FREE eBook

LEARNING video

Free unaffiliated eBook created from
Stack Overflow contributors.

#video

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: video

It is an unofficial and free video ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official video.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with video

Remarks

Video playback is front and center in a large range of modern solutions, with the software and standards evolving rapidly. To understand this field, you must first understand the multiple aspects involved in working with video:

- Raw color information captured from the physical world is commonly *encoded* using a codec - an algorithm whose purpose is to represent this data in a compressed form, often sacrificing some visual detail in favor of greater compression.
- For playback, the inverse algorithm is executed - the data is *decoded* to once again become raw color information that can be supplied to an output device (e.g. a monitor).
- Between encoding and decoding, the compressed data is *packaged* for storage, which may involve combining tracks of different types into a single file or the segmentation of content into a large number of small segments.
- The video is *delivered* to the end-user's device using a delivery technology, which can be as simple as a file download over HTTP or considerably more complex, involving live feedback from the network infrastructure and automatic adaptation of quality levels.
- Premium content is usually *encrypted* before packaging and can only be played back in a player equipped with a DRM technology that ensures decryption key security during use and actively protects against output capture.

While the visual portion is obviously dominant in visibility, audio and text also play a key role in media presentations, providing multilanguage features that make content accessible to a wide audience. In most workflows, audio and text tracks are handled in a manner equivalent to video tracks, being encoded, decoded, packaged and delivered along the same lines.

All of these aspects - and more - must be cared for in a modern solution, ensuring a pleasant experience for the end-users.

Examples

Understanding stand-alone media files

The sample content used here is Tears of Steel, by Blender Foundation. Specifically, we will use the download titled "HD 720p (~365MB, mov, 2.0)". This is a single file that ends with the extension "mov" and will play in just about any modern media player.

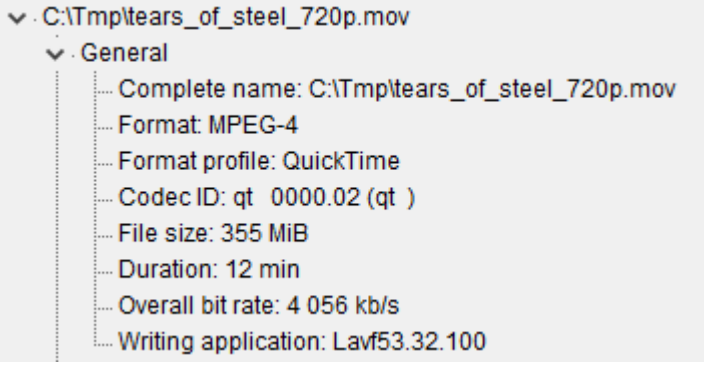
Note that the download page offers subtitles as separate SRT file downloads. In this sample content, there are no subtitles delivered together in the same file. We therefore leave subtitle analysis out of scope of this example.

An easy way to analyze various media files is using the tool/library MediaInfo. While the analysis functionality showcased here uses the GUI for simplicity, all the features are also available via the

MediaInfo API.

By opening this file in the MediaInfo GUI and switching to the tree view, you will see three sections: General, Video and Audio. The first contains basic information about the file, whereas the remaining two each describe a media track found in this file. Let's examine the most relevant information in each section of the output.

