start-up delay should be sufficiently low, i.e., `TR [k, 1] – Ti < TSMax`.
- The presented sample matches the one reported by the currentTime value within the tolerance of the sample duration.

8.2.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.2.5.3 Audio

None.

8.2.5.4 Subtitle

None.

8.3 Random Access to Fragment

8.3.1 Background

A track is randomly accessed, and playback is started from a specific time onwards.

The random access happens at a Fragment boundary.

8.3.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1] = 0`.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.3.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: expresses the minimum buffer that the Source Buffer maintains in the playback.
- `random_access_fragment`: defines the fragment at which the track is randomly accessed.
- `TSMax`: The maximum permitted startup delay set to 120ms.
  NOTE: This constraint is defined as a first approach but may be refined after running some initial tests.

8.3.4 Stimulus

For a track buffer that supports a media profile, Random Access of a CMAF Track `k` at fragment `random_access_fragment`, consisting of a sequence of CMAF Header and CMAF Fragments refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set time offset to `tf[k,i= random_access_fragment]`.

- Append CMAF Fragments `CF[k,i]` in order starting from `i=random_access_fragment`.
- Load as many CMAF fragments `CF[k,i]` starting from fragment `random_access_fragment` such that the buffer duration is at least `min_buffer_duration`.
- Once reached, initiate play-back on the media source and observe:
    - The measured time when playback is initiated is `Ti`.
    - The measured time when the first sample is rendered at time `TR[k,s1]`.
    - The measured time when sample s is rendered is time `TR[k,s]`
- While it is not the last fragment, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append next fragment `CF[k,i]`.
- Stop the playback at the end of the last CMAF Fragment buffer.

8.3.5 Required Observation

8.3.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration of the playback matches the duration of the CMAF Track starting from the presentation time of the first sample, i.e., `TR[k, S] = TR [k, s1] + td[k] - tf[k,i= random_access_fragment]`.
- The start-up delay should be sufficiently low, i.e., `TR[k, s1] – Ti < TSMax`
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.3.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.3.5.3 Audio

None.

8.3.5.4 Subtitle

None.

8.4 Random Access to Time

8.4.1 Background

A track is randomly accessed and played back, starting from a specific time onwards.

The random access in this case may occur in the middle of any Fragment.

8.4.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1] = 0`.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.4.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `random_access_time`: Defines the presentation time at which the track is randomly accessed.
- `TSMax`: The maximum permitted startup delay set to 120ms.
  NOTE: This constraint is defined as the first approach but may be refined after running some initial tests.

8.4.4 Stimulus

For a track buffer that supports a media profile, Random Access of a CMAF Track `k` at the presentation time `random_access_time`, consisting of a sequence of CMAF Header and CMAF Fragments refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Find the Fragment number containing the presentation time `random_access_time: tk[k,r]≤ random_access_time` and `tk[k,r+1]≥ random_access_time.`
- Set time offset to `random_access_time and` set the `currentTime` on the Media Element to the random_access_time.
- Append CMAF Fragments `CF[k,i]` in order starting from `i=r`.

- Load as many CMAF fragments `CF[k,i]` starting from fragment `i` such that the buffer duration is at least `min_buffer_duration.`
- Once reached, initiate play-back on the media source and observe:
    - The measured time when playback is initiated is `Ti.`
    - The measured time when the first sample is rendered at time `TR[k,s1].`
    - The measured time when sample s is rendered is time `TR[k,s].`
- While it is not the last fragment, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append next fragment `CF[k,i].`
- Stop the playback at the end of the last Fragment in buffer.

8.4.5 Required Observation

8.4.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` with presentation time larger or equal to `random_access_time` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration of the playback matches the duration of the CMAF Track starting from the presentation time of the first sample, i.e., `TR[k, S] = TR[k, s1]` + `td[k] - random_access_time`.
- The start-up delay should be sufficiently low, i.e., `TR[k, s1] – Ti < TSMax`
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.4.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.4.5.3 Audio

None.

8.4.5.4 Subtitle

None.

8.5 Switching Set Playback

8.5.1 Background

Playback of a switching set assumes that the application can switch across the fragments of different tracks without observing any temporal or spatial misalignment during playback.

8.5.2 Pre-Conditions

A CMAF Switching Set is available for playback following the properties in clause 5.3.2.4 with a total of `K` tracks in the Switching Set.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Master Header for the Switching Set.
2) Establishing a proper output environment.

8.5.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `playout[i]`: Provides the CMAF track number for every fragment position `i=1,…,N`. The value shall be between 1 and `K`.
- `TSMax`: The maximum permitted startup delay set to 120ms.
  NOTE: This constraint is defined as a first approach but may be refined after running some initial tests.

8.5.4 Stimulus

For a track buffer that supports a media profile, playback of a Switching Set of a CMAF Switching Set, consisting of `K` tracks, the following applies:

- Set the `LastHeader` to the CMAF Master Header `CH*` used in initialization
- Set presentation time offset to `tf[k=playout[i=1],i=1]`.
- Load as many CMAF fragments `CF[k,i]` starting from fragment 1 such that the buffer duration is at least `min_buffer_duration` using the Append-Algorithm-1, outlined below.
- Once reached the `min_buffer_duration`, initiate play-back on the media source and observe:
  - The measured time when playback is initiated is `Ti`.

    - The measured time when the first sample is rendered at time `TR[s=1]`.
    - The measured time when sample `s` is rendered is time `TR[s]`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is equal to `min_buffer_duration` or below, append next fragment `CF[k,i]` using Append-Algorithm-1, outlined below.
- Stop at the end of the last CMAF Fragment in the buffer.

Append-Algorithm-1:

- For each CMAF Fragment position `i=1,…,N`.
    - If CMAF Header `CH[k = playout[i]]!= LastHeader`.
        - Append the CMAF Header `CH[k=playout[i]]` to the Source Buffer
        - Set `LastHeader` to `CH[k=playout[i]]`.
    - Append CMAF Fragment `CF[k=playout[i],i]`.

8.5.5 Required Observation

8.5.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The duration of the playback matches the duration of the CMAF Track, i.e., TR[k, S] = TR [k, 1] + `td[k]`.
- The start-up delay should be sufficiently low, i.e., `TR[k, 1] – Ti < TSMax`
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.5.5.2 Video

In addition, for video the following is expected to be observed:

- The rendering for each track is scaled to the height and width of the predetermined window.
- No visible shifts of objects in the video.
- No visible spatial offset of pixels in the video.

Based on this, if the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.5.5.3 Audio

In addition, for audio the following is expected to be observed:

- The audio plays with no glitches, clicks or dropouts.

8.6 Regular Playback of Chunked Content

8.6.1 Background

Sequential Chunked Playback refers to the case that a CMAF/WAVE track is played from the beginning by providing CMAF chunks to the source buffer after initialization.

8.6.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1]=0`.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.6.3 Parameters and Variants

The playback has the following parameters:

- `min_buffer_duration`: expresses the minimum buffer that the Source Buffer maintains in the playback. This value shall be smaller than `df[k,i]` of all Fragments.

