CTA Specification

Web Application Video Ecosystem – Web Media Application Developer Guidelines

CTA-5002

December 2018

W3C®

Consumer Technology Association™

NOTICE

(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

Foreword

The CTA WAVE Project was launched at the 2016 CES in Las Vegas, NV. The goal of WAVE is to improve interoperability in the commercial Over-the-Top (OTT) video ecosystem using industry-standard protocols including HTML5 with MSE Extensions [MEDIA-SOURCE][1] and EME [ENCRYPTED-MEDIA][2], MPEG-CMAF[3], MPEG-CENC[4] and adaptive bit-rate streaming protocols MPEG-DASH[5] and Apple HLS[6].

The process under which WAVE develops specifications is available online at the WAVE public home page (https://cta.tech/Research-Standards/Standards-Documents/WAVE-Project/WAVE-Project.aspx).

The *Web Media Application Developer Guidelines* were co-developed between the CTA WAVE HTML5 API Task Force and the W3C Web Media API Community Group. It is jointly published between CTA (as a CTA specification, CTA-5002) and W3C (as a Final Community Group Report), by agreement between the two organizations.

Web Media Application Developer Guidelines

CTA Status: CTA Specification CTA-5002, "WAVE Web Media Application Developer Guidelines"

W3C Status: Final Community Group Report, 13 December 2018

Latest editor's draft:
https://w3c.github.io/webmediaguidelines/

Editors:
Joel Korpi (AppNexus)
Thasso Griebel (CastLabs)

Former Editors:
Jeff Burtoft (Microsoft)

Participate:
GitHub: https://github.com/w3c/webmediaguidelines/
File a bug: https://github.com/w3c/webmediaguidelines/issues/
Commit history: https://github.com/w3c/webmediaguidelines/commits/gh-pages
Pull requests: https://github.com/w3c/webmediaguidelines/pulls/

[1] https://w3c.github.io/webmediaapi/#bib-MEDIA-SOURCE
[2] https://w3c.github.io/webmediaapi/#bib-ENCRYPTED-MEDIA
[3] ISO/IEC 23000-19:2018, "Information technology – Multimedia application format (MPEG-A) – Part 19: Common media application format (CMAF) for segmented media", https://www.iso.org/standard/71975.html
[4] ISO/IEC 23001-7:2015, "Information technology – MPEG systems technologies – Part 7: Common encryption in ISO base media file format files", https://www.iso.org/standard/65271.html
[5] ISO/IEC 23009-1:2014, "Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats", https://www.iso.org/standard/65274.html
[6] Pantos, R., Ed., and W. May, "HTTP Live Streaming", https://tools.ietf.org/html/draft-pantos-http-live-streaming-20

Abstract

This specification is a companion guide to the Web Media API specification[7]. While the Web Media API spec is targeted at device implementations to support media web apps in 2018, this specification will outline best practices and developer guidance for implementing web media apps. This specification should be updated at least annually to keep pace with the evolving web platform. The target devices will include any device that runs a modern HTML user agent, including televisions, game machines, set-top boxes, mobile devices and personal computers.

The goal of this Web Media API Community Group specification is to transition to the W3C Recommendation Track for standards development.

These Guidelines were developed as part of the CTA WAVE Project[8].

Status of This Document

This specification was published by the Web Media API Community Group[9]. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA)[10] there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups[11].

[7] See https://w3c.github.io/webmediaapi/.
[8] http://cta.tech/WAVE
[9] https://www.w3.org/community/webmediaapi/
[10] https://www.w3.org/community/about/agreements/cla/
[11] https://www.w3.org/community/

Table of Contents

Web Application Video Ecosystem –
Web Media Application Developer Guidelines

1 Introduction

1.1 Scope

The scope of this document includes general guidelines, best practices, and examples for building media applications across web browsers and devices.

The target audience for these guidelines is software developers and engineers focused on building cross-platform, cross-device, HTML5-based applications that contain media-specific use cases.

The focus of this document is on HTML5-based applications, however the use cases and principles described in the guidelines can be applied to native applications (applications that have been developed for use on a particular platform or device). The examples in this document provide a starting point to build your media application and includes example implementations from various providers and vendors. This document also includes sample content and manifests as well as encoding guidelines to provide hints on achieving the best quality and efficiency for your media applications.

1.2 Accessibility

These guidelines will cover making your applications compliant with accessibility requirements from the perspective of delivering and consuming media. However, to make sure your entire application is accessibility-friendly, please see the W3C Web Content Accessibility Guidelines[12].

1.3 Glossary of Terms

The following lexicon provides the common language used to build this document and communicate concepts to the end reader.

Term	**Definition**
360 Video	Video content where a view in every direction is recorded at the same time, shot using an omni-directional camera or a collection of cameras. During playback the viewer has control of the viewing direction like a spherical panorama.
AAC	Advanced Audio Coding is a proprietary audio coding standard for lossy digital audio compression. Designed to be the successor of the MP3 format, AAC generally achieves better sound quality than MP3 at the same bit rate.

[12] See https://w3c.github.io/webmediaguidelines/#bib-wai-webcontent.