General

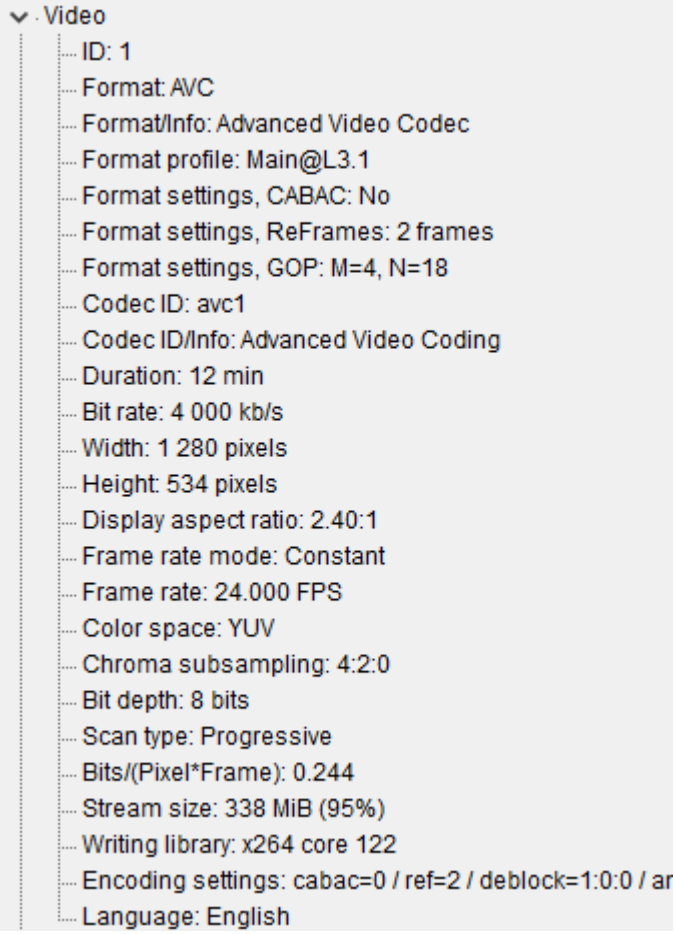


The first parameters of interest are *Format* and *Format profile*. The first indicates that the **packaging format** is from the MPEG-4 standards suite. MPEG-4 defines the *ISO Base Media File Format* and the *MP4 packaging format*. Furthermore, Apple has created their own specification that derives from these, named in MediaInfo as the "QuickTime" profile.

> Note: Be careful not to confuse MP4 and MPEG-4 - the former refers to a specific packaging format in the MPEG-4 suite of international standards, which also includes video and audio codecs. This can lead to confusion, so avoid using the term MPEG-4 when referring to anything other than the full set of standards.

All packaging formats based on the ISO Base Media File Format, defined in the MPEG-4 standards family, are very similar and can often be processed by the same tools, with their differences being largely a matter of custom vendor extensions that can often be safely ignored. Thus, we can expect the sample video here to be highly compatible with all modern video players.

Video



The most crucial detail about the video track is the codec used to transform raw color data into a compressed form. The name of the codec is provided by the *Format* parameter.

AVC is also known as H.264 and it is the video codec that today is the most widespread, supported on practically all modern devices and software platforms. A video track encoded using AVC is sure to play on just about any player.

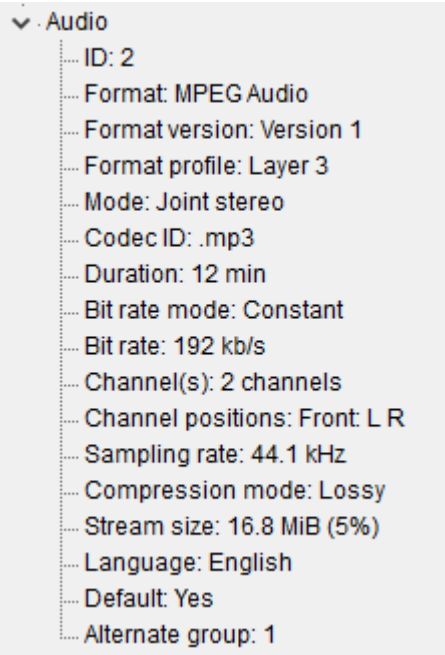
Codecs often have multiple *profiles* that allow codec functionality to be divided into tiers, enabling evolution of the technology in a controlled fashion. The *Format profile* parameter indicates that this video uses the Main profile. This profile is relatively uncommon, as just about all modern devices support the High profile, which offers greater compression efficiency.

The quality of the video track is often of paramount importance. Here we see the critical factors expressed by the *Bit rate*, *Width* and *Height* parameters. The latter two hint that this is meant to be a 720p video track, which is considered a lower-end HD quality. The picture is actually shorter vertically than the standard 720p frame of 1280x720 pixels.

The bit rate measures the amount of data that the compressed form of the video stream occupies, on average, for every second of playback. This is a crucial parameter for optimization, as the amount of delivered data is a major source of cost in large-scale video solutions.

The above data points about video quality are simply facts we obtain from analysis - any judgements on the appropriateness of these parameters is a topic that would need far more analysis and is tackled by separate topics in this documentation category, as are many other fine points of working with video tracks.

Audio



Once again, knowing the codec used to encode the audio data is of critical importance. This is expressed by the *Format* and *Format profile* parameters. "MPEG Audio Layer 3" is more commonly known as MP3 and it is a universally supported audio format that can be expected to play everywhere.

As with video, audio quality parameters are the second most important data points, expressed primarily by the *Bit rate* parameter.

Summary of analysis

The content is packaged using a very popular packaging format, built upon the MPEG-4 standards suite. It is encoded using universally adopted video and audio codecs. From this it is clear that the video is meant to be easily accessible to every viewer - compatibility and availability were key for its authors.