8.6.4 Stimulus

For a track buffer that supports a media profile, Sequential Chunked Playback of a CMAF Track `k`, consisting of a sequence of CMAF Header and CMAF Chunks refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set time offset to `tf[k,i=1]`.
- Append CMAF Chunk `CC[k,i,j]` in order starting from `i=1`, and `j=1`, incrementing `j` first to the end and then incrementing `i` and resetting `j=1`, and so on.

- Load as many CMAF Chunks `CC[k,I,j]` starting from the first Chunk of the track such that the buffer duration is at least `min_buffer_duration` and not larger than or equal to `df[k,i=1]`.
- Once reached, initiate playback on the media source and observe:
    - The measured time when playback is initiated is `Ti`.
    - The measured time when the first sample is rendered at time `TR[k,s=1]`.
    - The measured time when sample `s` is rendered is time `TR[k,s]`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append next Chunk `CC[k,i,j]`.
- Stop at the end of the last Chunk of track in the buffer .

8.6.5 Required Observation

8.6.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration matches the duration of the CMAF Track, i.e. `TR [k, S] = TR[k, 1] + td[k]`.
- The start-up delay should be sufficiently low, i.e. `TR[k, 1] – Ti < TSMax`.
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.6.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.6.5.3 Audio

None.

8.6.5.4 Subtitle

None.

8.7 Regular Playback of Chunked Content, non-aligned append

8.7.1 Background

Sequential Chunked Playback refers to the case that a CMAF/WAVE track is played from the beginning by providing CMAF chunks to the source buffer after initialization. Non-aligned refers to the situation where the data that is appended to the source buffer is appended progressively in pieces that do not necessarily align with the boundaries of the CMAF chunks.

8.7.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1]=0.`

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.7.3 Parameters and Variants

The playback has the following parameters:

- `min_buffer_duration:` Expresses the minimum buffer that the Source Buffer maintains in the playback. This value shall be smaller than `df[k,i]` of all Fragments.
- `TSMax`: The maximum permitted startup delay set to 120ms.
  NOTE: This constraint is defined as a first approach but may be refined after running some initial tests.

8.7.4 Stimulus

For a track buffer that supports a media profile, Sequential Chunked Playback of a CMAF Track `k`, consisting of a sequence of CMAF Header and CMAF Chunks refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set time offset to `tf[k,i=1].`
- For each `k,i` concatenate the `N` CMAF chunks `CC[k,i,j]` in order from `j=1,` incrementing `j` to the end `(j=N)` to form a chunked fragment `CF[k,i].`
- For each `k,i` randomly choose `2N` positive non-zero integers `L` such that the sum of all `L[k,i,r]` equals the length in bytes of the chunked fragment `CF[k,i].`

- Split the chunked fragment `CF[k,i]` into `2N` byte ranges `BR[k,i,r]` each with length `L[k,i,r]`.
- Append byte ranges `BR[k,i,r]` in order starting from `i=1`, and `r=1`, incrementing `r` first to the end (`2N`) and then incrementing `i` and resetting `r=1`, and so on.
- Load as many byte ranges `BR[k,i,r]` starting from the first byte range of the track such that the buffer duration is at least `min_buffer_duration` and not larger than or equal to `df[k,i=1]`.
- Once reached, initiate playback on the media source and observe:
    - The measured time when playback is initiated is `Ti`.
    - The measured time when the first sample is rendered at time `TR[k,s=1]`.
    - The measured time when sample `s` is rendered is time `TR[k,s]`.
- While it is not the last byte range, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append next byte range `BR[k,i,r]`.
- Stop at the end of the last byte range of track in the buffer.

8.7.5 Required Observation

8.7.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration matches the duration of the CMAF Track, i.e. `TR[k, S] = TR[k, 1] +td[k]`.
- The start-up delay should be sufficiently low, i.e. `TR[k, 1] – Ti < TSMax`.
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.7.5.2 Video

If the track is a video track, then

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.7.5.3 Audio

None.

8.7.5.4 Subtitle

None.

8.8 Playback over WAVE Baseline Splice Constraints

8.8.1 Background

Playback over splices enables the content provider to offer a sequence of Switching Sets with less restricted encoding requirements than a track. This feature is especially important in case of program changes and ad insertion.

8.8.2 Pre-conditions

Two CMAF Switching Sets are available for playback following the properties in clause5.3.3.5.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

- Appending the CMAF Master Header for the two Switching Sets.
- Establishing a proper output environment.

8.8.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `playout[i]`: Provides the triple (Switching Set, CMAF track number, Fragment number) for every playout position `i=1,…,N` that is be played out.

8.8.4 Stimulus

For a track buffer that supports a media profile, playback of two WAVE Baseline Splice Constraints Switching Sets of a CMAF Switching Set, consisting of K tracks, the following applies:

- Set the `LastHeader` to the CMAF Master Header `CH*` used in initialization.
- Set presentation time offset to `tf[k=playout[i=1],i=1]`.
- Load as many CMAF fragments `CF[k,i]` starting from fragment 1 such that the buffer duration is at least `min_buffer_duration` using the Append-Algorithm-2, outlined below.
- Once reached the `min_buffer_duration`, initiate playback on the media source and observe:
    - o The measured time when playback is initiated is `Ti`.
    - o The measured time when the first sample is rendered at time `TR[s=1]`.

    - The measured time when sample `s` is rendered is time `TR[s]`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is equal to `min_buffer_duration` or below, append next fragment `CF[k,i]` using Append-Algorithm-2, outlined below.
- Stop at the end of the last CMAF Fragment in the buffer.

Append-Algorithm-2:

- For each CMAF Fragment position `i=1,…,N`:
    - If CMAF Header `CH[k = playout[i]]!= LastHeader`
        - Append the CMAF Header `CH[k=playout[i]]` to the Source Buffer.
        - Set `LastHeader` to `CH[k=playout[i]]`.
    - Append CMAF Fragment `CF[k=playout[i],i]`.

8.8.5 Required Observations

8.8.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration matches the duration of the CMAF Track, i.e. `TR[k, S] = TR[k, 1] +td[k]`.
- The start-up delay should be sufficiently low, i.e., `TR[k, 1] – Ti < TSMax`
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.8.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.9 Out-Of-Order Loading

8.9.1 Background

Out of order loading refers to the case that a CMAF/WAVE track is played from the beginning by providing CMAF fragments to the source buffer after initialization, but CMAF fragments are loaded out of order within the buffer duration.

8.9.2 Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4 with the earliest presentation time `tf[k,i=1]=0`.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5 including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.9.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `max_buffer_duration`: Expresses the duration of media that can always be accommodated in the Source Buffer.
- `loading[i]`: Provides the CMAF Fragment number that is loaded at step `i`, constrained such that `(MAX(i-loading[i]) + MAX(loading[i]-i)) * MAX(df[k,i]) < max_buffer_duration`.