Term	**Definition**
ABR	Adaptive Bitrate Streaming is a method to optimize video playback by automatically selecting a specific bitrate rendition of a video file within the video player. Some ABR algorithms use client bandwidth throughput to determine which rendition to play. For example, if a client tries to play a video on a slow internet connection, the player could automatically select the lower bitrate rendition to maximize playback speed.
AVOD	Advertising-supported Video on Demand. AVOD services monetize their content by serving ads to users, as opposed to other business models such as paid subscription or pay-per-title.
Bit rate	(Also known as data rate) The amount of data used for each second of video. In the world of video, this is generally measured in kilobits per second (kbps) and can be constant or variable.
codec	A codec is an algorithm defining the compression and decompression of digital video or audio. Most codecs employ proprietary coding algorithms for data compression. MP3, H.264, and HEVC are examples of codecs.
CDN	A content delivery network (CDN) is a system of distributed servers (network) that deliver pages and other media content to a user, based on the geographic locations of the user, the origin of the webpage and the content delivery server.
Chunk or *Chunking*	ABR technologies typically break a video or audio stream into chunks to make transmission more compact and efficient. These chunks are typically 2-10 seconds in length and contain at least one I-Frame so that the video player has complete information for which to render the video from that point in the manifest.
Closed Captions	Used for accessibility and usability, closed captions are a visual representation of the audio content of a media file stored as metadata file or track and displayed as an overlay in the video player in synchronization with the video/audio track. Typical formats for closed captioning are WebVTT and SRT.
DRM	A combination of encryption, authentication and access control technologies that are used to prevent unauthorized users from consuming copyrighted media content. The most widely used DRM solutions today are Microsoft PlayReady, Google Widevine, and Apple FairPlay Streaming. There are also open technologies that use standards such as Common Encryption [*MPEGCENC*] [13] and Media Source Extensions (MSE) [*MEDIA-SOURCE*][14] to create a secure environment without third-party products.
Embed	Most video players in a web page use what's called an "Embed" or "Embed Code" that is placed in the code of your HTML page or app that will render the video playback environment.

[13] https://w3c.github.io/webmediaguidelines/#bib-mpegcenc.
[14] https://w3c.github.io/webmediaguidelines/#bib-media-source.

Term	**Definition**
Encoding	Also referred to as compression, this is the process of removing redundancies from raw video and audio data, thereby reducing the amount of data required to deliver the media across a network, on a physical disc, etc. The process can result in reduced visual or auditory quality of the encoded media, but the loss is usually imperceptible to the user. H.264 and HEVC are examples of codecs that use compression during transcoding.
EME (Encrypted Media Extensions)	Encrypted Media Extensions (EME) is a recommended W3C specification for providing a communication channel between web browsers and digital rights management (DRM) agent software. This allows the use of HTML5 video to play back DRM-wrapped content such as streaming video services without the need for third-party media plugins like Adobe Flash or Microsoft Silverlight. The use of a third-party key management system may be required, depending on whether the publisher chooses to scramble the keys.
Container (Format)	A format or "container format" is used to bind together video and audio information, along with other information such as metadata or even subtitles. For example, .MP4, .MOV, .WMV etc. are all container formats that contain both audio, video, and metadata in a single file. Formats contain tracks that are encoded using codecs. For example an .MP4 might use the AAC audio codec together with the H.264 video codec.
H.264	Also known as MPEG-4 AVC (Advanced Video Coding) it is now one of the most commonly used recording formats for high definition video. It offers significantly greater compression than previous formats.
HEVC	High Efficiency Video Coding is one of the newest generation video codecs that is able to achieve efficiency up to 4x greater than H.264, but it requires the HEVC codec to be present on the device. HEVC typically takes considerable more processing power to encode and decode than older codecs such as H.264.
HLS	HTTP Live Streaming (HLS) is an adaptive streaming technology created by Apple that allows the player to automatically adjust the quality of a video delivered to a web page based on changing network conditions to ensure the best possible viewer experience.
I-Frame (video)	In video compression, an I-Frame is an independent frame that is not dependent on any future or previous frames to present a complete picture. I-Frames are necessary to provide full key frames for the encoder/decoder.
In-stream (Advertisement)	A video advertisement that accompanies the main video content. Examples include pre-rolls, mid-rolls, and post-rolls which play before, during, or after a main piece of content, respectively.
Live Streaming	Live streaming is a type of broadcast that is delivered over the Internet where the source content is typically a live event such as a sporting event, religious service, etc. Unlike VOD, viewers of a live stream all watch the same broadcast at the same time.

Term	**Definition**
Manifest	A manifest is a playlist file for adaptive streaming technologies (such as HLS, DASH, etc.) that provides the metadata information for where to locate each segment of a video or audio source. Depending on the configuration, some technologies have two types of manifests: one "master" manifest that contains the location of each rendition manifest and one rendition manifest for each rendition that contains the location (relative or absolute) of each chunk of a video or audio source.
Media Renderer	A process that takes as input file-based video bytes and renders those bytes to the client display, such as a computer monitor or television display. The media renderer is responsible to ensure that the video is displayed accurately, per the encoded profile.
Metadata Track	ABR streaming technologies contain the ability to include not only video and audio tracks within the stream, but also allow for metadata tracks for applications such as closed captions, advertising cues, etc.
Mezzanine File	An intermediate file typically created after post-production that is used for distribution or for input into another workflow process, such as transcoding. Mezzanine files are typically very high quality to ensure the minimum quality degradation. Often times, a lossless codec is used to achieve this high quality.
MPEG-DASH	Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is an adaptive bitrate streaming technique that enables high quality streaming of media content over the Internet delivered from conventional HTTP web servers.
Outstream Advertisement	A video advertisement that stands alone, without accompanying main video content. Examples include a video ad that is embedded within a text news article page that contains text and images as the main content medium.
Player	A video or audio media player that is used to render a media stream in your application environment. There are commercially available and open source video players and SDKs for almost every platform.
Rendition	A specific video and audio stream for a target quality level or bitrate in an adaptive streaming set or manifest. For example, most HLS or DASH adaptive streaming manifests contain multiple renditions for different quality/bitrate targets so that the viewer automatically views the best quality content for their specific Internet connection speed.
SDK	A Software Development Kit is a set of tools that allow the creation of applications for a certain framework or development platform. Typically they are the implementation of one or more APIs to interface to a particular programming language, and include debugging utilities, sample code, and documentation.
Streaming	Delivering media content to a viewer via Internet protocols such as HTTP or RTMP.

Term	Definition
SVOD	Subscription-supported Video on Demand. SVOD services monetize their content through paid subscriptions from users, as opposed to other business models such as advertising or pay-per-title.
Transcoding	The process of converting a media asset from one codec to another.
Trick Play/Mode	Trick mode, sometimes called trick play, is a feature of digital video systems including Digital Video Recorders and Video on Demand systems that mimics the visual feedback given during fast-forward and rewind operations that were provided by analogue systems such as VCRs.
URL Signing	URL signing is a mechanism for securing access to content. When URL Signing enforcement is enabled on a property; requests to a content server or content delivery network must be signed using a secure token. The signing also includes an expiration time after which the link will no longer work.
VOD	Video on demand is video that is served at the request of a user. Delivery is typically through a content delivery network to optimize delivery speed and efficiency.
VR	Virtual Reality is a realistic, immerse, three-dimensional environment, created using interactive software and hardware, and experienced or controlled by movement of the body. VR experiences typically require wearing a head mounted display (HMD).