The use of MP3 shows the age of the example content, as it is no longer considered up to par with modern competitors - instead, AAC (Advanced Audio Coding) is the breadwinner in the field of audio codecs.

The same can be said about the use of the H.264 Main profile. It is very rare that any H.264 profile besides High is used, given that almost all decoders support it, enabling everyone to take advantage of the improved efficiency enabled by High profile features.

The bitrates used are slightly higher than expected for today's environment. This may be explained by the authors' desire for high quality or simply by the limitations of the encoders that were available when the content was created.

More

Other useful tools for media file analysis are FFprobe, which is part of the FFmpeg software package, and the Bento4 tools for working with MP4 files. Both are also available in library form. They are capable of more low-level analysis than MediaInfo, in situations where you need to examine individual elements that make up media files.

Read Getting started with video online: https://riptutorial.com/video/topic/5690/getting-started-with-video

Chapter 2: Understanding a media presentation

Remarks

Video playback is front and center in a large range of modern solutions, with the software and standards evolving rapidly. To understand what makes up a media presentation, you must first understand the multiple aspects involved in working with video:

- Raw color information captured from the physical world is commonly *encoded* using a codec - an algorithm whose purpose is to represent this data in a compressed form, often sacrificing some visual detail in favor of greater compression.
- For playback, the inverse algorithm is executed - the data is *decoded* to once again become raw color information that can be supplied to an output device (e.g. a monitor).
- Between encoding and decoding, the compressed data is *packaged* for storage, which may involve combining tracks of different types into a single file or the segmentation of content into a large number of small segments.
- The video is *delivered* to the end-user's device using a delivery technology, which can be as simple as a file download over HTTP or considerably more complex, involving live feedback from the network infrastructure and automatic adaptation of quality levels.
- Premium content is usually *encrypted* before packaging and can only be played back in a player equipped with a DRM technology that ensures decryption key security during use and actively protects against output capture.

While the visual portion is obviously dominant in visibility, audio and text also play a key role in media presentations, providing multilanguage features that make content accessible to a wide audience. In most workflows, audio and text tracks are handled in a manner equivalent to video tracks, being encoded, decoded, packaged and delivered along the same lines.

All of these aspects - and more - impact the composition, formatting and use of a media presentation. They must be understood as a whole in order to make effective use of media technologies.

Examples

Understanding stand-alone media files

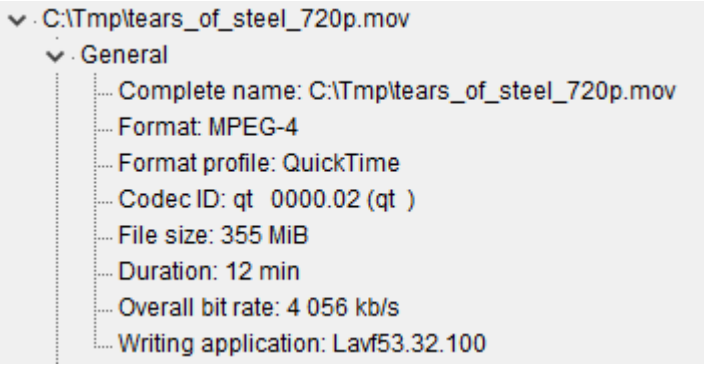
The sample content used here is Tears of Steel, by Blender Foundation. Specifically, we will use the download titled "HD 720p (~365MB, mov, 2.0)". This is a single file that ends with the extension "mov" and will play in just about any modern media player.

Note that the download page offers subtitles as separate SRT file downloads. In this sample content, there are no subtitles delivered together in the same file. We therefore leave subtitle analysis out of scope of this example.

An easy way to analyze various media files is using the tool/library MediaInfo. While the analysis functionality showcased here uses the GUI for simplicity, all the features are also available via the MediaInfo API.

By opening this file in the MediaInfo GUI and switching to the tree view, you will see three sections: General, Video and Audio. The first contains basic information about the file, whereas the remaining two each describe a media track found in this file. Let's examine the most relevant information in each section of the output.