The parameters need be such that in any case the loaded CMAF Fragment needs to be in the playout buffer.

8.9.4 Stimulus

For a track buffer that supports a media profile, Sequential Playback of a CMAF Track `k`, consisting of a sequence of CMAF Header and CMAF Fragments refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set presentation time offset to `tf[k,i=1]`.
- Append CMAF Fragments `CF[k,loading[i]]` in order starting from `i=1`.

- Load as many CMAF fragments `CF[k,loading[i]]` starting from index 1 such that the contiguous buffered duration is at least `min_buffer_duration`.
- Once reached, initiate play-back on the media source and observe:
    - The measured time when playback is initiated is `Ti`.
    - The measured time when the first sample is rendered at time `TR[k,s=1]`.
    - The measured time when sample `s` is rendered is time `TR[k,s]`.
- While it is not the last fragment, do:
    - As soon as the contiguous buffered duration is `min_buffer_duration` or below, append next fragment `CF[k,loading[i]]`.
- Stop at the end of the last CMAF Fragment in the buffer.

8.9.5 Required Observation

8.9.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration of the playback matches the duration of the CMAF Track, i.e., `TR[k, S] = TR[k, 1] + td[k]`.
- The start-up delay should be sufficiently low, i.e., `TR[k, 1] – Ti < TSMax`.
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.9.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.9.5.3 Audio

None.

8.9.5.4 Subtitle

None.

8.10 Overlapping Fragments

8.10.1 Background

Playback of overlapping fragments assumes that the application can overwrite the buffer with other fragments. This is for example useful in case that a fragment is loaded in lower quality but may then be replaced by a better-quality fragment.

8.10.2 Pre-condition

A CMAF Switching Set is available for playback following the properties in clause 5.3.2.5 with in total `K` tracks in the Switching Set.

A Media Source is established as defined in clause 6.2.

The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5, including the relevant output. This includes:

1) Appending the CMAF Header for the track.
2) Establishing a proper output environment.

8.10.3 Parameters and Variants

The playback has the following parameters:

- `min_buffer_duration:` Expresses the minimum buffer that the Source Buffer maintains during the playback. This value shall be equal or larger than `2*MAX(df[k,i])` for all `i` and `k`.
- `playout[i]:` Provides the CMAF track number for every fragment position `i=1,…,N`. This value shall be between `1` and `K`.

8.10.4 Stimulus

For a track buffer that supports a media profile, Overlapped Fragment Playback of a CMAF Switching Set, consisting of `K` tracks refers to the following actions, starting with `k=1` for the track with lowest quality to `k=K` with the highest quality:

- Set the `LastHeader` to the CMAF Master Header `CH*` used in initialization.
- Set presentation time offset to `tf[k=playout[i=1],i=1].`
- Load as many CMAF fragments `CF[k,i]` starting from fragment 1 such that the buffer duration is at least `min_buffer_duration` using the Append-Algorithm-3, outlined below.
- Once reached the `min_buffer_duration`, initiate play-back on the media source and observe:
    - o The measured time when playback is initiated is `Ti.`

    - The measured time when the first sample is rendered at time `TR[s=1]`.
    - The measured time when sample `s` is rendered is time `TR[s]`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is equal to `min_buffer_duration` or below, append next fragment `CF[k,i]` using Append-Algorithm-3, outlined below.
- Stop at the end of the last CMAF Fragment in the buffer.

Append-Algorithm-3:

- For each CMAF Fragment position `i=1,…,N`:
    - If CMAF Header `CH[k = playout[i]]!= LastHeader`
        - Append the CMAF Header `CH[k=playout[i]]` to the Source Buffer.
        - Set `LastHeader` to `CH[k=playout[i]]`.
    - Append CMAF Fragments `CF[k=playout[i],i-1]` and `CF[k=playout[i],i]`.

8.10.5 Required Observation

8.10.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration matches the duration of the CMAF Track, i.e. `TR[k, S] = TR[k, 1] +td[k]`.
- The start-up delay should be sufficiently low, i.e. `TR[k, 1] – Ti < TSMax`.
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.10.5.2 Video

In addition, for video the following is expected to be observed:

- The rendering for each track is scaled to the height and width of the predetermined window.
- No visible shifts of objects in the video.
- No visible spatial offset of pixels in the video.

Based on this, if the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.10.5.3 Audio

In addition, for audio the following is expected to be observed:

- The audio plays with no glitches, clicks or dropouts.

8.10.5.4 Subtitle

None.

8.11 Full Screen Playback of Switching Sets

8.11.1 Background

Fullscreen playback ensures that the device provide proper scaling and letter boxing.

8.11.2 Pre-conditions

Same as documented in clause 8.9.2.

Output is set to fullscreen.

8.11.3 Parameters and Variants

Same as documented in clause 8.9.3.

8.11.4 Stimulus

Same as documented in clause 8.9.4.

8.11.5 Required Observations

8.11.5.1 General

If the above algorithm is carried out, the following observations are expected:

- Every sample `S[k,s]` shall be rendered and the samples shall be rendered in increasing presentation time order.
- The playback duration matches the duration of the CMAF Track, i.e. `TR[k, S] = TR[k, 1] + td[k]`.
- The start-up delay should be sufficiently low, i.e. `TR[k, 1] – Ti < TSMax`
- The presented sample matches the one reported by the `currentTime` value within the tolerance of the sample duration.

8.11.5.2 Video

If the track is a video track, then:

- Every video frame `S[k,s]` shall be rendered such that it fills the entire video output window.

8.12 Playback of Encrypted Content

8.12.1 Introduction

Playback of encrypted content requires that a device under test contain a Content Decryption Module (CDM) that performs two functions:

1. Decryption of media samples encrypted with Common Encryption (CENC) scheme ‘cenc’ or ‘cbcs’, followed by normal decoding and rendering equivalent to unencrypted content.
2. Exchange of license requests and licenses containing keys between the CDM and a license server, e.g., by the test runner through the W3C Encrypted Media Extension API (EME). See 7.2.2.

Requesting and downloading a license is a function of a player or test runner and a license server, not the Device under test. The test runner and license server are not being tested but must function correctly to securely acquire keys that are cryptographically bound to the CDM for the purpose of testing device decryption.

The security and functionality of the DRM system is considered out of the scope of WAVE tests, other than to enable media decoding and rendering of encrypted content test cases. It is assumed that each CDM is fully specified and implemented according to the requirements of the DRM provider.

8.12.2 Encrypted Content Use Cases

8.12.2.1 Introduction

The following use cases combine four methods of key management with the other media playback use cases and observations specified elsewhere in this document:

1. A single Track or Presentation with one key.
2. A single Presentation with different audio and video keys.
3. Multiple Presentations sequenced in a Program, encrypted and clear. One license, but interspersed clear content, e.g. an encrypted show interspersed with unencrypted ads.
4. Multiple Presentations sequenced in a Program that require different licenses (“license rotation”). Licenses must be downloaded and updated without interrupting playback.