For a more detailed list of relevant terms, please see the glossary of terms for the vendors or technologies you use in your workflow.

2 Media Playback Use Cases

2.1 General Description

Material (typically in video or audio content) is made available by a content provider via a web-enabled application and delivered by a content distribution network. There are three distinct interlocking processes: generation, delivery and consumption / playback.

2.1.1Content Generation

Mezzanine File content is normally delivered to the service provider as a file with near lossless compression. Specifically, for High Definition content at 1080p with 24, 30, 50, or 60 frames-per-second, the bitrate of the original content is typically 25Mbps to 150Mbps for 1080P HD and 150Mbps - 300 Mbps for 4K UHD.

ABR streaming content is generated by transcoding the Mezzanine File against an transcoding profile. Firstly, the mezzanine file is transcoded into a set of versions, each with its own average bitrate. Secondly, each version is packaged into segments of a specified duration.

The first process is defined by an transcoding profile. A profile describes the set of constraints to be used when video is being prepared for consumption by a range of video applications. The description includes the different bitrates to be generated during the transcoding process that will allow for the same content to be consumed on a wide variety of devices on different networks from cellular to LAN.

The second process is performed by a packager which segments the different bitrates. Next, the output is packaged into a transport format such as transport streams (.ts) or fragmented MP4s (.m4s). Lastly, they are optionally encrypted with a DRM that is suitable for the environment where the content is going to be played out. The packager is also responsible for the generation of a manifest file, typically a DASH (.mpd), HLS (.m3u8) or possibly Smooth (.ism) or HDS (.f4v), that specifies the location of the media and its format.

2.1.2 Content Delivery

After the content has been generated the resulting segments of video and corresponding manifest files are pushed to an origin server. However, the assets are rarely delivered directly from the origin.

At this stage, the control in the chain switches to the client's web video application. The content provider supplies the client with the URL of a manifest file located on a CDN rather than the origin. The manifest URL is typically passed to a video player for playback. The player makes a HTTP GET request for the manifest from CDN edge. If the CDN edge does not currently have the manifest available, the CDN requests a copy of the file from the origin and the file is cached at the edge for later use. The CDN edge then returns the manifest to the requesting player. *(NOTE: There are different levels of caching; popular VOD content is kept closer to the edge in the CDN network in this way it can be delivered to customers faster than an edge server that isn't tuned for high volume delivery.)*

2.1.3 Content Playback

As mentioned in the Content Generation section, the player uses a tag within the manifest to determine the playout type. In HLS, if there is a type with the value of VOD, the player will not reload the manifest. This has important consequences if there are changes in availability after the session as commenced. In DASH, the difference between a live and VOD playlist are subtler. At this point the behavior differs between players found on the different devices depending on the transport formats. However, broadly, it behaves in the following way:

1. While the player’s ABR algorithm determines which rendition to use during playback, for instance based on the currently measured network throughput or the current buffer, the algorithm might not have enough information at playback start to use such metrics. The initial decision depends on the player’s ABR algorithm but is often based on prior knowledge. This can for instance be previous measurements of network throughput, a simple default value that is used if not enough data is available to estimate the current

network throughput, or a custom algorithm to determine the initial bandwidth estimation.

2. Once the player has this information it can then compare this with the metadata from the manifest that describes the different qualities that the content provider is supplying. It picks the quality level with an average bitrate that is as close to the calculated throughput but within its bounds to avoid a situation where a consistent experience is interrupted as the player requires more data than the current network bandwidth can supply, where the player's buffer is emptying faster than it is being filled.
3. It then requests a segment from a location on the edge server that typically relative to the location of the manifest.
4. The downloaded segment is then added to the video decoder pipeline for playback. For player's that are based on Media Source Extensions [*MEDIA-SOURCE*][15] the segment is appended to the corresponding media source buffer. Note that the segment is appended as is even if the content is encrypted. The underlying DRM system is usually integrated with the decoder pipeline and will decrypt the samples prior to decoding.
5. The media renderer pulls the video data from the buffer and passes it to the video surface where it is rendered.
6. If the available bandwidth remains constant the player will continue to request segments from the same bitrate stream, pulling down chunks and filling its video buffer. In the event a change in network availability the player will make a decision about the need to either drop to a lower rendition or request a higher bitrate. In case of HLS, switching the rendition will trigger a download of a new child manifest prior to the download of the associated segment. In case of MPEG-DASH the player can start downloading the associated segment directly.

NOTE: This workflow may have additional steps if DRM is used.

2.2 On-Demand Streaming (VOD)

Despite the almost identical mechanics, content generation, delivery, and play out used for VOD and live, a large organization will typically maintain two distinct workflows as there are subtle but important ways in which they differ.

2.2.1 Content Generation

For VOD the source is typically static file-based rather than a feed. The encoding profiles are also subtly different. A greater priority can be placed on high quality as the latency, time to live, is not a requirement. There are important differences in the manifests created. In HLS there is a tag that tells the player whether the playlist is describing on demand material: #EXT-X-PLAYLIST-TYPE:VOD. As we will see shortly this is used by the player. There are also client-side restrictions where certain profiles are blocked due to rights restrictions and network

[15] https://w3c.github.io/webmediaguidelines/#bib-media-source.

consumption capped bitrates on movies and entertainment while being allowed on sports content. Fragments size will also effect playout as a player’s ABR can be more responsive if the chunk is smaller, e.g. 2 seconds rather than 10 seconds.

2.2.2 Content Delivery

The CDN configuration and topology for delivering VOD content is also different to live. There are different levels of caching; popular VOD content which is kept closer to the edge in the CDN network in this way it can be delivered to customers faster than an edge server that isn't tuned for high volume delivery. Older and less popular content is retained in mid-tier caching while the long tail content is relegated to a lower tier.