General

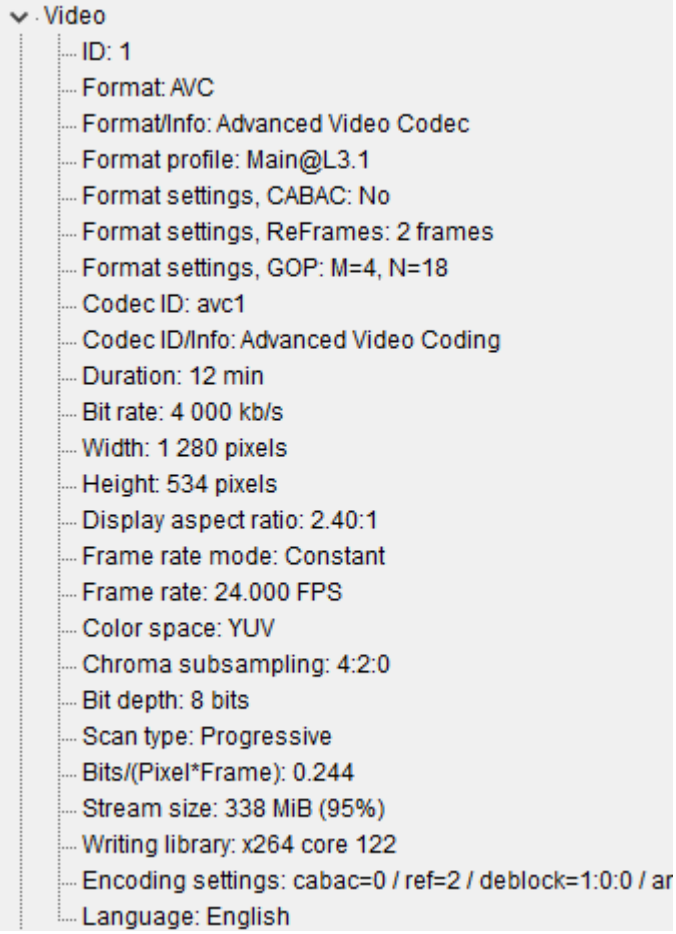


The first parameters of interest are *Format* and *Format profile*. The first indicates that the **packaging format** is from the MPEG-4 standards suite. MPEG-4 defines the *ISO Base Media File Format* and the *MP4 packaging format*. Furthermore, Apple has created their own specification that derives from these, named in MediaInfo as the "QuickTime" profile.

> Note: Be careful not to confuse MP4 and MPEG-4 - the former refers to a specific packaging format in the MPEG-4 suite of international standards, which also includes video and audio codecs. This can lead to confusion, so avoid using the term MPEG-4 when referring to anything other than the full set of standards.

All packaging formats based on the ISO Base Media File Format, defined in the MPEG-4 standards family, are very similar and can often be processed by the same tools, with their differences being largely a matter of custom vendor extensions that can often be safely ignored. Thus, we can expect the sample video here to be highly compatible with all modern video players.

Video



The most crucial detail about the video track is the codec used to transform raw color data into a compressed form. The name of the codec is provided by the *Format* parameter.

AVC is also known as H.264 and it is the video codec that today is the most widespread, supported on practically all modern devices and software platforms. A video track encoded using AVC is sure to play on just about any player.

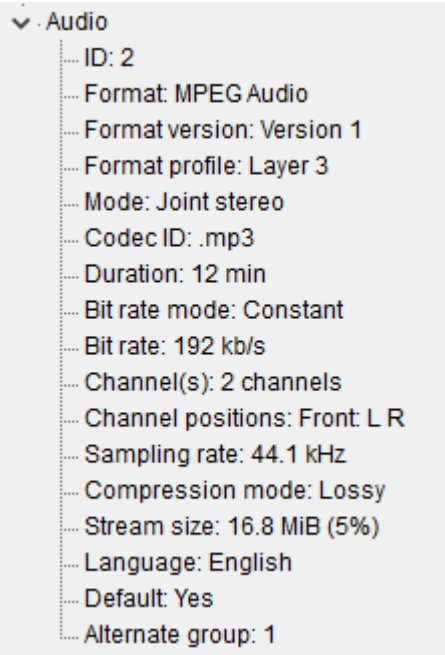
Codecs often have multiple *profiles* that allow codec functionality to be divided into tiers, enabling evolution of the technology in a controlled fashion. The *Format profile* parameter indicates that this video uses the Main profile. This profile is relatively uncommon, as just about all modern devices support the High profile, which offers greater compression efficiency.

The quality of the video track is often of paramount importance. Here we see the critical factors expressed by the *Bit rate*, *Width* and *Height* parameters. The latter two hint that this is meant to be a 720p video track, which is considered a lower-end HD quality. The picture is actually shorter vertically than the standard 720p frame of 1280x720 pixels.

The bit rate measures the amount of data that the compressed form of the video stream occupies, on average, for every second of playback. This is a crucial parameter for optimization, as the amount of delivered data is a major source of cost in large-scale video solutions.