The Playlist should include a license tags prior to the first dependent segment tag to simulate just in time license processing.

Playback Observation: Playback observations SHALL equal the equivalent unencrypted test case, except where DRM security requirements prohibit normal playback.

For instance, playback of UHD or HD content could be blocked or subsampled to lower resolution on analog video interfaces or digital interfaces with too low an HDCP version number. That is correct playback behavior under DRM control, given a device in a particular system configuration with those license restrictions.

Tests, such as trick play, are valid for testing parser and CDM handling of encryption and key changes out of sequence, but should test media pipeline functions, not the speed of license acquisition from the test license server. The test runner will not respond to unidentified keys found in content (by handling fired “need key” events and initiating license downloads), but instead test playlists SHALL include tags instructing the test runner to download the necessary keys in advance of their being needed for decryption.

WAVE tests are not defined at this time for “key rotation” and hierarchical licenses, which can be stored in ‘pssh’ boxes in Fragments. Key rotation allows keys to change over time without requiring a CDM-unique out of band license downloaded for each key change. Hierarchical licenses also enable authorizing a channel, subscription, etc. with a single parent license that is unique to each CDM but authorizes child licenses that are different for each piece of content but readable by all CDMs, so can be broadcast in the content. These are currently considered functions and tests covered by each DRM system.

8.12.2.2 Signaling Requirements and Capability Selection

A test playlist SHALL identify the encryption format applied and one or more DRM systems, DRM initialization data, and license server locations able to provide the necessary decryption keys.

The playlist encryption information SHALL match the default_KID and Common Encryption scheme found in the Track Encryption Box (‘tenc’) in the CMAF Headers that follow it. If different keys are used in different Switching Sets, i.e. different keys for audio and video, or SD, HD, and UHD video, then the playlist SHALL identify the default_KID needed prior to the first instance of appending a segment from that Switching Sets. Playlist tags are sequentially processed, so the location of a license tag determines the relative timing of the license request relative to a Header or segment request.

The test runner uses playlist tags to identify the encryption scheme used and DRM licenses offered so the test runner can determine if a device can decrypt the signaled decryption scheme (‘cenc’ or ‘cbcs’) and supports one of the listed DRM systems (e.g. Widevine, FairPlay, or PlayReady). The test runner is expected to automatically query device DRM system capability, and request licenses in a device supported DRM system. See Section 7, DRM Protected Media.

The test runner is only expected to respond to tags in the playlist, not Header information such as the sample entry or ‘tenc’ box, or Fragment information such as sample groups and their descriptions. Headers, sample groups, and sample auxiliary information are used by the device’s media pipeline to determine what scheme to decrypt with what keys and byte ranges when fully parsing and processing the media. But the test runner only needs to parse the playlist to fetch the correct license before appending dependent media segments.

The test runner and license server implement an authorization protocol for WAVE testing using an access token or other means to directly authenticate the test runner and authorize license downloads. The license server(s) maintain a secure key database indexed by KID so license requests need not convey the keys in the license requests (typically done in real world applications using JSON web tokens attached by Auth servers forwarding the license request to the license server).

8.12.2.3 Playback Requirements

Devices SHALL decrypt at least one CENC scheme (‘cenc’ or ‘cbcs’) using at least one DRM system.

Devices SHOULD support decryption and DRM capability discovery through the EME API or similar native API.

Devices SHOULD indicate which encryption schemes and DRM systems are supported for each Media Profile in the case where only some combinations of Media Profile, decryption scheme, and DRM system are supported in combination. See 7.2.2.

8.12.3 Test Pre-condition

A CMAF Track is available for playback following the properties in clause 5.3.2.4. The track is encrypted with Common Encryption `'cenc'` or `'cbcs'` encryption scheme and a key identified by the key identifier `default_KID`. There is a license available for the track and a license server. The device needs to include a CDM and APIs that a test runner can use to negotiate a license with the license server as defined in clause 7.2.2.

A capability check for the specific encryption scheme is done and was successful, as defined in clause 7.2.2.

A license suitable for the supported device capabilities is selected. A license is acquired and available as defined in clause 7.2.2 before the Media Source is established.

A Media Source is established as defined in clause 6.2.

The media capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.

A Source Buffer is created as defined in clause 6.5.

8.12.4 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: expresses the minimum buffer that the Source Buffer maintains in the playback.
- Encryption scheme ('cenc'|'cbcs') and default_KID.
- One or more DRM SystemID and InitData tags and a listed or known license server URL.

8.12.5 Stimulus

After the necessary licenses are acquired and the secure media pipeline initialized:

For a track buffer that supports a media profile, Sequential Playback of a CMAF Track *k*, consisting of a sequence of CMAF Header and CMAF Fragments refers to the following actions:

- Append the CMAF Header `CH[k]` to the Source Buffer.
- Set presentation time offset to 0.
- Append CMAF Fragments `CH[i]` in order starting from `i=1`.
- Load as many fragments starting from fragment 1 such that the buffer is at least `min_buffer_duration`.
- Start-play time `T1`.
    - The first sample is rendered at time `T2`.
- While it is not the last fragment, do:
    - As soon as the `buffer` is `min_buffer_duration` or below, append the next fragment.
- Stop at the end of the last fragment in the buffer.

8.12.6 Expected Observation

The playback is continuous and time-accurate.

No unnecessary startup delay, e.g., initializing a secure video path, negotiating a secure HDCP display connection, etc.

Every sample shall be rendered and shall be rendered in order.

The playlist plays until the end.

The duration of playback matches the duration of the CMAF Presentation, i.e. Last frame is presented at T2 + duration of the CMAF Track.

The rendering time report on the API matches the rendered sample.

8.13 Restricted Splicing of Encrypted Content

8.13.1 Conditions

Unencrypted samples and key changes SHALL be signaled by sample groups and sample group descriptions in each Fragment that contain keys or encryption state different from the default signaled in the ‘tenc’ box of the Master Header.

A Master Header for each Switching Set SHALL cause initialization of a media pipeline sufficient to decrypt, decode, and display all Fragments that follow.

Multiple Track Switching Sets SHALL conform to CMAF single initialization constraints.

The test playlist SHALL request licenses with the necessary keys in advance.

Time discontinuities SHALL be signaled by a discontinuity tag, and the presentation time offset set to the ‘tfdt’ `BaseMediaDecodeTime`.

8.13.2 Stimulus and Observation

A test playlist SHALL first initialize a secure media pipeline for each media component, i.e. one that includes a CDM and protected path in response to initializing decryption as defined in clause 7.2.2.

One set of licenses based on the default_KID in each Master Header SHALL be downloaded and SHALL be sufficient to decrypt the duration of the Program without additional license downloads.

Common Encryption ‘seig’ sample groups SHALL be processed without interruption of the secure media pipeline.

Playback Observation: The same as 8.12.