2.2.3 Content Playback

As mentioned in the content generation section the player uses a tag within the manifest to determine the playout type. In HLS if there is a type with the value of VOD, then the player will not reload the manifest. This has important consequences if there are changes in availability after the session as commenced. In DASH the difference between a live and VOD playlist are more subtle but one of the primary identifier for liven content in MPEG-DASH is the profiles attribute of the MPD element which could for example reference the urn:mpeg:dash:profile:isoff-live:2011 profile to indicate live content.

There are other differences in playout as well. Unlike live, a VOD asset has a predefined duration, information around duration and playback position can be used to update the UI to provide feedback to the user on the proportion of the asset watched.

Additionally, the UX requirements are different between VOD and live. Unlike live content, which lends itself to being browsed within an EPG (electronic program guide), VOD content is typically navigated using tiles or poster galleries. There is also a trick play bar to view the current playback position and seek to other points within the content.

2.2.4 VOD Pre-caching

In the previous sections, we outlined the use cases associated with video streaming. In this section, we give some examples of use-cases that are specific to on-demand streaming and are mainly related to strategies employed on the clients to improve performance in some way.

Pre-caching is a strategy used in on-demand streaming and is important to ensure the best user experience. By buffering the beginning segments of a video, you can make playback as close to instantaneous as possible. This is important because buffering (the state of the video application when the player has insufficient content within its framebuffer to continuously play content), has a direct relationship to engagement and retention. For every second of buffering within a session, a certain number of users abandon a video stream. Web video application developers will use points within an application's UX to pre-cache content. For example, when entering a mezzanine/synopsis page, the application might preemptively cache the content by filling the player's video buffer or, alternatively, storing the chunks locally. Pre-caching allows the video to commence playing without buffering and provides the user with a more responsive initial playback experience. This technique is used by Netflix, Sky and the BBC in the case of on-demand content being watched in an in-home context. This technique is not used for cellular sessions where the user's mobile data would be consumed on content that they do not watch.

2.3 VOD – further considerations

2.3.1 Byte range requests in context of web video application

An HTTP range request allows a client to request a portion of a larger file. In the VOD scenario where a user wants to access a specific location, a range request allows the player access content at a specific location without downloading the entire transport chunk.

Both the player and the server need to support range requests. The player needs to be able to playback a source buffer and the server must be configured to serve ranges. The exchange begins when a client makes an HTTP HEAD request. If range requests are supported, the server will then respond with a header that includes Accept-Ranges: bytes and the client can issue subsequent requests for partial content. The returned bytes are added to an ArrayBuffer and then appended to the SourceBuffer, which in turn is used as the SRC parameter of the of the video HTML5 tag/element.

2.3.2 HDCP security requirement for HDMI

Some content has limitations placed on its distribution by the owners. There are different ways of protecting the owner's rights. The most common is DRM, which prevents the content being watched on a client device if the user cannot 'unlock' it with a key. However, once the content is unlocked the service provider should make every effort to prevent the user from re-distributing this content.

A technical solution to prevent interception as a signal travels from a device to a television or projector is termed High Bandwidth Content Protection (HDCP). If implemented by a device manufacturer, HDCP takes the form of a key exchange between devices. If the exchange fails, the signal is not transmitted to the display and the device is responsible for notifying the end user of the error.

2.3.3 Watermarking

Support for real-time watermarking is becoming an important consideration for distributors of high-value content. A service provider's right to distribute content is linked to their ability to protect it with studios and sports channels insisting on the capacity for a service provider to detect a breach of copyright on a stream by stream basis. A service provider can include a vendor's SDK in the client that can add a watermark at run-time. Normally, the watermark is invisible to the consumer and takes the form of a digital signal, this is referred to as forensic water marking. Part of the service provided by the watermarking vendor is to monitor the black market for re-distributed material. Illicit streams or files are intercepted and screened for the digital fingerprint inserted on the client. If a suspect stream is found the vendor directs the service provider(s) to the source of the misappropriated stream.

While watermarking is not an issue that developers will often be faced with, the processing requirements can dictate the choice of platform for the content distribution. The watermarking process might require processing power and low-level functionality through ring-fenced resources that platforms such as Android, iOS or Roku provide. The options for watermarking in the browser are limited to overlays and this limitation is then reflected in a distributor's choice of platform for their web video application.

2.4 Live Streaming

Even though Live and On-Demand streaming scenarios have a lot in common, there are a few distinct differences.

2.4.1 Content Generation

In contrast to VOD content generation, the typical input is a live feed of data. Usually there is also a higher priority on low-latency and time to live, which is in turn reflected in smaller segments.

A big difference between VOD and Live content generation can be found in the differences between manifests for the two content types. Besides a general profile that tells the player if the manifest represents live content, there are a few other, important properties that define how a player will be able to playback the live content, when updates will be fetched, and how close to the live edge playback starts. These properties are pre-defined and expressed in the manifest during content preparation but play an important role in content playback and view experience.

2.4.2 Content Delivery

The content delivered through a CDN is generally very similar to the VOD playback use-case. There might be slight differences when it comes to caching and distribution of segments, and you might observe higher load on the origin when playback is configured to be very close to the live edge.

2.4.3 Content Playback

Even though playback and playback parameters between Live and VOD playback are very similar, there are critical differences. Maybe the biggest difference is the playback start position. While VOD playback session usually start at the beginning on the stream, a Live playout usually starts close to the live edge of the stream, with only a few segments between the playback start position and the end of the current manifest.

The other main difference is that the live content changes over time and the player needs to be aware of these changes. In both DASH and HLS, this results in regular manifest updates. Depending on the format, it is possible to define the interval for the manifest updates.

What happens during a typical playback session is:

1. Player loads the manifest for the first time
2. Playback starts at the live-edge defined in the manifest
3. Player re-loads the manifest in regular, pre-defined, intervals.
4. The new manifest is used to update playback state information and the segment list

This is a typical loop that the player goes through on each manifest update cycle.

