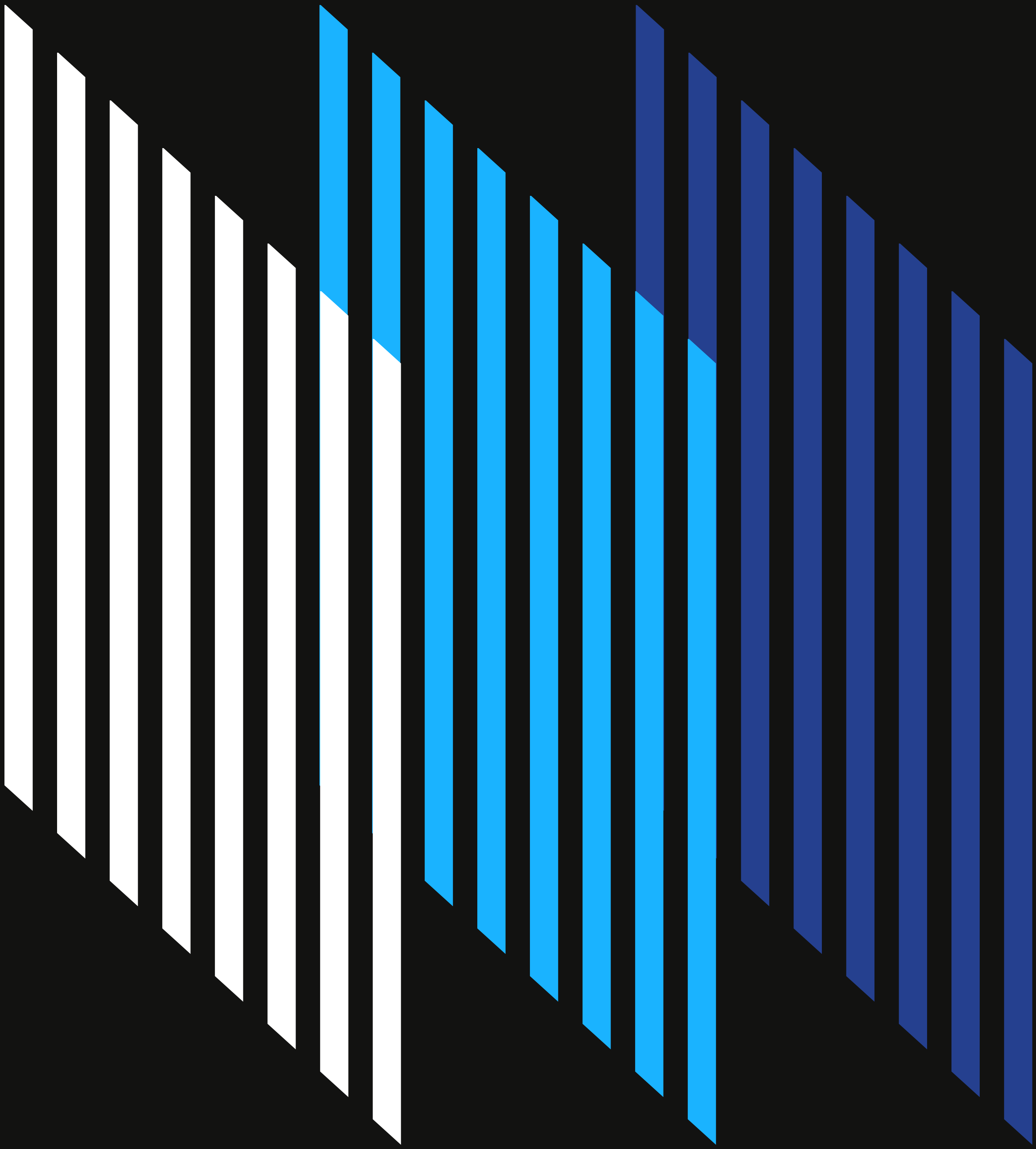
WHITEPAPER



Ultimate Guide to Container Formats

by Armin Trattnig

Table of Contents

Chapter 1

Welcome to the ultimate guide to Container Formats. This all-inclusive whitepaper covers the four most common container formats (as of 2019) and why it matters to you. The first chapter defines key terminology and how containers function within video players.

Container Basics & Terminology

Definition of Codecs

Codecs are an internet protocol used to store media signals in binary form. The goal of most codecs to compress the raw media signal in a lossy way, meaning that the compression is irreversible. Partial data is discarded from the raw media signal and approximations are made. The most common media signals are video, audio and subtitles. Specific examples of video codecs include: AV1, H.264, HEVC, and VP9; The most commonly used audio codecs are: AAC, MP3, and Opus. There are many different codecs for each media signal (as illustrated in fig 1).

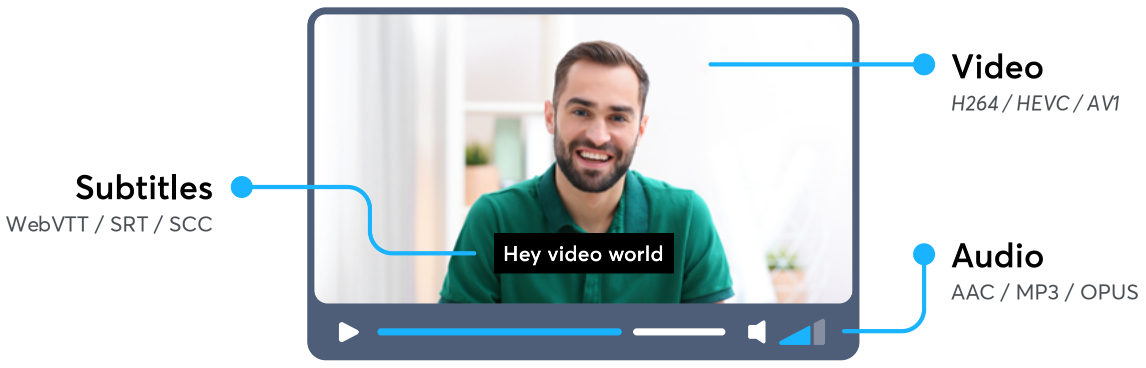


Figure 1

A single media signal is often called an Elementary Stream or more simply - Stream. The video development/broadcast industry will often use terms such as Codecs, Media, or H.264 (or H.2655) interchangeably with "Video Streams".