The above data points about video quality are simply facts we obtain from analysis - any judgements on the appropriateness of these parameters is a topic that would need far more analysis and is tackled by separate topics in this documentation category, as are many other fine points of working with video tracks.

Audio



Once again, knowing the codec used to encode the audio data is of critical importance. This is expressed by the *Format* and *Format profile* parameters. "MPEG Audio Layer 3" is more commonly known as MP3 and it is a universally supported audio format that can be expected to play everywhere.

As with video, audio quality parameters are the second most important data points, expressed primarily by the *Bit rate* parameter.

Summary of analysis

The content is packaged using a very popular packaging format, built upon the MPEG-4 standards suite. It is encoded using universally adopted video and audio codecs. From this it is clear that the video is meant to be easily accessible to every viewer - compatibility and availability were key for its authors.

The use of MP3 shows the age of the example content, as it is no longer considered up to par with modern competitors - instead, AAC (Advanced Audio Coding) is the breadwinner in the field of audio codecs.

The same can be said about the use of the H.264 Main profile. It is very rare that any H.264 profile besides High is used, given that almost all decoders support it, enabling everyone to take advantage of the improved efficiency enabled by High profile features.

The bitrates used are slightly higher than expected for today's environment. This may be explained by the authors' desire for high quality or simply by the limitations of the encoders that were available when the content was created.

More

Other useful tools for media file analysis are FFprobe, which is part of the FFmpeg software package, and the Bento4 tools for working with MP4 files. Both are also available in library form. They are capable of more low-level analysis than MediaInfo, in situations where you need to examine individual elements that make up media files.

Understanding components of video tracks

This example will explore how to see the layout of a video track and how to extract the individual pictures within it.

The sample content used here is Tears of Steel, by Blender Foundation. Specifically, we will use the download titled "HD 720p (~365MB, mov, 2.0)". This is a single file that ends with the extension "mov" and will play in just about any modern media player.

We will use the mp4info and mp4dump tools from the Bento4 suite for track layout and structure analysis and FFmpeg for extracting the individual pictures that make up the video track.

The sample movie uses the "QuickTime" (MOV) packaging format, which is based on the ISO Base Media File Format - an international standard that underlies all packaging formats in the MP4 file format family. This makes it highly compatible with the majority of available tools and enables easy analysis.

Let's first examine the overall structure of the file. All media files based on ISO Base Media File Format are structured as a hierarchy of boxes - a mini-filesystem of sorts. Use the mp4dump utility to extract the box structure, by executing the following command:

```
mp4dump tears_of_steel_720p.mov
```

The output will be similar to the following:

```
[ftyp] size=8+12
  major_brand = qt
  minor_version = 200
  compatible_brand = qt
[wide] size=8+0
[mdat] size=8+371579623
[moov] size=8+598972
  [mvhd] size=12+96
    timescale = 1000
    duration = 734167
    duration(ms) = 734167
  [trak] size=8+244250
    [tkhd] size=12+80, flags=f
      enabled = 1
      id = 1
      duration = 734167
      width = 1280.000000
      height = 534.000000
...
```

This represents the internal structure of the file. For example, you see here a *moov* box that has an 8-byte header and 598972 bytes of contents. This box is a container for various metadata boxes that describe the contents of the file. For more information about the meaning of the various boxes and their properties, see ISO/IEC 14496-12.

The actual media samples themselves - the compressed pictures and audio waveforms - are stored in the *mdat* box, the contents of which are opaque to the mp4dump utility.

To exclude irrelevant data and simplify the analysis workflow - this example focuses on the video track - we now remove the audio track from our sample movie. Execute the following command:

```
ffmpeg -i tears_of_steel_720p.mov -an -vcodec copy video_track.mov
```

> Note that the above step will also remove various custom extension elements from the input video, packaging the essence of the visual content into a new container file and discarding anything else. If you do this in a production scenario, ensure that you truly are free to discard all other elements in the input file!

Encoded video tracks are a sequence of pictures. With the H.264 codec used here - and all other commonly used modern codecs - the pictures can be of various different types:

- I-pictures- these are independent pictures, decodable solely using the data contained within the picture.
- P-pictures- these take another picture as a baseline and apply a transform to that image (e.g. "move these particular pixels to the right by 5 pixels").
- B-pictures- similar to P-frames, but bidirectional - they can also reference pictures from the future and define transforms such as "these particular pixels that will be fully visible in 5 frames are now 10% visible".

The exact combination of picture types may be freely chosen by the encoding workflow, creating many optimization opportunities, although certain use cases may constrain the available flexibility by e.g. requiring an I-frame to be present at exactly 2 second intervals.