2.4.4 Potential Issues

One of the most common cases that an application needs to be prepared for is the player falling behind the live window. Assume for example that you have a buffer window of 10 seconds on your live stream. Even if the user interface does not allow explicit seeking, buffering events can still stall the playout and push the playback position towards the border of the live window and even beyond it. Depending on the player implementation the actual error might be different, but implementation should be aware of the scenario and be prepared to recover from it.

Another common problem that can occur both in Live and VOD playback sessions are missing segment data or timestamp alignment issues. It might happen that a segment for a given quality is not available on the server. At the same time, misalignments in segment timestamps might occur. Although both of these issues might be critical, how they are handled depends on the player implementation. A player can try to jump over missing segments or alignment gaps. A player can also try to work around missing segments by downloading the segment from a different rendition and allow a temporary quality switch. At the same time, a player implementation might also decide to treat such issues as fatal errors and abort playback. Applications can try to manually restart playback sessions in such scenarios.

2.5 Thumbnail Navigation

Thumbnails are small images of the content taken at regular time intervals. They are an effective way to visualize scrubbing and seeking through content.

There is currently no default way to add thumbnail support to a playback application. And there is not out-of-the-box browser support. However, since thumbnails are just image data, the browser has all the capabilities for a client to implement thumbnail navigation in an application.

The most common way to generate thumbnails is to render a set of images out of the main content in a regular time interval, for example every 10 seconds. The information about the location of these images then needs to be passed down to the client, which can then request and load an image for a given playback position. For more efficient loading, images are often merged into larger grids (sometimes called sprites). This way, the client only needs to make a single request to load a set of thumbnails instead of a request per image.

Unfortunately, neither DASH or HLS do currently specify a way to reference thumbnail images directly from manifests. However, the DASH-IF Guidelines [*DASHIFIOP*][16] describe an extension to reference thumbnail images. The thumbnails would be exposed as single or gridded images. All parameters required to load and display the thumbnail images are contained in the Manifest. This approach also works for live Manifests that are updated regularly by the player. The following example shows how thumbnails can be referenced according to the DASH-IF Guidelines:

Example 1: Thumbnail Reference in DASH-Manifest

```
<AdaptationSet id="3" mimeType="image/jpeg" contentType="image">
  <SegmentTemplate media="$RepresentationID$/tile$Number$.jpg" duration="125"
startNumber="1"/>
  <Representation bandwidth="10000" id="thumbnails" width="6400"
height="180">
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/thumbnail_tile" value="25x1"/>
  </Representation>
</AdaptationSet>
```

3 Media Playback Methods

3.1 Device Identification

Device identification is required both at the level of device type and family and also to uniquely identify a device. Different techniques are used in different environments.

3.1.1 Device Type

In the context of a mobile client the broad device type is already known - you can’t install an Android client on an iOS device. However, operating systems evolve and are extended. Within the application layer you will still need to determine the level of support for feature X and branch your code accordingly.

If the application is hosted or the same application is deployed via different app stores then the clients runtime could be more ambiguous. The classic approach is to use the navigator.userAgent. While this returns a string that does not explicitly state the device by name a regular expression match can be used to look for patterns that can confirm the device family. As an example, the name ‘Tizen’ can be found in the user agent strings for Samsung and ‘netcast’ for LG devices. In the Chrome browser the strings will be different on Windows, Mac, Android and webview.

[16] https://w3c.github.io/webmediaguidelines/#bib-dashifiop.

Another method similar to feature detection is to look for available APIs, many of these are unique to a device, for example if(tizenGetUser){ then do X }.

Besides the mentioned feature switches, device type identification can be crucial to determine which content should be loaded on a client device. A typical scenario for DRM protected content is to have different content types for different platforms based on the platforms playback capabilities. One might for instance package HLS and MPEG-DASH variants of the same content but using different encryption and DRM schemes. In that case the client application needs to decide which content to load based on the device type and its capabilities.

3.1.2 Unique device Identifier

After you have determined what type of device you are on you may need to identify that device uniquely. Most device manufacturers provide an API so once you have discovered what type of device you are on you then know what API's will be available to you. Each manufacturer provides a string constructed in a different way, some use the serial number of the device itself while others use lower level unique identifiers taken from the hardware. In each case the application layer should attempt to namespace this in some way to avoid an unlikely, but theoretically possible, clash with another user in any server-side database of users and devices. In a classic PC/browser, rather that mobile or STB (set-top-box), environment there are technical issues with using unique identifiers as a user can access these stored locally and either delete them or reuse them in another context, this could allow them to watch content on more devices than their user account allows. For this reason some vendors prefer to use DRM solutions that both create and encrypt unique identifiers in a way that obscures them from the user.

As mentioned above one typical use case where a unique device identifier is needed is to restrict the number of devices that a user can use. This could refer to overall devices or the number of devices a user can use in parallel. In both cases a unique device identifier will be required.

3.2 Device Content Protection Capabilities

A web video application needs to determine the content protection capabilities of the device that is being used to playback the content. The method of doing this will vary from device to device and between DRM systems.

Regardless of a device's capability to play back a stream encrypted with a specific DRM it is worth noting that a content provider will be aware of this capability in advance and consequently encrypt the streams to target a specific device. Beyond technical feasibility: can device X play back content encrypted with DRM Y? The provider will have a number of considerations when choosing a DRM for playback on a specific device: cost, utility, complexity and content value (the last consideration being mapped to contractual obligations). As a consequence, in most situations the API call that requests the stream from a back-end service will either be to a service that is only configured to return streams with a suitable encryption or the server will use data from the request headers or key-value pairs in the request payload to determine which streams to return.

In the context of the browser, the major vendors broadly dictate the DRMs available. Apple’s Safari browser and Apple TV support Apple’s FairPlay DRM. Microsoft's Internet Explorer and Edge browsers only support Microsoft's PlayReady out of the box, while Google’s Widevine Modular is supported by Google’s Chrome browser but is also included in the browser developers not tied to a significant hardware provider: Opera and Firefox.