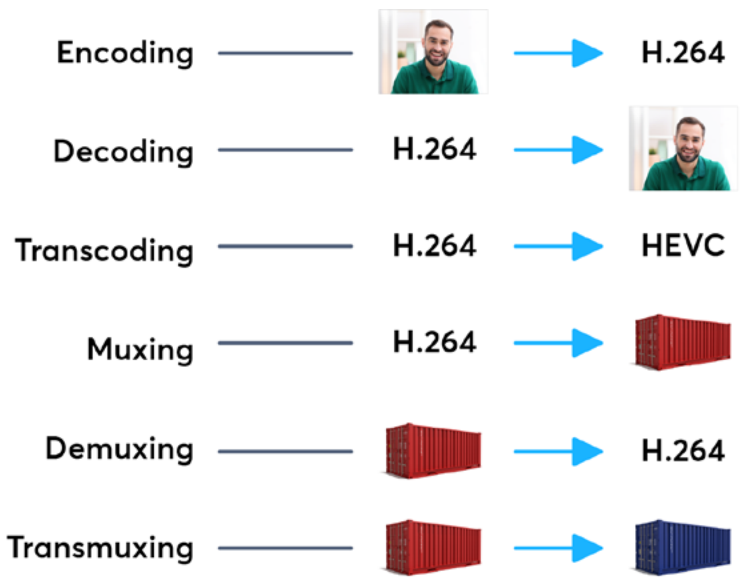
What is a media container?

Media Containers (also known as Container Formats) are a metafile format specification describing how different multimedia data elements (streams) and metadata coexist in files. Some of the elements specified by a container format are:

- **Stream Encapsulation -** Allowance of one or more media streams to exist in a single file.
- **Timing/Synchronization -** The container adds data on how the different streams in the file can be used together. Ex: Affixing correct timestamps to synchronize lip-movement in a video stream with sounds in the audio stream.
- **Seeking -** The container provides additional time-oriented information that determines a specific point which a viewer can jump to in a video file/stream. Ex: A viewer wants to watch a movie from a specific scene or would like to Skip the Intro of their favorite series

- **Metadata -** There are many types of metadata and one can easily add them to a video file using a container format. Ex: Language of an audio stream - subtitles are also sometimes considered as metadata.

The most common container formats are: MP4, MPEG2-TS and Matroska, and can represent different video and audio codecs. Each container format has its strengths and weaknesses defined by a video's compatibility, streaming and size overhead.



Data Conversion

The following terms are used to describe to most common transformations for media assets. A transformation reduces the size of the media data, to add compatibility or to enrich media data with additional data, like metadata or media data.

Figure 2

- **Encoding:** The process of converting a raw media signal to a binary file of a codec. For example encoding a series of raw images to the video codec H.264. Encoding can also refer to the process of converting a very high quality raw video file into a mezzanine format for simpler sharing & transmission - Ex: taking an uncompressed RGB 16-bit frame , with a size of 12.4MB, for 60 seconds (measured at 24 frames/sec) totalling 17.9GB - and compressing it into 8-bit frames with a size of 3.11MB per frame, which for the same video of 60 seconds at 24fps is 2.9GB in total. Effectively compressing the size of the video file down by 15GB!
- **Decoding:** The opposite of encoding; decoding is the process of converting binary files back into raw media signals. Ex: H.264 codec streams into viewable images.
- **Transcoding:** The process of converting one codec to another (or the same) codec. Both decoding & encoding are necessary steps to achieving a successful transcode. Best described as: decoding the source codec stream and then encoding it again to a new target codec stream. Although encoding is typically lossy, additional techniques like frame interpolation and upscaling increase the quality of the conversion of a compressed video format.
- **Muxing:** The process of adding one or more codec streams into a container format.
- **Demuxing:** Extracting a codec stream from a container format.
- **Transmuxing:** Extracting streams from one container format and putting them in a different (or the same) container format.
- **Multiplexing:** The process of interweaving audio and video into one data stream. Ex: An elementary stream (audio & video) from the encoder are turned into Packetized Elementary Streams (PES) and then converted into Transport Streams (TS).
- **Demultiplexing:** The reverse operation of multiplexing. This means extracting an elementary stream from a media container. E.g.: Extracting the mp3 audio data from an mp4 music video.

- **In-Band Events:** This refers to metadata events that are associated with a specific timestamp. This usually means that these events are synchronized with video and audio streams. E.g.: These events can be used to trigger dynamic content replacement (ad-insertion) or the presentation of supplemental content.

Containers in OTT Media Delivery

Media Encoding

Media delivery is frequently considered one of two ends of content process. These processes are defined by the containers described before, and are present anywhere that digital media exists. For example, if you record a video using your smartphone, the audio and video are both stored in one container file, e.g. an MP4 file. Another example of containers in the wild is media streaming over the internet. Here, containers are the main entity of media that are handled from end-to-end. At one end of a content generation cycle, a packager multiplexes encoded media data (content) into containers, which are then transported to the other end via the network of the requesting clients' device. Lastly, the client selects which packets/segments are demuxed within the network, the content is decoded, and presented to the end user. Process illustrated below in fig 3:

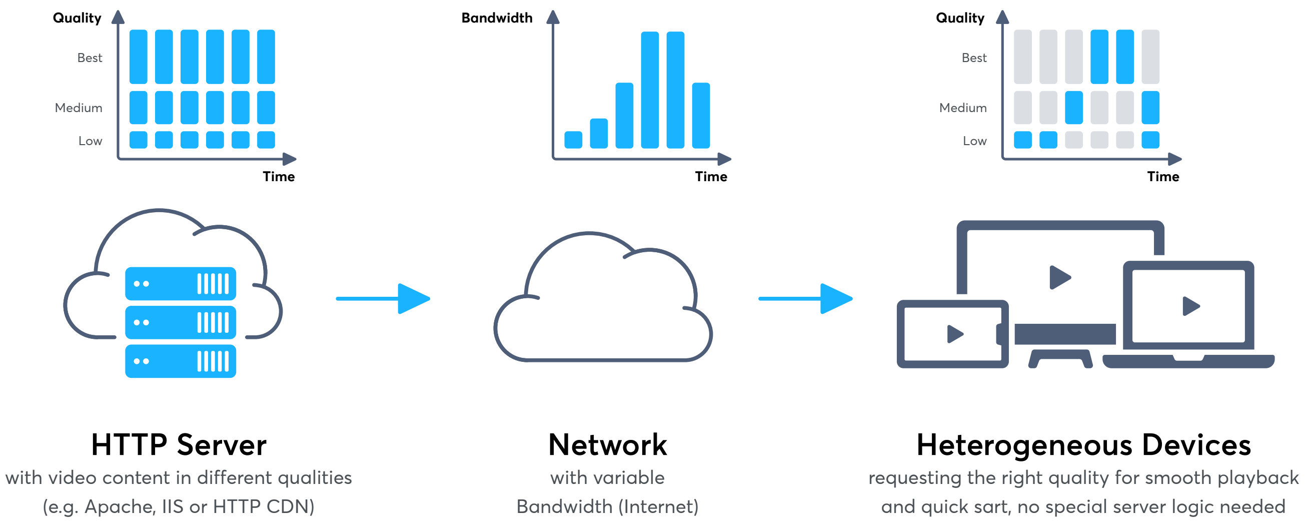

Quality
Best
Medium
Low
Time
Bandwidth
Time
Quality
Best
Medium
Low
Time
HTTP Server
with video content in different qualities
(e.g. Apache, IIS or HTTP CDN)
Network
with variable
Bandwidth (Internet)
Heterogeneous Devices
requesting the right quality for smooth playback
and quick sart, no special server logic needed

Figure 3

Handling Containers in the Player

Media Decoding

The other end of a content process is formally considered "the client-side," or in layman's terms, a video player. For the content to appear on a user's device, the video player needs to extract some basic information about the media from a container; for example – an individual segment's playback time, video duration and involved codecs.

Additionally, there is often metadata present in a container that most browsers would not extract or handle out-of-box. Some examples of this are CEA-608/708 captions or inband events (as seen in fig 4: EMSG boxes of fMP4 segments), where the player has to parse relevant data from the media container format file, keep track of a timeline and process the data at the correct time within the content (like displaying the right captions at the right time). This requires the player implementation to have desired handling in place.

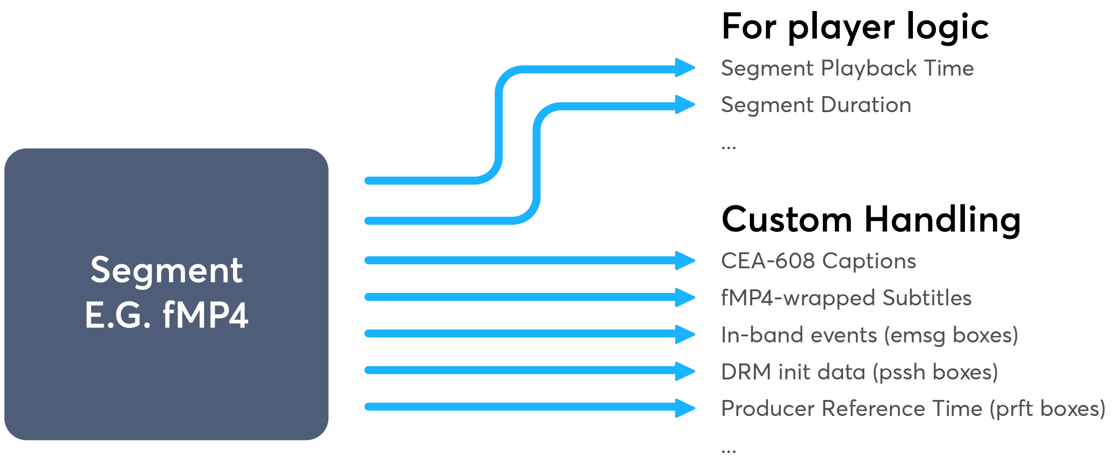


Figure 4

Client-side Transmuxing

In scenarios where a simple encode-decode cycle doesn't work, the process of transmuxing comes into play. This is most commonly seen within various browsers that often lack support for certain container formats. A prime example of this issue is that web browsers, Chrome, Firefox, Edge and Internet Explorer not (properly) supporting the MPEG-TS container format. The MPEG (Motion Picture Experts Group) Transport Stream format was specifically designed for Digital Video Broadcasting (DVB) applications. You can find more details on this format in chapter three (p. X). Since MPEG-TS is very commonly used container format, the only solution is to convert media from MPEG-TS to a container format that these browsers do support (ex: fMP4). This conversion step can take place at the client directly before the content is forwarded to the browser's media stack for demuxing and decoding. This process includes demultiplexing the MPEG-TS and then re-multiplexing the elementary streams to fMP4. This process is visualized below in fig 5: Transmuxing Packets.

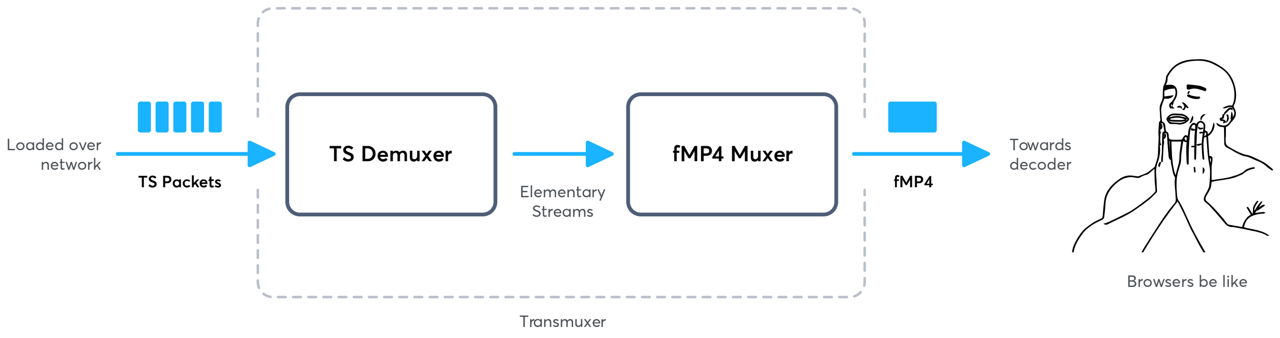


Figure 5

Chapter 2

With the basic terminology behind container formats covered, the following chapter will dive into specific container formats, namely, MP4 and CMAF. This includes additional terminology around Boxes as well as the use and application of fragmentation or chunking of files.