> NOTE: Initialization and license downloading rely on the playlist and test runner. Signaling of the encryption scheme, encryption options, subsample byte ranges, default KID, sample group KID, initialization vectors, etc. are all signaled by Common Encryption information in ISOBMFF boxes that enable decryption by a CDM and secure media pipeline without reliance on the playlist or test runner (other than appending Headers at splices where encryption changes).

8.14 Sequential Playback of Encrypted and Non-Encrypted Baseline Content

8.14.1 Conditions

Test vectors SHALL include a sequence of Presentations conforming to WAVE Baseline splice constraints, including both encrypted and unencrypted Presentations using either ‘cenc’ or ‘cbcs’ scheme, but not both.

A test playlist SHALL append a Header and signal presentation time offset discontinuity at each splice point between Presentations.

8.14.2 Stimulus and Observation

A test playlist SHALL first initialize a secure media pipeline for each media component, i.e. one that includes a CDM and protected path in response to initializing decryption as defined in clause 7.2.2.

Unencrypted Presentations SHALL be processed without reconfiguration of the secure media pipeline, but splices SHALL include Header appends in the test playlist, i.e. changing between encrypted and clear content as signaled by a 'tenc' box or the lack thereof, or change in the default_KID, which requires a license tag to tell the runner to fetch a different license.

Playback Observation: The same as 8.12.

Note: Initialization and license downloading rely on the playlist and test runner. Signaling of the encryption scheme, encryption options, subsample byte ranges, default KID, sample group KID, initialization vectors, etc. are all signaled by Common Encryption information in ISOBMFF boxes that enable decryption by a CDM and secure media pipeline without reliance on the playlist or test runner (other than appending Headers at splices where encryption changes).

8.15 Source Buffer Re-Initialization

This test provides a stimulus in case that a source buffer needs to be re-established, for example because of a codec change or a codec parameter change.

The first version of this specification does not define a test for this.

8.16 Playback Other than Real Time

8.16.1 Background

This test refers to the playback of content faster or slower than real-time, or even in reverse direction. Content conforms even in different speed to the profile/level constraints of decoder.

There are three ways to do this:

- Play at X times as long as it is in the decoder profile/level constraint (app forwards all content to the device and device plays with X times).
- Use special content (e.g., I-frame only content, to address any of these issues – and use the above).
- Download key frames from regular content and feed those into the MSE source buffer.

The first version of this specification does not define a test for this.

8.17 Buffer Underrun and Recovery

8.17.1 Background

This case addresses if the buffer in play mode runs out of media and is recovering from this situation. Different options are considered:

- Stall at fragment boundary and resume with next fragment (decoding order)
- Stall at chunk boundary and resume with next chunk
- Stall/Underrun, but continue with live timing (so basically in a loss)
    - For fragment
    - For chunk

The first version of this specification does not define a test for this.

8.18 Truncated Playback and Restart

8.18.1 Background

Playback can be stopped at an arbitrary presentation time within a CMAF fragment and playback can be started with a new presentation seamlessly.

The first version of this specification does not define a test for this.

8.19 Long Duration Playback

8.19.1 Background

This requirement addresses the proper playback of a CMAF track over a longer time; in particular, no drift in the playout occurs.

The first version of this specification does not define a test for this.

8.20 Event Message Processing

CTA WAVE content may contain event messages of relevance for the application.

The first version of this specification does not define a test for this.

9 WAVE Content Playback Requirements

9.1 Introduction

This clause deals with requirements of playback of WAVE Programs.

9.2 Regular Playback of a CMAF Presentation

9.2.1 Background

This case refers to the case for which a CMAF Presentation is provided and is played back. The playback instructs the player to play multiple tracks, but at most one of each media type.

9.2.2 Pre-condition

A WAVE Presentation is available for playback following the properties in clause 5.3.3. The WAVE Presentation has at least a Switching Set of type video and one of type audio. It may also have subtitles. The earliest presentation time of the first CMAF Fragment in each track are identical and are 0. The CMAF Presentation has a duration.

A Media Source is established as defined in clause 6.2.

For each media type:

- The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.
- A Source Buffer is created as defined in clause 6.5.

9.2.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: expresses the minimum buffer that the Source Buffer maintains in the playback.

9.2.4 Stimulus

For each source buffer an independent process is run.

Start play is done on the media source/element ➔ observe `T1`.

The first rendered sample is specific to the source buffer `T2[s]` with s source buffer index.

Stop play is done on the media source/element.

9.2.5 Expected Observation

9.2.5.1 General

The playback is continuous and time-accurate.

The duration of the playback matches the duration of the CMAF Presentation.

Every sample for every media type included in the CMAF Presentation duration shall be rendered and shall be rendered in order.

The presentation starts with the earliest video sample and the audio sample that corresponds to the same presentation time.

While continuing playback, the media samples of different tracks with the same presentation times are presented jointly.

9.3 Random Access of a WAVE Presentation

9.3.1 Background

In scenarios such as live programs, the client accesses an ongoing Presentation to join in the live event. This is a typical case that media needs to be accessed and played back at a random access time.

9.3.2 Pre-condition

A WAVE Presentation is available for playback following the properties in clause 5.3.3. The WAVE Presentation has at least a Switching Set (likely only one track) of type video and one of type audio. The Presentation may also have subtitles. The earliest presentation time of the first CMAF Fragment in each track are identical and are 0. The CMAF Presentation has a duration.

A Media Source is established as defined in clause 6.2.

For each media type

- The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.
- A Source Buffer is created as defined in clause 6.5.

9.3.3 Parameters and Variants

The playback has the following parameters

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `start_index_video`: The CMAF Fragment number of the video track to start with.

9.3.4 Stimulus

For each source buffer an independent process is run.

The `td[start_index_video]` is the decode time of the first video CMAF Fragment.

The first video CMAF Fragment to be appended is `CF[start_index_video]`.

Pick the CMAF Fragment with largest index that has an earliest presentation time that is smaller than or equal to the earliest presentation time of the CMAF Video fragment `CF[start_index_video].`

Set the presentation time offset to `td[start_index_video]`?

Start play on Media Source Element.

9.3.5 Expected Observation

9.3.5.1 General

The playback is continuous and time-accurate.

The duration of the playback matches the duration of the CMAF Presentation.

Every sample for every media type included in the CMAF Presentation duration shall be rendered and shall be rendered in order.

The presentation starts with the earliest video sample and the audio sample that corresponds to the same presentation time.

While continuing playback, the media samples of different tracks with the same presentation times are presented jointly.

9.4 Splicing of WAVE Program with Baseline Constraints

9.4.1 Background

Apps want to splice content that was properly conditioned based on the WAVE Program Baseline Constraints. The expectation is that the device plays back these splices seamlessly.

9.4.2 Pre-condition

Two CMAF Presentations are available for playback following the properties in clause 5.3.3. Each CMAF Presentation has at least a Switching Set (likely only one track) of type video and one of type audio. The Presentation may also have subtitles. The earliest presentation time of the first CMAF Fragment in each track are identical and are 0. The CMAF Presentation has a duration.