Browser DRM detection capabilities are tested via the EME (Encrypted Media Extensions) API [*ENCRYPTED-MEDIA*][17]. The API is an extension to the HTMLMediaElement. The process of determining the available system is broadly as follows:

1. The stream is passed to the VIDEO or AUDIO HTML5 tag/media element.
2. The browser detects the stream is encrypted.
3. The media event encrypted is thrown.
4. A check is made to see if there are already MediaKeys associated with the element.
5. If there are no keys already associated with the element, then use the EME API navigator.requestMediaKeySystemAccess() to determine which DRM system is available. This is done by passing a key value pair that includes a string representation of each MediaKey system to the above method which in turn returns a Boolean.
6. Use the MediaKeySystemAccess.createMediaKeys() to return a new MediaKeys object.
7. Then use HTMLMediaElement.setMediaKeys(mediaKeys).

Please note that the browser will throw the encrypted event if it detects DRM init data or identifies an encrypted stream by other means. That does not mean that a client can establish encrypted playback only once the encrypted event was triggered. An encrypted session can also be established manually, without relying on the browser to detect encrypted content first.

The remaining steps are covered in a later section on using EME but it’s worth noting at this stage that the navigator.requestMediaKeySystemAccess() is not uniformly implemented across all modern browsers that support EME. As an example Chrome returns true for com.widevine.alpha however IE and Safari throw errors and Firefox returns null. A possible solution to this is offered at https://stackoverflow.com/questions/35086625/determine-%3ca%3eDRM%3c/a%3e-system-supported-by-browser.

3.3 Ad Insertions

There are many ways to incorporate media advertisements into an application. The most common types are:

1. In-stream: Video advertisement that is played before, mid, or after main content (such as a clip or episode).

[17] https://w3c.github.io/webmediaguidelines/#bib-encrypted-media.

2. Outstream: Video advertisement stands alone (without other video content) and is placed natively within other parts of the application

For the purposes of these guidelines, we will focus on in-stream.

There are two primary ways to insert advertisements into an in-stream media playback session. *Client-side ad insertion (CSAI)* and *Server-side ad insertion (SSAI)*. The difference here is that Server-side insertions are embedded into the playout directly, while Client-side insertions are added dynamically by the player and are handled in parallel to the main playout.

3.3.1 Client-side Ad Insertion

Client-side ad insertions typically involve using a script available in the client runtime, JavaScript in the case of the web, to insert an advertisement during the user's session. CSAI can be used for any type of content, including on-demand and live.

For Live Streaming use cases, in-video-stream metadata (for example, using ID3 in HLS) can be used to enable all viewers to see an ad at the same time, synced with the content at a specific time offset. Another approach is to use web sockets to signal a player via a pushed event to play an advertisement.

Additionally, for live TV (broadcast TV), many broadcasters use SCTE-104/35 in their workflows to signal ads. When preparing the content to be distributed through Internet these SCTE-104/35 signals are usually "converted" to ID3/EMSG messages for HLS/DASH respectively. This keeps the media workflow for live ads insertion easy (SCTE to ID3/EMSG conversion is supported by most of encoders/media servers used by live content generators), is easy to consume from media players (most of players support parsing ID3/EMSG metadata), ensures synchronization with the content and avoids the need of using alternative communication channels like websocket that add complexity to the solution.

As an example workflow: an ad serving vendor typically provides a client-side script that a publisher can embed in their application. These scripts create a bridge between the advertising demand source and the publisher's video player. Close to the player's initiation the client library makes its API available. The web application listens to events associated with playback, for example the video elements media event 'playing'. The web application then calls the DOM pause method on the video element and then calls the play method provided by the client-side ad library, passing it the 'id' of the video asset. This is then returned to the vendor, possibly along with other identifiers that can be used to target the audience with a specific ad. At this stage an auction is performed with business logic at the ad vendor determining which provider supplies the ad (this is a complex topic and outside the scope of this document). The vendor responds with a VAST (Video Ad Serving Template) payload that includes the URI of the ad content appropriate for the playback environment. In some cases, there is no ad, if this is the case the user is presented with the content they originally requested, and control is passed back to the web video application.

If there is an ad targeted against the content, then the library performs DOM manipulation and injects a new video element into the document this is typically accompanied by a further script that provides the vendor with insights based on the current session. The ad plays. The ad object will conform to VPAID (Video Player Ad Serving Definition) and present a standardized interface to the player for possible interaction, it will issue a standard set of events which the web application can listen to. In response to an ‘adEnded’ event the local library will tear down the injected DOM elements and in turn issue an event that the web application can use to trigger a return to playing the original content.

3.3.2 Server-Side Ad Insertion

Especially in live playback environments, because of the live nature of the playout, ad-insertions are usually not handle client side. One of the reasons is that the live feed continues independent of the client-side ad insertion. In that case the player can easily fall behind the live window. Server side ad insertions allow the player to play a continuous feed, independent of any ad insertions, which are "injected" on the server side, directly into the playout feed.

One of the requirements of the underlying streaming format such as DASH or HLS is that the encoding does not change during the playout of a single rendition. To still be able to insert advertisements, HLS and DASH propose different strategies.

HLS allows the packager to add a "discontinuity" tag. This is expressed with a #EXT-X-DISCONTINUITY entry before any potential format change. See Section 4.3.2.3. in [*HLS*][18] for more detail. This tag then usually appears before the ad starts and again before the main content continues.

In contrast, DASH uses multiple "periods" to split the content into main and advertisement sections. See Section 3.1.31 in [*MPEGDASH*][19] for more details. Each switch between periods is a trigger for the player to potentially reset the decoder.

Where the player usually receives dedicated events in CSAI such as 'adStarted' or 'adEnded', Server-side insertions require a different setup to trigger such events. Typically, in-band notifications are used to trigger time-based events. For example, SCTE-35 markers are a common format to insert time based meta-data into the feed. These markers are read and interpreted by the player and can be used to trigger events or carry additional meta-data such as callback URLs.