MP4 - Overview of Standards

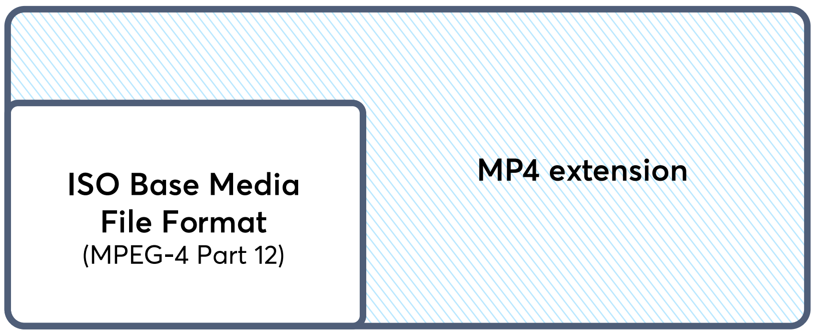
Base Formats Terminology

- **FileTypeBox (Ftyp) -** A four-letter code found within the structure of a video file to identify the "type" of encoding being using, it's "compatibility", or its "intended usage".
- **TrackBoxes -** Contains either audio or video track.
- **Movie Box -** Contains the box header information and the TrackBoxes.
- **Movie Extends Box -** Contains header and information to signal that the movie continues in fragments. Fragments usually only contain a fraction of the whole movie.
- **Movie Fragmented Box -** Contains movie fragment header data and track fragment data.

MPEG-4 Part 14

MPEG-4 Part 14 (MP4) is one of the most commonly used container formats and often has a .mp4 file ending. It is used for Dynamic Adaptive Streaming over HTTP (DASH) and can also be used for Apple's HLS streaming. MP4 is based on the ISO Base Media File Format (MPEG-4 Part 12), which is based on the QuickTime File Format.

MPEG-4 Part 14 - MP4 File Format



MPEG stands for Moving Pictures Experts Group and is a cooperation of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). MPEG was initially formed to create and maintain a set of standards for audio and video compression and transmission. MPEG-4 however, applies specifically towards the standards of coding of audio-visual (AV) objects.

The MP4 container format supports a wide range of codecs, most commonly: H.264 or HEVC for video and Advanced Audio Coding (AAC) for audio. AAC was designed as the successor to the famous MP3 codec.

ISO Base Media File Format

ISO Base Media File Format (ISOBMFF, MPEG-4 Part 12) is the base of the MP4 container format. ISOBMFF is a standard that defines time-based multimedia files. Time-base multimedia usually refers to audio and video, often delivered as a steady stream. It is designed to be flexible and easy to extend, enabling interchangeability, management, editing and presentability of multimedia data.

The base component of ISOBMFF are boxes, otherwise known as atoms. These boxes are defined using classes and an object oriented approach - there are currently hundreds of different class types, please refer to the publicly available standards lists by the International Organization of Standards (ISO) for relevant class information. Using inheritance, all boxes extend a base class Box and can be made specific in their purpose by adding new class properties. An example of this could be a general class 'car' which then has a specific subclass which is 'SUV', the 'SUV' class inherits all the properties of 'car' plus defines some new ones. In the same way a box in the context of MP4 ISOBMFF is a general class which then a MovieBox is inheriting all the properties of ISOBMFF and then defining some specific properties of a MovieBox. The structure of an ISOBMFF is defined below in fig 7:

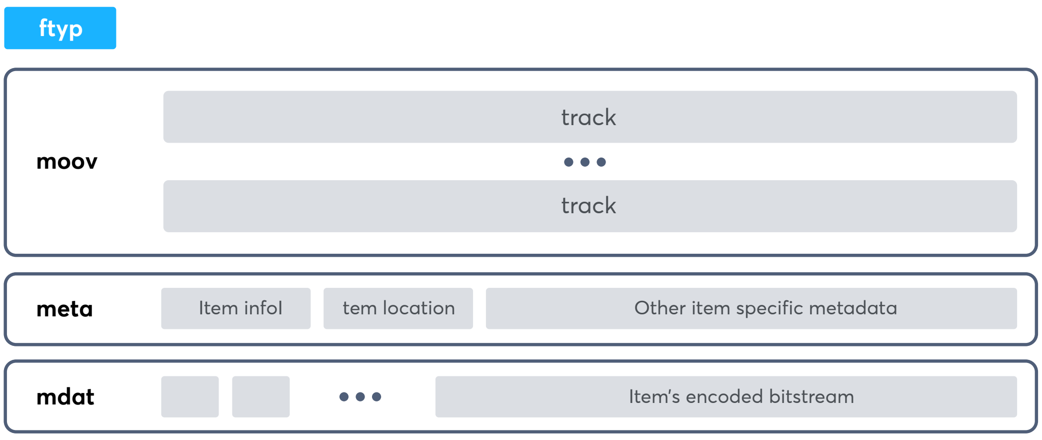
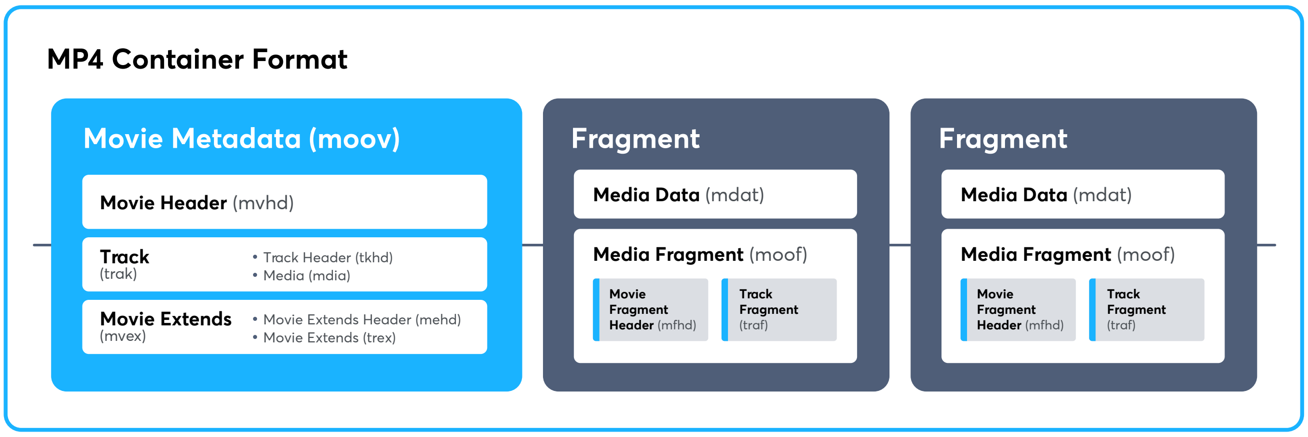