The Adaptation Sets for a media type in each Presentation share a Common CMAF Header.

A Media Source is established as defined in clause 6.2.

For each media type:

- The capability discovery as defined in clause 6.4 using the parameters assigned to the track was successful.
- A Source Buffer is created as defined in clause 6.5 using the Common Header.

A Media Source is established as defined in clause 6.2.

9.4.3 Parameters and Variants

The playback has the following parameters:

- `min_buffer_duration`: Expresses the minimum buffer that the Source Buffer maintains in the playback.
- `playout[i]`: Provides the triple (Switching Set, CMAF track number, Fragment number) for every playout position `i=1,…,N` that is be played out for every media type.

9.4.4 Stimulus

For each source buffer an independent process is run.

The first video CMAF Fragment to be appended is `CF[start_index_video]`.

Pick the CMAF Fragment with largest index that has an earliest presentation that is smaller than or equal to the earliest presentation time of the CMAF Video fragment `CF[start_index_video]`.

Set the presentation time off set to `td[start_index_video]`.

Start play on Media Source Element.

9.4.5 Expected Observation

9.4.5.1 General

The playback is continuous and time-accurate.

The duration of the playback matches the duration of the CMAF Presentation.

Every sample for every media type included in the CMAF Presentation duration shall be rendered and shall be rendered in order.

The presentation starts with the earliest video sample and the audio sample that corresponds to the same presentation time.

While continuing playback, the media samples of different tracks with the same presentation times are presented jointly.

9.5 Joint Playback of Video and Subtitles

9.5.1 Background

This test specifies the joint playback of video and subtitles and the proper rendering.

No tests are defined for this feature in the first version of the specification.

10 Video Capabilities and Requirements

10.1 Introduction

The WAVE content specification [WAVE-CON] defines a set of WAVE Media Profiles for video.

Each video Media Profile has associated with it a specific video codec and a set of constraints limiting parameters such as codec profile and level, maximum resolution, maximum frame rate, colorimetry and so on.

Clause 10.2 defines the device playback requirements associated with the WAVE Media Profiles.

10.2 Media Profiles

10.2.1 General

Content that conforms to a particular WAVE Media Profile may have any resolution within the limits specified for that Media Profile.

Devices that support one or more Media Profiles that have a defined maximum resolution of 1920x1080 shall support the decoding and display of pictures with the resolutions listed below for those Media Profiles. This does not preclude the use of other resolutions in WAVE content. However, a limited number of resolutions are listed here to ease device testing.

Horizontal	Vertical
1920	1080
1600	900
1280	720
1024	576
960	540
852	480
768	432
720	404
704	396
640	360
512	288
480	270
384	216
320	180
192	108

Devices that support one or more Media Profiles that have a defined maximum resolution of 3840x2160 shall support the decoding and display of pictures with the resolutions listed below for those Media Profiles. This does not preclude the use of other resolutions in WAVE content. However, a limited number of resolutions are listed here to ease device testing.

Horizontal	**Vertical**
3840	2160
3200	1800
2560	1440
1920	1080
1600	900
1280	720
1024	576
960	540
852	480
768	432
720	404
704	396
640	360
512	288
480	270
384	216
320	180
192	108

Devices that support one or more Media Profiles that have a defined maximum frame rate of 60 Hz shall support the decoding and display of pictures with the following frame rates for those Media Profiles. This does not preclude the use of other frame rates in WAVE content. However, a limited number of frame rates are listed here to ease device testing:

- 6/1.001 Hz, 12/1.001 Hz and 24/1.001 Hz
- 6 Hz, 12 Hz, 24 Hz
- 6.25 Hz, 12.5 Hz, 25 Hz and 50 Hz
- 7.5/1.001 Hz, 15/1.001 Hz, 30/1.001 Hz and 60/1.001 Hz
- 7.5 Hz, 15 Hz, 30 Hz and 60 Hz

Devices shall support switching frame rates within each family listed above.

Devices supporting a particular Media Profile shall support all of the color coding and transfer characteristics options defined for that Media Profile.

10.2.2 Media Profile: CMAF AVC HD ('cfhd')

10.2.2.1 Introduction

The media profile CMAF AVC HD is defined in [WAVE-CON], clause 4.2 and [CMAF].

10.2.2.2 Capability Discovery Options

A device supporting this media profile shall support at least one of the following capability discovery mechanisms:

1) The “is supported type query” for the media profile with argument `video/mp4 profile="cfhd"` results in a `yes`.
2) The “is supported type query” for the codec with argument `video/mp4 codec="XXX"` or results in a `yes` with `XXX` defined in [WAVE-CON].
3) The playback can be started by a CMAF header that conforms to the media profile of this specification.

10.2.2.3 Source Buffer Initialization Requirements

If no other video source buffer is available, then the device shall support the creation of a source buffer with any of the following codecs parameters AS DEFINED IN [WAVE-CON], clause 4.2 for this profile.

10.2.2.4 Content Options

Support for this media profile implies that the device should support playback of content (as defined in clause 10.2.2.5) for all content options enabled by the profile.

Support for this media profile implies that the device shall support playback of content (as defined in clause 10.2.2.5) for all parameters suitable for this profile as defined in clause 10.2.1.

10.2.2.5 Playback Requirements

Support for this media profile implies that the device:

- Shall support the following playback requirements as documented in clause 8:
    - 8.2 Sequential Track Playback
    - 8.3 Random Access to Fragment
    - 8.4 Random Access to Time
    - 8.5 Switching Set Playback
    - 8.6 Regular Playback of Chunked Content
    - 8.7 Regular Playback of Chunked Content, non-aligned append
    - 8.8 Playback over WAVE Baseline Splice Constraints

- Should support the following playback requirements as documented in clause 8:
    - 8.9 Out-Of-Order Loading
    - 8.10 Overlapping Fragments
    - 8.11 Full Screen Playback of Switching Sets
    - 8.12 Playback of Encrypted Content
    - 8.13 Restricted Splicing of Encrypted Content

10.2.3 Conditions

-
- 8.14 Sequential Playback of Encrypted and Non-Encrypted Baseline Content

10.2.4 Media Profile: CMAF HEVC HHD10 ('chh1')

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

10.2.5 Media Profile: CMAF HEVC UHD10 ('cud1')

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

10.2.6 Media Profile: CMAF HEVC HDR10 ('chd1')

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

10.2.7 Media Profile: CMAF HEVC HLG10 ('clg1')

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

10.3 Cross-Media Profile Video Splice Playback Requirements

This version of the specification does not define any splice requirements for the playback of two non-conforming media profiles.

11 Audio Capabilities and Requirements

11.1 General

The details for audio profiles will be added to an updated version of the specification after a questionnaire to the media profile proponents.

11.2 Media Profiles

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

11.3 Cross-Media Profile Audio Splice Playback Requirements

This version of the specification does not define any splice requirements for the playback of two non-conforming media profiles.