4 Content Encoding Guidelines

Encoding video for playback can be a challenge given the variety of ways users may watch your content. In order to provide quality video for your audience, there are multiple aspects to consider such as codec support, resolutions, frame-rates, browser versions, bandwidth availability and device compatibilities. Fortunately there are some steps you can follow that will help you produce content that is playable for all audiences. Preparing your source content and planning your outputs ahead of time are important activities to complete prior to actual

[18] https://w3c.github.io/webmediaguidelines/#bib-hls
[19] https://w3c.github.io/webmediaguidelines/#bib-mpegdash.

transcoding. Time spent here will help reduce trial and error later, and ensures maximum quality throughout your workflow, resulting in an excellent viewer experience for your entire audience. If these steps are rushed, then there will certainly be inefficiencies in video processing and wasted bandwidth, and the outputs may not be sufficient to cover your target audience, leading to higher cost and missed opportunities. There are a few key steps to get your content ready for web delivery.

These steps are best done in the following order:

1. Create a mezzanine file
2. Decide on rendition set
3. Decide on delivery formats to support
4. Create encoding profiles
5. Transcode mezzanine into specific rendition output formats

4.1 Create a Mezzanine File

The first step in preparing your content is to create a high-quality encoding master file from your source footage. This mezzanine file will be used as a source file to create all downstream outputs. Content producers typically export files from a non-live editor using the highest resolution input files available; from a master magnetic, optical media or digital source. This ensures all downstream outputs have the necessary quality to work with, without needing to re-export from the editor every time.

In order to maximize quality of the mezzanine, your export settings should use a lossless codec such as Apple ProRes, H.264 Intra-frame, or Motion JPEG 2000. A lossless codec guarantees all data from the original will exist on the export. For detailed info on how to configure your output settings, see the appropriate provider recommended settings documentation for your codec of choice.

Mezzanines are typically rendered in the native resolution and frame-rate that the source material was captured with. A common configuration using Apple ProRes would be as follows:

codec	Resolution	Bitrate	Framerates
Apple ProRes 422	3840x2160	340-1650 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	1920x1080	145-220 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	1280x720	20-60 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	640x480	8-12 Mbps	25, 29.97, 50, 59.94

4.2 Decide on Rendition Set

In modern video streaming use-cases, ABR streaming technologies use a rendition set to ensure every playback environment has an appropriate file to render. A rendition set is simply a grouping of the different transcodes of the same mezzanine file. During playback, ABR technologies detect the user's playback environment, available codecs, resolution and bandwidth; then selects the right segment from one of the video renditions to send to the video player. This reduces probability of buffering as only the segments of the matching video rendition is loaded. To create a proper rendition set, you should list the codecs and resolutions that exists for your target audience. Then you need to ascertain the proper bitrate for each codec and resolution, high enough to meet your quality goals but not exceed bandwidth availability.

In order to select optimum bitrates, determine your target user based upon target devices and user experience. For example, for OTT (Over-the-top) applications for Smart Televisions, typically a higher bitrate rendition is set as the default since resolution & bandwidth are reliably available. The table below shows an example of some possible renditions, resolutions, and framerates. This is not an exhaustive list nor is it meant to be used as best practices.

codec	Example Resolutions	Example Bitrates	Example Framerates
codec A	2160p	10 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	1080p	5 Mbps 3 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	720p	2 Mbps 1 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	480p	768Kbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
codec B	2160p	18 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	1080p	9 Mbps 5 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	720p	4 Mbps 2.5 Mbps	23.98, 24, 25, 29.97, 30, 50, 59.94, 60
	480p	1.5 Kbps 768 Kbps 380 Kbps	25, 29.97, 50, 59.94

4.3 Decide on Delivery Formats to support

In addition to defining resolutions and bitrates in your Rendition Set, the delivery format should also be considered. For most audiences, a format of a MP4 container with H.264 video and AAC audio is sufficient for most SD/HD videos. However if your video content warrants the need for advanced features, then you should also consider adding a rendition to your set that includes higher profile H.264, H.265, VP9, AAC-HI formats, so that users with environments supporting those features can enjoy them, while still providing a baseline experience via the H.264/AAC MP4 rendition set.

4.4 Create encoding profiles

After you have listed all the renditions that will be necessary, then create transcoding profiles (or templates depending on your tool) for each rendition in your transcoding software of choice (FFMPEG, Compressor, Vantage).

4.5 Transcode

If all the previous steps have been carefully thought out, then actual transcoding is comparatively easy. It is the act of taking the mezzanine file and sending it to each transcoding profile. To save time, this should be eventually scripted if possible since you may need to revisit this step each time you adjust your rendition set.

At the end of this process you should have a set of files that can be hosted at for web delivery. You will need to package these so that your chosen ABR technology can utilize them, this process is known as packaging.

4.6 Packaging your video files for ABR delivery

In order for an ABR technology to utilize your rendition set, each file needs to be properly segmented and exposed to the ABR technology following their spec. Segmentation refers to the breakdown of each file into timed chunks so ABR can smoothly switch renditions without causing a break in the video playback. Exposure of rendition set and segments are done through a manifest file - .m3u8 for HLS and .mpd for DASH. The manifest is simply a text file that serves as a directory for the renditions and includes details (resolution, codec, bitrate, etc) about each file available. This is necessary so that the ABR can find the correct segment for the users playback environment. A manifest can also contain references to other files that may be necessary for specific video features, such as .vtt files for subtitles and additional audio tracks for multi-lingual video.

Information about authoring a manifest for HLS can be found here, and authoring for DASH can be here.

5 Web App Structure

Media companies more and more are making their content and channels available where ever users are consuming media. More often than not this includes the web and native app platforms. As we see rise in media companies using web technologies to build cross platform apps (that stretch across the web and native app platforms) it becomes more important to define clarify the different approaches you can take to building "apps" with web technologies.

5.1 Web App Content Management Approaches

The easiest way to delineate between the types of web apps are by how we manage the content inside them. In the case of traditional applications and mobile apps, the entire application is downloaded from the store and then ran locally. In contrast, a website keeps all of its content on the server and then when the pages loader. It brings the entire application down into the browser. The same approaches are utilized with web content inside of the app space there often known as packaged web apps and hosted web apps.