Figure 7

- **Base Class - Example FileTypeBox:**

A FileTypeBox (ftyp) is used to identify the purpose and usage of an ISOBMFF file and is most commonly applied in the beginning of a file. A box can also have children and form a tree of boxes (illustrated above). For example: a MovieBox (moov) can have multiple TrackBoxes (track). A track in the context of ISOBMFF is a single media stream and typically references its binary codec data. E.g. a MovieBox contains a track box for video and one track box for audio. The binary codec data can be stored in a Media Data Box (mdat).

Fragmented MP4 (fMP4)



The MP4 container format has the capability to split a video file into multiple fragments (fMP4). The advantage of fragmenting a video file is that streaming protocols like DASH or HLS will allow player softwares to download only the fragments that the viewer wants to watch – therefore saving bandwidth and loading times. A fragmented MP4 file consists of a standard MovieBox with TrackBoxes to signal which media streams are available. A Movie Extends Box (mvex) is used to signal that the movie is continued in the fragments. Another advantage is that fragments can be stored in different files. A fragment consists of a Movie Fragment Box (moof), which is very similar to a Movie Box (moov). A moof contains the information about the media streams within a single fragment. E.g. it contains timestamp information for 10 seconds of video, which are stored within a fragment. Each unique fragment has its own Media Data (mdat) box.

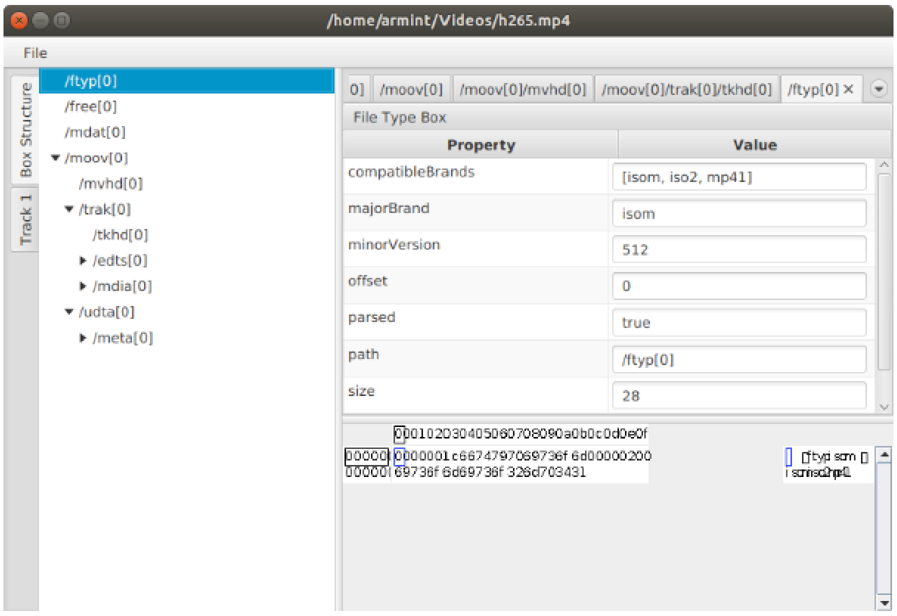
Debugging (f)MP4 files

Viewing the boxes (atoms) of an (f)MP4 file is often necessary to discover bugs and other unwanted configurations of specific boxes. To view a list of the latest media file releases (by operating system) we recommend the following tools:

- **MediaInfo (https://mediaarea.net/en/MediaInfo/Download)**
- **ffprobe, which is part of the ffmpeg binaries (https://ffbinaries.com/downloads)**

Unfortunately those tools do not display the box structure of an (f)MP4 file. For this information, we recommend the following tools:

- **Boxdumper (https://github.com/l-smash/l-smash)**
- **IsoViewer (https://github.com/sannies/isoviewer) – illustrated in fig 9.**
- **MP4Box.js (http://download.tsi.telecom-paristech.fr/gpac/mp4box.js/filereader.html)**
- **Mp4dump (https://www.bento4.com/)**



Property	Value
compatibleBrands	[isom, iso2, mp41]
majorBrand	isom
minorVersion	512
offset	0
parsed	true
path	/ftyp[0]
size	28

```
        000102030405060708090a0b0c0d0e0f
00000   0000001c667479706973 6f 6d00000200        ftyp isom
00000   69736f 6d69736f 326d703431                isomiso2mp41
```

Figure 9: Isoviewer Screenshot of how (f)MP4 files are sectioned

CMAF

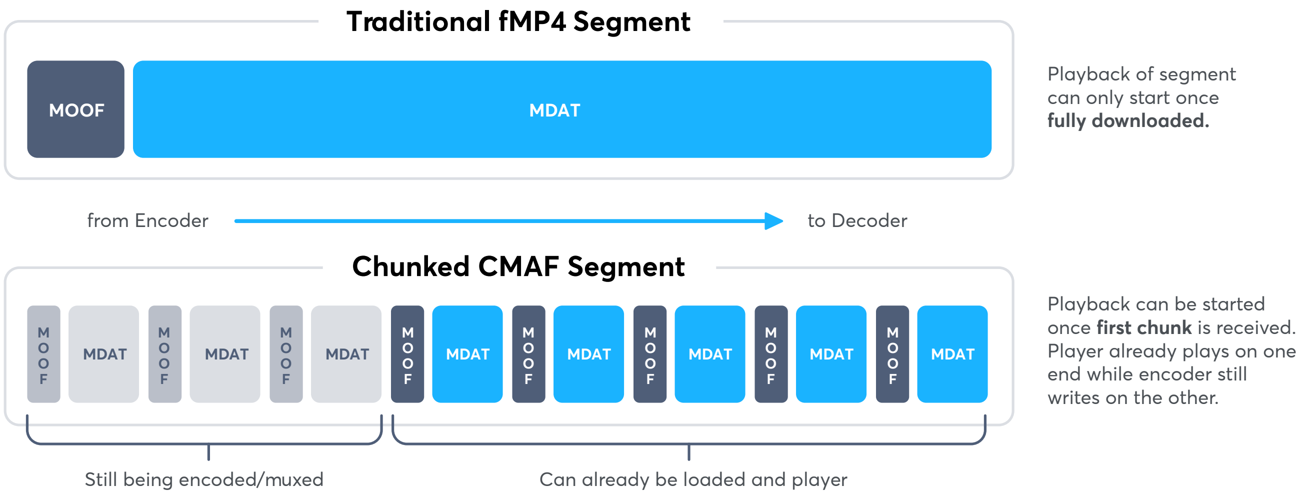
MPEG-CMAF (Common Media Application Format)