12 Subtitle Capabilities and Requirements

12.1 Introduction

The details for this profile will be added to an updated version of the specification after a questionnaire to the media profile proponents.

13 Other Device Playback Requirements

This version of the specification does not define any additional playback requirements beyond media profile playback. In the future this may for example define requirements on supporting certain graphics overlays, protocol versions of HTTP, etc.

14 Device Core Profiles

14.1 Introduction

A device core profile defines a set of playback and other requirements that must be supported by a device to claim conformance against a device core profile.

This first version of the specification does not define a device core profile.

15 Device Extension Profiles

15.1 Introduction

A device extension profile defines a set of playback and other requirements that must be supported by a device to claim conformance against a device extension profile.

A device extension profile assumes that at least one device core profile is supported.

This first version of the specification does not define a device core profile.

16 Configurations

16.1 Introduction

Configurations are requirements for devices. A device shall support at least one of any of the defined configurations.

16.2 Encryption

16.2.1 Configuration Options

A WAVE device shall support at least one of the two encryption modes: “cenc” or “cbcs”.

A WAVE device should support both encryption modes: “cenc” and “cbcs”.

16.2.2 Capability Discovery

Capability discovery for encryption configuration is for further study.

16.2.3 Playback Requirements

Please refer to clause 7.

Annex A: Device Capability Discovery (Informative)

A.1. General

The application needs to determine if it can playback the offered content following the requirements of this specification.

It is important to assume that the content is properly labelled (through media profile identifier and other indicators) and formatted according to the controlling specifications, primarily the WAVE content specification [WAVE-CON].

Labeling of the content may for example be done through one or more of the following means:

- A WAVE content signaling as defined in the content specification.
- The Internet media type of the defined in RFC6381 including the profile and codecs parameter
- Signaling of the content in the manifest, for example in the DASH-MPD using Adaptation Set signaling, predominantly the `@mimeType` and `@codecs` parameter.
- The signaling in the CMAF Header, specifically:
    - the compatibility brands in the ftyp box, and
    - The information of the sample description box in the CMAF Header.

Note that the information may be duplicated on different levels.

For each media profile, the signaling requirements are provided in the WAVE content specification.

This specification provides means on how to use content signaling for capability discovery.

Some options:

- `HTMLMediaElement.canPlayType` to determine if a `mimetype/codec` is supported.
- `SourceBuffer` can be created to handle the media type we are interested in, so we should be using `MediaSource.isTypeSupported`.

A.2. Capability Discovery Options (not about signaling)

This section discusses options that were considered during the development of this specification that may or may not be available or may be available as a proprietary solution or arrangement.

A.2.1. Media Profile

The application uses the media profile for capability discovery. The media profile may for example be provided in the manifest or the CMAF Header in the ftyp box. The application queries the device of the media profile using the `isSupportedType()` API if it can be played back using:

- `<mediatype>/mp4 profile="<media profile 4CC>"`

The device may provide one of the following answers:

- `Yes`: If yes is provided, then the playback requirements for this media profile as documented in this specification are expected to be fulfilled.
- `No`: If no is provided, then the playback of the media profile is not supported by the device and the application shall not playback this media profile.
- `unknown`: In this case the application should find other options to identify if the media profile can be played back.

Note that the media profile does not support the configuration signaling and requires an additional capability mechanism on which configuration is preferably used.

A.2.2. CMAF Header

In this case an API between the app and the platform exists, such that the application queries the device if the content described in the CMAF header can be played back. This has the advantages of being complete, accurate, future-proof, but the drawback of not being human readable and possibly requires transmitting more information than the other approaches.

Again, the device may provide one of the following answers:

- `Yes`: If yes is provided, then the playback requirements for this media profile as documented in this specification are expected to be fulfilled.
- `No`: If no is provided, then the playback of the media profile is not supported by the device and the application shall not playback this media profile.
- `unknown`: In this case the application should find other options to identify if the media profile can be played back.

If a no or an unknown is provided, the response should provide an indication based on what feature the device rejected the playback.

A.2.3. MIME Subparameters

This option consists in using one or more MIME sub-parameters to describe the different required capabilities (pre-decoding, decoding, and post-decoding). It is the mostly used options today because it has the advantages of enabling a progressive, detailed, compact and almost human readable signaling.

Post-decoding requirements are indicated in the ISO base media file format with restricted schemes. For example, the 'resv' sample entry type can be used for video tracks that require certain post-decoding operations. Similarly, pre-decoding requirements are indicated in the ISO base media file format with the protected scheme.

In this case, the application uses the detailed MIME type string for the communication with the device platform. The application queries the device of the media profile can be played back using:

- `<mediatype>/mp4 mime-subparameters`

The device may provide one of the following answers:

- `Yes`: If yes is provided, then the playback requirements for this media profile as documented in this specification are expected to be fulfilled.
- `No`: If no is provided, then the playback of the media profile is not supported by the device and the application shall not playback this media profile.
- `unknown`: In this case the application should find other options to identify if the media profile can be played back.

If a no or an unknown is provided, the response should provide an indication based on what feature the device rejected the playback.

A.2.4. Media Capabilities

Aligned with the Media Capabilities API[2], APIs exposing information about the decoding and encoding capabilities of a device platform for a given format, but also output capabilities of the current device to find the best match based on the device's display may be used. The application would query a vector of capabilities and if all required capabilities are supported, then the playback may be initiated.

A.2.5. Device Capability – Persistent Item Solution

Another approach is to standardized device capabilities in a known format for storage by the manufacturer and recall in some manner available to the application. The preferred approach uses standardizes key-value pairs for relevant player characteristics that can be communicated to servers via JavaScript APIs or Objects. One option is to use the HTTP User Agent String as described in [HBBTV] 7.3.2.4.

[2] https://wicg.github.io/media-capabilities/

A.2.6. User Agent String

The approach can provide something that can be deployed on devices where the new media capabilities API (or some future derivative) isn't supported. Typically, when integrating a browser on to a device, the HTTP User Agent is something that can be set by the device vendor / integrator without changing the code of the browser.

The user agent string would indicate which WAVE media profiles are supported in the HTTP UA. One example of how this could be done would be to include 3 bit-fields, one each for video, audio and captions/subtitles. Each would have one bit for each WAVE media profile with '1' indicating supported and '0' indicating not supported. HTML pages can read the HTTP User Agent string via the `Navigator.userAgent` property.

The strengths of this specific proposal (relative to others) are:

- Simplicity.
- Ease of integration for device makers/integrators due to not needing changes to the browser code.
- Something WAVE specific can also indicate support for WAVE device playback requirements and not "just" some support for the codec (which can be tested via `isTypeSupported` and/or `canPlayType`).

The weaknesses of this specific proposal (relative to others) are:

- Generic hostility to using the UA string.
- The ease with which it can be faked.
- Does not expose the same level of detail as the media capabilities API such as "smooth" and “powerEfficient”.
- More work would be needed to quantify how much added value it has over isTypeSupported / canPlayType in practice.