Execute the following command to see the picture structure of the video track:

```
mp4info --show-layout video_track.mov
```

In addition to presenting a human-readable form of the overall file metadata, you will see a detailed printout of the video track's picture layout.

```
...
  00000959 [V] (1)  size=  7615, offset=15483377, dts=491008 (39958 ms)
  00000960 [V] (1)* size=104133, offset=15490992, dts=491520 (40000 ms)
  00000961 [V] (1)  size= 16168, offset=15595125, dts=492032 (40042 ms)
  00000962 [V] (1)  size=  4029, offset=15611293, dts=492544 (40083 ms)
  00000963 [V] (1)  size= 24615, offset=15615322, dts=493056 (40125 ms)
  00000964 [V] (1)  size=  4674, offset=15639937, dts=493568 (40167 ms)
  00000965 [V] (1)  size= 18451, offset=15644611, dts=494080 (40208 ms)
  00000966 [V] (1)  size= 95800, offset=15663062, dts=494592 (40250 ms)
  00000967 [V] (1)  size= 30271, offset=15758862, dts=495104 (40292 ms)
```

```
  00000968 [V] (1)  size= 10997, offset=15789133, dts=495616 (40333 ms)
  00000969 [V] (1)  size= 28458, offset=15800130, dts=496128 (40375 ms)
  00000970 [V] (1)  size=  9593, offset=15828588, dts=496640 (40417 ms)
  00000971 [V] (1)  size= 24548, offset=15838181, dts=497152 (40458 ms)
  00000972 [V] (1)  size=  6853, offset=15862729, dts=497664 (40500 ms)
  00000973 [V] (1)  size= 27698, offset=15869582, dts=498176 (40542 ms)
  00000974 [V] (1)  size=  7565, offset=15897280, dts=498688 (40583 ms)
  00000975 [V] (1)  size= 24682, offset=15904845, dts=499200 (40625 ms)
  00000976 [V] (1)  size=  5535, offset=15929527, dts=499712 (40667 ms)
  00000977 [V] (1)  size= 38360, offset=15935062, dts=500224 (40708 ms)
  00000978 [V] (1)* size= 82466, offset=15973422, dts=500736 (40750 ms)
  00000979 [V] (1)  size= 13388, offset=16055888, dts=501248 (40792 ms)
  00000980 [V] (1)  size=  2315, offset=16069276, dts=501760 (40833 ms)
  00000981 [V] (1)  size= 21983, offset=16071591, dts=502272 (40875 ms)
  00000982 [V] (1)  size=  3384, offset=16093574, dts=502784 (40917 ms)
  00000983 [V] (1)  size= 22225, offset=16096958, dts=503296 (40958 ms)
...
```

Each row in this printout is a picture contained in the video track. Those marked with an asterisk as `(1)*` are I-pictures. You can see how they are the largest in size, with the others enabling greater compression by referencing existing pictures and only describing the differences.

> The listing also contains the offset of the picture data in the video file and the decoding timestamp of the picture, enabling further correlation and analysis. Note that the decoding order/timing of pictures is not necessarily the same as the presentation order/timing! If B-pictures exist in the video, they can only be decoded *after* any pictures they reference, even if they are presented *before* the referenced pictures!

Having gained some insight into the structure of the video track, execute the following command to extract the 30 pictures starting at the 40 second mark as PNG files:

```
ffmpeg -i video_track.mov -ss 00:00:40 -vframes 30 picture%04d.png
```

> The extracted pictures will be fully decoded, as they would appear in a video player - it is not possible (without extremely specialized tooling) to get a visual representation of the raw data in P-frames or B-frames.

Observe how the 7th generated picture is a complete scene change in the video. You can easily correlate this with the mp4info output above - the 7th picture starting from the 40 second mark (number 00000966) is far larger in size than the ones near it. Scene changes are difficult to encode, as they refresh the entire picture and contain much new data. If the encoder is not given enough leniency to optimize for scene changes (i.e. not allowed to generate a large picture), the visual output will be low quality or "blocky" until the next I-picture. By examining the allocation of bandwidth (bytes) to various pictures, you can gain insight into such visual artifacts that may suddenly appear in a video.

Understanding DASH adaptive streaming presentations

DASH is the most widely deployed adaptive streaming technology in modern solutions, used to deliver video in a wide variety of scenarios. The best way to understand DASH presentations is to observe the network activity that takes place during playback.

This example uses Fiddler to capture and analyze browser network traffic, though any similar tool will also suffice. We will use the dash.js open source player for video playback.