Serving every platform as a content distributor can prove to be challenging as some platforms support limited container formats. To distribute a specific piece of content it might be necessary to produce multiple copies of the content in different container formats, e.g. MPEG-TS and fMP4. This results in additional infrastructure, content creation and storage costs for hosting multiple copies of the same piece of content. In addition, it decreases the efficiency of CDN caching. MPEG-CMAF aims to solve these problems by converging to a single existing container format for OTT media delivery. A major benefit of CMAF is that it's closely related to fMP4; therefore the transition from fMP4 to CMAF can be considered very low effort. Furthermore, Apple's involvement in CMAF reduces (if not completely eliminates) the necessity of muxing content in MPEG-TS in order to serve Apple devices. In other words, CMAF is a movement towards container standardization can be used on every device and every browser.

MPEG-CMAF also yields improvements to the interoperability of Digital Rights Management (DRM) solutions by the use of MPEG-CENC (Common Encryption). It is possible to encrypt a piece of content once and use it across all the different state-of-the-art DRM systems. However, there is no standardized encryption scheme; as a result, there are many competing options on the market, like Widevine or PlayReady. While most DRM systems are not compatible with each other, the DRM industry and providers are slowly moving to one encryption scheme, the Common Encryption format.

Chunked CMAF

An interesting feature of MPEG-CMAF is the ability to encode segments in so-called CMAF chunks. Chunked encoding of content with media file delivery using HTTP chunked transfer encoding enables lower latencies in live streaming uses cases than previously possible. The difference between a traditional (f)MP4 vs a Chunked CMAF segment is illustrated below in fig 10.


Traditional fMP4 Segment
MOOF
MDAT
Playback of segment can only start once **fully downloaded.**
from Encoder
to Decoder
Chunked CMAF Segment
MOOF
MDAT
Playback can be started once **first chunk** is received. Player already plays on one end while encoder still writes on the other.
Still being encoded/muxed
Can already be loaded and player

In traditional (f)MP4 the whole segment has to be fully downloaded to be played out. With chunked encoding, any completely loaded chunks of the segment can be decoded and played, while loading the rest of the segment. Hereby, the achievable live latency is no longer dependent on the segment duration as muxed chunks of an incomplete segment can be loaded and played at the client.

With (f)MP4, MPEG-CMAF, and Chunked CMAF have been covered, you're ready for the complex container functions: MPEG-TS & Matroska (WebM)

Chapter 3

MPEG Transport Streams (MPEG-TS) & Muxing

MPEG Transport Stream was standardized in MPEG-2 Part 1 and specifically designed for Digital Video Broadcasting (DVB) applications. Compared to its counterpart, the MPEG Program Stream, which was aimed for storing media and found its use in applications like DVD, the MPEG Transport Stream (MPEG-TS) is a more transport-oriented format. An MPEG-TS consists of small individual packets built to increase resilience against and minimize implications of corruption or loss. Furthermore, the format includes Forward Error Correction (FEC) techniques that which catch and correct transmission errors at the receiver. Simply put, the MPEG-TS format was designed for the use on lossy transport channels.

Muxing in MPEG-TS Containers

So, what is the process behind implemented lossy transport channels – Muxing the stream! The process begins with an Elementary Stream (ES) from an encoder converting into Packetized Elementary Stream (PES) by adding a PES Header. The newly added PES header typically includes a stream identifier, the PES packet length and information about media timestamps, among other things – like padding. Next, the PES is split up into 184 byte chunks and turned into a Transport Stream (TS) by adding a 4 byte header to each chunk. The resulting TS consists of packets with a fixed length of 188 bytes. Each TS packet's header carries the same Packet Identifier (PID) as the packet from the elementary stream it originated. In short: Muxing = ES ----> PES ---> TS (illustrated in fig 11)