There are also of course issues common to other approaches:

- That they do not expose support for options within a WAVE media profile
  concerns that exposing the supported media profiles can be used in combination with other information to identify / track the user.

A.2.7. WAVE Playback Capabilities

Finally, there may be defined a dedicated capability code for the platform that matches against the full requirements in this specification. While such an approach may provide the most stringent interoperability, at the same time adding yet another option to the already complex world of capability signaling was dispensed during the development of this specification.

A.2.8. Rendering and Display Capabilities

For the case when remotely connected displays are used, the display capabilities (or audio rendering system) needs to be discovered from the user agent. It is unclear if this is possible in general and what the limitations are. May be a question towards HTML 5 API.

Discussion on HDMI details are for further study.

A.3. Recommendations for Capability Discovery APIs

Based on the discussions in clause 6.2.2, it is recommended that new devices support one of the following two API structures:

- Media Profile capability API as defined in clause 6.2.2.1. However, this requires an additional signaling for identifying configuration options.
- CMAF header API as defined in clause 6.2.2.2. However, this requires an additional signaling for identifying configuration options.

In both cases, if there is a `YES` response to the capability, the device shall support the playback requirements defined in this specification for a specific media profile. If the device responds with a no, then it should indicate what capabilities are not supported.

However, devices may not implement either of these two capability APIs, the response of the device for such an API may be `unknown` and in particular existing devices may not support such an API at all. If a media profile wants to enable playback on devices that do not support one of the two APIs as recommended above, then it is recommended that the media profile documents other means for capability detection for such a media profile, in particular using one of the following:

- MIME Subparameters as documented in clause 6.2.2.3
- Media Capability approach as documented in clause 6.2.2.4

Each media profile should provide sufficient information on how to use APIs for capability discovery in order to ensure the playback of the media profile following the requirements in this specification. Specifically, suitable capability discovery for existing devices is recommended to be added.

Annex B: Relevant HTML 5 APIs (Informative)

B.1 General

This clause documents the HTML 5 APIs that are relevant for the purpose to implement the high-level tests as defined in this specification.

B.2 Relevant Web Media APIs

The following APIs are relevant and their existence is assumed when implementing the tests based on an HTML 5 platform. The exact mapping will be addressed in a future version of the specification.

- **HTMLMediaElement**
  - Properties
    - i. HTMLMediaElement.buffered - OBSERVATION - returns a TimeRanges object that indicates the ranges of the media source that the browser has buffered (if any) at the moment the buffered property is accessed.
    - ii. HTMLMediaElement.currentTime - OBSERVATION INPUT - is a double indicating the current playback time in seconds. Setting this value seeks the media to the new time.
  - Methods
    - i. HTMLMediaElement.fastSeek() - INPUT - directly seeks to the given time.
    - ii. HTMLMediaElement.pause() - INPUT - pauses the media playback.
    - iii. HTMLMediaElement.play() - INPUT - begins playback of the media.

Other properties and methods are available as follows:

- **HTMLMediaElement**
  - Properties
    - i. HTMLMediaElement.defaultPlaybackRate - OBSERVATION INPUT - is a double indicating the default playback rate for the media.
    - ii. HTMLMediaElement.duration - OBSERVATION - returns a double indicating the length of the media in seconds, or 0 if no media data is available.
    - iii. HTMLMediaElement.initialTime - OBSERVATION - returns a double that indicates the initial playback position in seconds.
    - iv. HTMLMediaElement.mediaKeys - OBSERVATION INPUT - returns a MediaKeys object or null. MediaKeys is a set of keys that an associated HTMLMediaElement can use for decryption of media data during playback.
    - v. HTMLMediaElement.muted - OBSERVATION INPUT - is a Boolean that determines whether audio is muted. true if the audio is muted and false otherwise.
    - vi. HTMLMediaElement.paused - OBSERVATION - returns a Boolean that indicates whether the media element is paused.
    - vii. HTMLMediaElement.playbackRate - is a double that indicates the rate at which the media is being played back.

viii. HTMLMediaElement.played - OBSERVATION - returns a TimeRanges object that contains the ranges of the media source that the browser has played, if any.
ix. HTMLMediaElement.preservesPitch - OBSERVATION INPUT - is a Boolean that determines if the pitch of the sound will be preserved.
x. HTMLMediaElement.seekable - OBSERVATION - returns a TimeRanges object that contains the time ranges that the user is able to seek to, if any.
xi. HTMLMediaElement.volume - OBSERVATION INPUT - is a double indicating the audio volume, from 0.0 (silent) to 1.0 (loudest).
xii. HTMLVideoElement.height - OBSERVATION INPUT - Is a DOMString that reflects the height HTML attribute, which specifies the height of the display area, in CSS pixels.
xiii. HTMLVideoElement.width - OBSERVATION INPUT - Is a DOMString that reflects the width HTML attribute, which specifies the width of the display area, in CSS pixels.
xiv. HTMLVideoElement.videoHeight - OBSERVATION - returns an unsigned long containing the intrinsic height of the resource in CSS pixels, taking into account the dimensions, aspect ratio, clean aperture, resolution, and so forth, as defined for the format used by the resource.
xv. HTMLVideoElement.videoWidth - OBSERVATION - returns an unsigned long containing the intrinsic width of the resource in CSS pixels, taking into account the dimensions, aspect ratio, clean aperture, resolution, and so forth, as defined for the format used by the resource.

- **MediaSource**
  - Properties
    i. MediaSource.sourceBuffers – OBSERVATION - returns a SourceBufferList object containing the list of SourceBuffer objects associated with this MediaSource.
    ii. MediaSource.activeSourceBuffers - OBSERVATION - returns a SourceBufferList object containing a subset of the SourceBuffer objects contained within SourceBuffers — the list of objects providing the selected video track, enabled audio tracks, and shown/hidden text tracks.
    iii. MediaSource.readyState – OBSERVATION - returns an enum representing the state of the current MediaSource, whether it is not currently attached to a media element (closed), attached and ready to receive SourceBuffer objects (open), or attached but the stream has been ended via MediaSource.endOfStream().
    iv. MediaSource.duration – OBSERVATION INPUT - the duration of the media that is playing
  - Methods
    i. MediaSource.addSourceBuffer() - INPUT - creates a new SourceBuffer of the given MIME type and adds it to the MediaSource's SourceBuffers list.

ii. MediaSource.removeSourceBuffer() – INPUT - removes the given SourceBuffer from the SourceBuffers list associated with this MediaSource object.
iii. MediaSource.endOfStream() – INPUT - signals the end of stream.

Consumer Technology Association Document Improvement Proposal

If in the review or use of this document a potential change is made evident for safety, health or technical reasons, please email your reason/rationale for the recommended change to standards@CTA.tech.

Consumer Technology Association
Technology & Standards Department
1919 S Eads Street, Arlington, VA 22202
FAX: (703) 907-7693 standards@CTA.tech

Consumer Technology Association™