For our demo content, we will use the Axinom DASH test vectors, specificially the single-period 1080p variant of the "Clear" test vector.

With your network capture running, open the dash.js nightly build sample player in any modern browser and enter the URL http://media.axprod.net/TestVectors/v6-Clear/Manifest_1080p.mpd into the textbox. Press Load to start playback. You will observe the following files being downloaded:

```
http://media.axprod.net/TestVectors/v7-Clear/Manifest_1080p.mpd
http://media.axprod.net/TestVectors/v7-Clear/2/init.mp4
http://media.axprod.net/TestVectors/v7-Clear/15/init.mp4
http://media.axprod.net/TestVectors/v7-Clear/18/init.mp4
http://media.axprod.net/TestVectors/v7-Clear/18/0001.m4s
http://media.axprod.net/TestVectors/v7-Clear/2/0001.m4s
http://media.axprod.net/TestVectors/v7-Clear/15/0001.m4s
```

```
http://media.axprod.net/TestVectors/v7-Clear/15/0002.m4s
http://media.axprod.net/TestVectors/v7-Clear/5/init.mp4
http://media.axprod.net/TestVectors/v7-Clear/5/0002.m4s
http://media.axprod.net/TestVectors/v7-Clear/18/0002.m4s
http://media.axprod.net/TestVectors/v7-Clear/5/0003.m4s
...
```

The first file is the presentation manifest - an XML document whose format is defined in ISO/IEC 23009-1. This describes the DASH presentation to a depth sufficient to allows the player to understand how to play it back.

If you look inside the manifest, you will see various `AdaptationSet` elements, each of which describes a single adaptation of the content. For example, there is one adaptation set for the video track, three adaptation sets for three audio languages and five adaptation sets for five subtitle languages.

Inside adaptation sets are `Representation` elements. For the video adaptation set, there are several of these - each representation contains the same visual content encoded using a different quality level. Each audio and text adaptation set only has one representation.

To perform playback, a player will need to decide which adaptation sets to present to the viewer. It can make this decision based on any custom or built-in business logic that it desires (e.g. language order of preference). The adaptation sets that the content author considers primary have a `Role` element in the manifest declaring the "main" role.

Furthermore, the player will need to decide which representation to present to the viewer (if an adaptation set offers multiple representations). Most players start conservatively and apply a heuristic algorithm that will attempt to present the maximum quality level that the viewer's network

connection can sustain.

The player is free to change the active set of representations and/or adaptation sets at any time, either in response to user action (selecting a different languge) or automated logic (bandwidth heuristics result in quality level change).

The `SegmentTemplate` element defines the URL structure that the player can use to access the different representations. A key factor of DASH presentations is that content is split into small segments of a few seconds each (4 seconds in case of our sample movie), which are downloaded independently. Each representation also has an initialization segment, named "init.mp4" for this sample movie, which contains representation-specific decoder configuration and must therefore be loaded before any other segment from that representation can be processed.

> The behavior described here is accurate for the DASH Live profile, which is the most commonly used variant of DASH. There also exist other profiles with slightly different behavior, not covered here. Pay attention to the "profile" attribute on the DASH manifest root element to ensure that this description applies to your videos!

As you examine the list of URLs obtained from network traffic capture and compare against the information provided by the manifest, you will conclude that the player performed the following actions after downloading the manifest:

1. Download the initialization segments for representations 2 (360p video), 15 (English audio) and 18 (English subtitles).
2. Download the first segments of the above three representations (0001.m4s).
3. Download the second segment of the audio representation.
4. From the second video segment onwards, switch to the 1080p video stream! This is indicated by downloading the initialization segment and the second segment of representation 5 (1080p video).
5. Continue to download more segments of the active representations.

By observing network activity, it becomes easy to observe the decisions that a DASH adaptive streaming player makes in operation. Such a player is simply a mechanism that downloads segments of various tracks and provides them consecutively to a media playback engine, switching tracks as appropriate.

The Axinom DASH test vectors also contain archive files that let you download the entire presentation for filesystem-level analysis. You will find that the files on disk are exactly as they are on the network level. This means that DASH presentations can be served by arbitrary HTTP servers, without the need for any custom server-side logic.

A aspect of Live profile DASH that complicates analysis is that the media samples are spread across a large number of segments. Most media analysis tools are unable to process individual segments, operating only on whole tracks. You can often get past this limitation by simply concatenating the segments of a single representation, starting with the initialization segment. For example, on Windows you may use the following command:

```
copy init.mp4 /b + 0001.mp4 /b + 0002.mp4 /b + 0003.mp4 /b track.mp4
```

This will create a track.mp4 file that contains the first three media segments from a representation. While not identical in structure to a stand-alone MP4 file, such a file can still be analyzed by most tools (such as mp4info and FFmpeg) without significant loss of functionality.