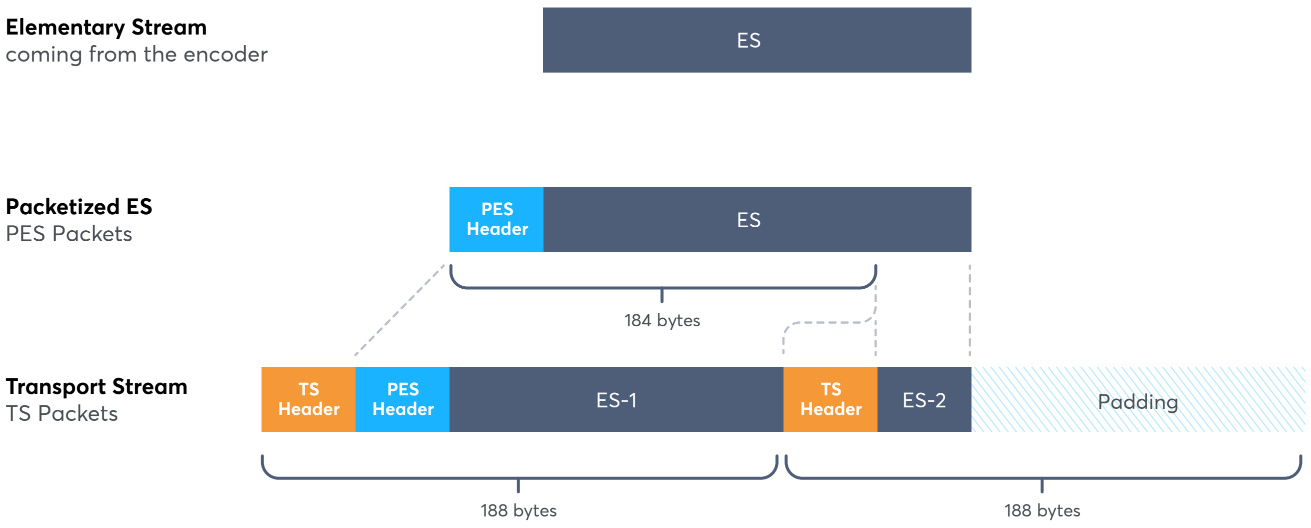
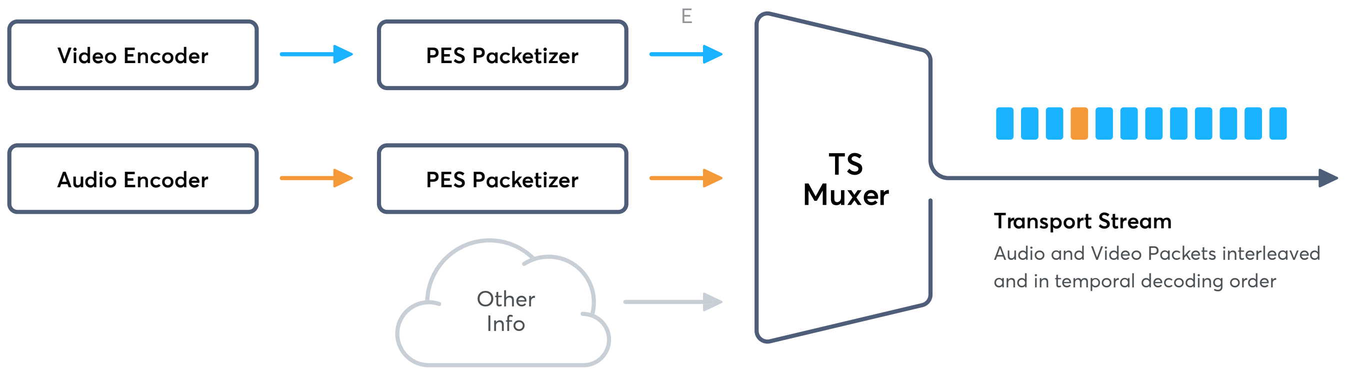


Figure 11

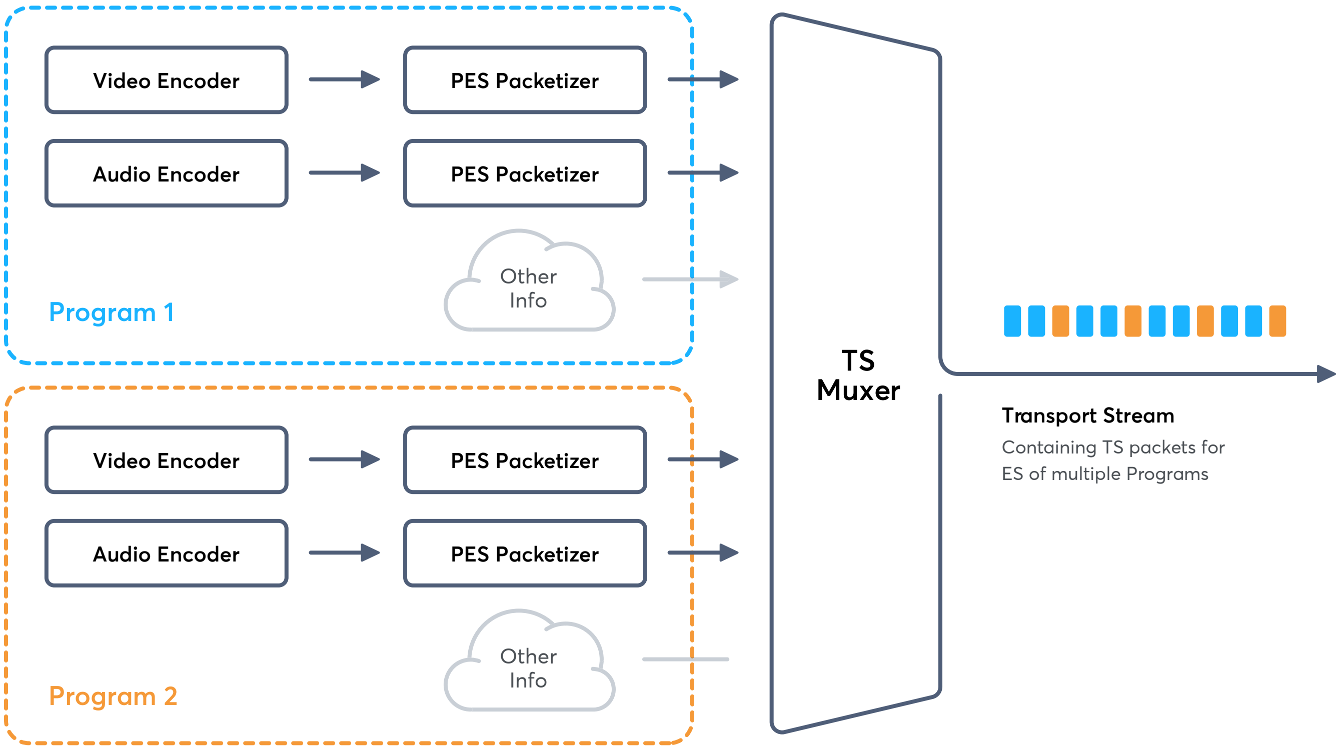
Muxing multiple Elementary Streams

A single elementary stream represents either audio or video content. Most video elementary streams are accompanied by at least one audio elementary stream. These correlated ES's are muxed into the same transport stream with separate PIDs for each ES and it's packets. Illustrated below in fig 12:



Muxing multiple Programs

The most complex variation of muxing a stream is the process of muxing multiple programs. With MPEG-TS, a program is a set of related elementary streams that belong together, e.g. video and the matching audio. A single transport stream can carry multiple programs, e.g. a different TV channel. This process can be seen in Fig 13:

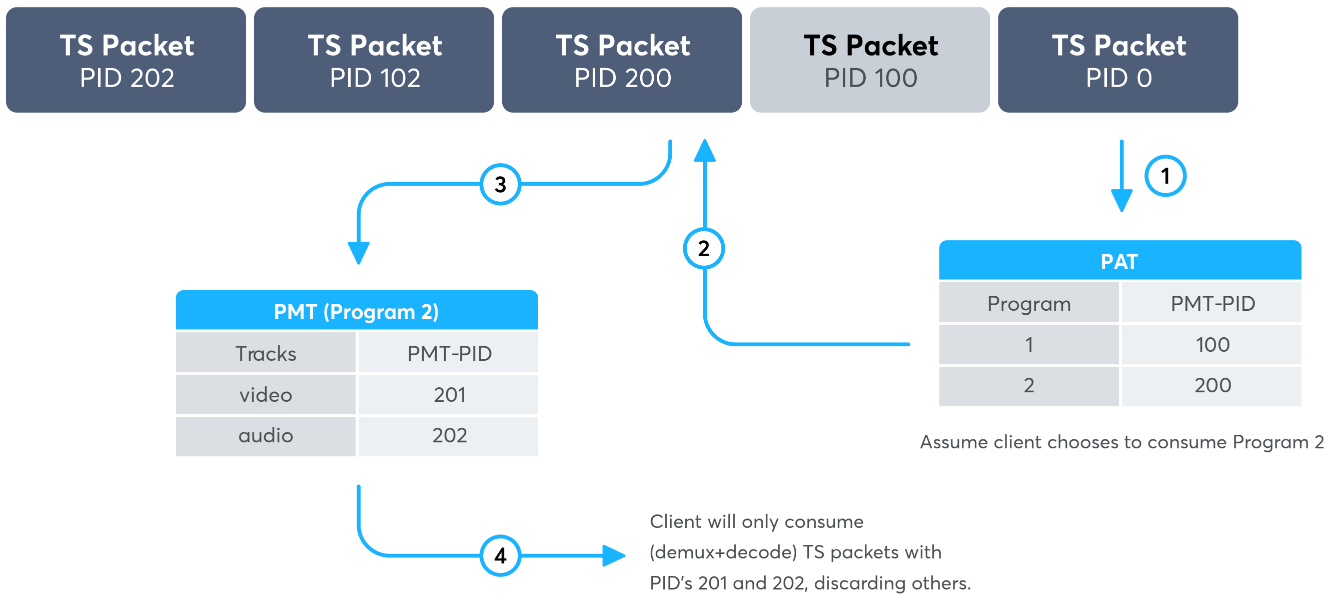


Program Association

From a low-level perspective a transport stream is just a sequence of 188 byte long TS packets. As previously mentioned, there can be many programs with multiple elementary streams, but a client is usually only capable of presenting one program at a time. It must therefore determine which packets to consume and which to discard upon receiving the transport stream. For this purpose there are two kinds of special packets:

- **Program Association Table (PAT) - PAT packets have the reserved PID of 0 and contains the PIDs for Program Map tables of all programs within the transport stream.**
- **Program Map Table (PMT) - The PMT represents a single program and contains the PIDs for all elementary streams of the program**

The process by which PATs and PMTs determine which packets to consume takes four steps, as defined in fig 14 and elaborated below:



1. Inspect the TS Packet with PID 0, which contains the PAT.
2. Find the PMT-PID of the Program the player should play back in the PAT (in this example: 200).
3. Get the TS Packet with the relevant PMT-PID, which contains the PMT (200).
4. The PMT contains the PID for all the media tracks, which are part of the Program to play.

A client receiving the transport stream would first read the PAT packet it received and pick a program to be played depending on a user's selection. The client pulls the selected program's PMT (including the ESs and PIDs) from the PAT. Then, the client filters for the specified PIDs, each representing a separate ES of the chosen program, and consumes (demuxes), decodes and plays them for the user.

OTT-specific aspects and Conclusion

MPEG-TS is very broadcast-oriented, in OTT, however, there are additional special considerations. OTT clients have network connections that are unstable and do not have guaranteed bandwidth;

this requires that only content that will be played should be loaded. Given that a client is only able to present a single program at a time, having multiple programs in the same transport stream and loading them would be a waste of bandwidth. As a result, OTT would never have multiple programs in one transport stream. Similar arguments regarding multiplexing elementary apply for multi-audio content streams, i.e. all of them should be multiplexed into their own transport stream.

Given it's diverse application, MPEG-TS is still widely used for OTT, especially when working within the Apple ecosystem.

A downside of MPEG-TS is that due to the small packet size and all the packet headers the overhead is higher compared to FMP4.

MPEG-TS is transport-oriented and includes considerations for lossy communication channels which is not exactly a good fit for HTTP-based media delivery where transport loss is already handled by the network stack. To debug and inspect MPEG-TSs, try the following tools:

http://www.digitalekabeltelevisie.nl/dvb_inspector/ (GUI, open source).
http://thumb.co.il/ (GUI/Web, open source).
http://dvbsnoop.sourceforge.net/ (CLI, open source).
https://github.com/tsduck/tsduck (CLI, open source).
The final container format necessary to complete this guide is the free and Open-Standard, Matroska (WebM).

Matroska (Webm)

Overview

Matroska is a binary, free and open-standard container format based on Extensible Binary Meta Language (EBML), an eXtensible Markup Language (XML) - a method of representing information in nested tags. This fact makes the standard easy to extend and can support virtually any codec.

WebM

WebM is based on the Matroska container format. The development was mainly driven by Google to provide a free and open alternative to MP4 and MPEG2-TS to be used on the web. It was also developed to support Google's open and free codecs like: VP8, VP9 for video and Opus and Vorbis for audio. It is also possible to use WebM with DASH to stream VP9 and Opus over the web.

Debugging Matroska/Webm

The best tool to debug and view the contents of a Matroska or WebM file is mkvinfo (https://mkvtoolnix.download/).

Thanks for reading our Ultimate Guide to Container Formats Whitepaper!

Want to learn more?

Check out our resources page: bitmovin.com/resources

Or contact us at:

Bitmovin Inc
41 Drumm Street
San Francisco | CA 94105 | USA
US: 1-833-248-6686
Europe: +43 463-203-014
Sales@bitmovin.com