5.1.1 The Packaged App

The Packaged App is a type of web app that distributed in its entirety to a user through a single download. Generally, we see packaged apps within App Markets. Packaged Apps have these characters:

- Downloaded in entirety through a store or market
- Generally offline by default, as all the code is living on the user's device
- Retrieve data from the Internet via services similar to native apps. Generally called via AJAX

Some Markets, such as Window Store and Amazon App Store enable you to submit packaged apps directly to the market without the need of any external packaging tools. These markets also expose additional APIs for packaged apps that aren't available to apps via the browser. Other markets, such as iOS App Store and Android Play require a third-party utility such as Cordova or Crosswalk to enable packaged web apps.

5.1.2 The Hosted App

Hosted Apps are web apps submitted to app markets that point to web content that remains on the webserver as appose to being packaged and submitted to the store. Hosted app's have these Characteristics:

- Have a thin layer of native code, usually containing a webview and/or an app manifest that is submitted to the market.
- Can run dynamic web content from a web server (thank asp.net PHP, etc.).
- By default have no functionality when device is offline (just like web).
- Hosted web apps are updated on the web server instead of pushing packages to the app store.

5.1.3 Progressive Web Apps

Progressive web apps or PWAs has become the de facto app format for the web. Just like hosted web apps, Progressive web apps load their content from the server when the app is opened. However, Progressive web apps also can store this web app locally through a new technology called service workers, which allows Progressive web apps to bridge across both the benefits of package web apps and hosted web apps. Some platforms run PWAs in the browser while others (like Android and Windows 10) run PWAs in a stand along app container.

PWAs are build around a common universal web app manifest that is being standardized by the W3C. The web manifest contains meta data about your application such as presentation preference, start URL, and even ratings and categories for store listing.

Example 2: common manifest:

```
{
 "lang": "en",
 "dir": "ltr",
 "name": "Donate App",
 "description": "This app helps you donate to worthy causes.",
 "short_name": "Donate",
 "icons": [{
  "src": "icon/lowres.webp",
  "sizes": "64x64",
  "type": "image/webp"
 },{
  "src": "icon/lowres.png",
  "sizes": "64x64"
 }, {
  "src": "icon/hd_hi",
  "sizes": "128x128"
 }],
 "scope": "/racer/",
 "start_url": "/racer/start.html",
 "display": "fullscreen",
 "orientation": "landscape",
 "theme_color": "aliceblue",
 "background_color": "red",
 "serviceworker": {
  "src": "sw.js",
  "scope": "/racer/",
  "use_cache": false
 },
 "screenshots": [{
  "src": "screenshots/in-game-1x.jpg",
  "sizes": "640x480",
  "type": "image/jpeg"
 },{
  "src": "screenshots/in-game-2x.jpg",
  "sizes": "1280x920",
  "type": "image/jpeg"
 }]
}
```

Learn more about building Progressive Web Apps at https://developers.google.com/web/progressive-web-apps/.

5.2 Web Based Platforms

Some platforms such as webOS are themselves a web-based environment, meaning all applications in that environment or built with the web and run as independent applications within the environment just like a web page may run inside of a web browser. Device manufacturers (of all kinds) use web engines as a basis for a web-based platform that will allow web apps to run as standalone applications.

The most common platform may be ChromeOS itself. ChromeOS is Chromium running on top of Linux that uses web apps for its primary app ecosystem (however it recently has been augmented to run Android apps as well). Another common Example is WebOS, which powers many LG tvs today, and runs web content as standalone apps.

5.3 Web App Containers

Delivering the web app on the platform is not a trivial task as each platform may have a different approach on how a web app is delivered to a user. Some platforms such as Xbox (Windows 10) and Play station give the web apps their own standalone container, which allows the app to be downloaded from a store and run independently of a browser. These apps are rendered with the web rendering engine native to the operating system, and generally have lower overhead than running in a browser, and often are granted access to OS level APIs not available in the browser.

5.3.1 Mobile App container

For a web app to be distributed through the store on platforms like iOS and Android, an app container needs to be used. Popular app containers such as Cordova and Crosswalk are used by web developers to deliver a web app through a store. This approach often adds over head to the application as the web app is rendered in a webview or the app may contain the entire browser runtime within each app container.

5.3.2 Chrome Embedded Framework and Electron

In desktop environments, we see the use of Electron and Chromium Embedded Framework (CEF) to deliver web apps as desktop applications. CEF and Electron uses the open source version of chrome as a runtime environment for the application. Much like we see one mobile platforms, this approach has additional overhead as instead of using the native rendering engine of the platform, chromium is delivered inside each application to render the web app. This gives the web app additional privileges to access native API’s, but at the same time raises many security concerns.

A. References

A.1 Informative References

[DASHIFIOP]

Guidelines for Implementation: DASH-IF Interoperability Points. DASH Industry Forum. 9 April 2018. Version 4.2. URL: https://dash-industry-forum.github.io/docs/DASH-IF-IOP-v4.2-clean.pdf

[ENCRYPTED-MEDIA]

Encrypted Media Extensions. David Dorwin; Jerry Smith; Mark Watson; Adrian Bateman. W3C. 18 September 2017. W3C Recommendation. URL: https://www.w3.org/TR/encrypted-media/

[HLS]

HTTP Live Streaming (HLS). R. Pantos; W. May. IETF. 15 October 2012. Internet Draft. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming

[MEDIA-SOURCE]

Media Source Extensions™. Matthew Wolenetz; Jerry Smith; Mark Watson; Aaron Colwell; Adrian Bateman. W3C. 17 November 2016. W3C Recommendation. URL: https://www.w3.org/TR/media-source/

[MPEGCENC]

ISO/IEC 23001-7:2016 Preview Information technology – MPEG systems technologies – Part 7: Common encryption in ISO base media file format files. ISO/IEC. February 2016. URL: https://www.iso.org/standard/68042.html

[MPEGDASH]

ISO/IEC 23009-1:2014 Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. ISO/IEC. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

[WAI-WEBCONTENT]

Web Content Accessibility Guidelines 1.0. Wendy Chisholm; Gregg Vanderheiden; Ian Jacobs. W3C. 5 May 1999. W3C Recommendation. URL: https://www.w3.org/TR/WAI-WEBCONTENT/

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™