Read Understanding a media presentation online:
https://riptutorial.com/video/topic/5879/understanding-a-media-presentation

Chapter 3: Video aspect ratios

Remarks

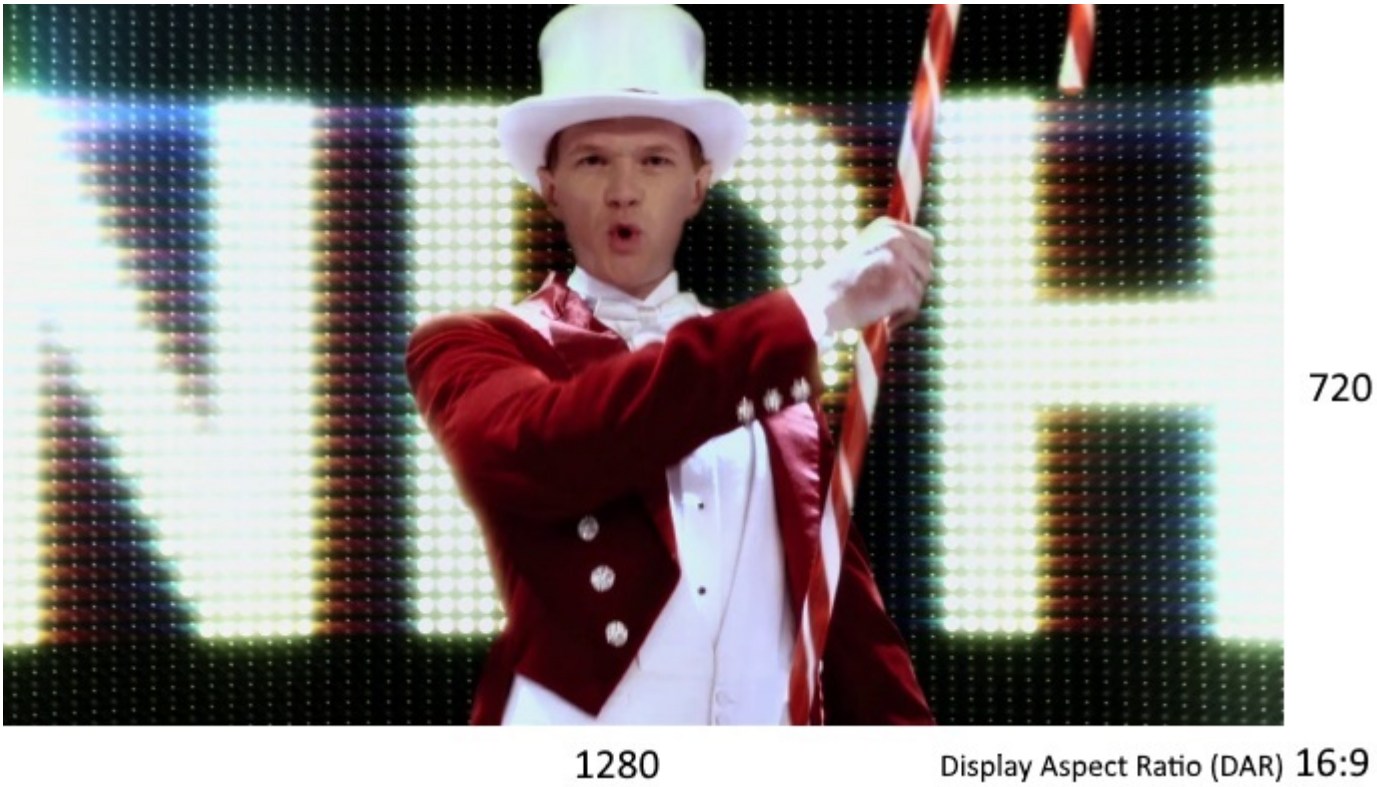
Aspect ratios are often expressed as a width:height ratio which is often - but not always - simplified and sometimes also as a simple floating point integer.

All of the following aspect ratios are the same value expressed in different ways:

- 1280:720
- 16:9
- 1.77777777777777777777777777778

Examples

Display aspect ratio (DAR)



This is a screenshot of a video playing. You see a normal 16:9 video like you would expect to see in any modern video solution. This - the aspect ratio that the viewer sees - is what is called the *display aspect ratio* or DAR.

From the illustrated parameters, we see that DAR = 1280:720 = 16:9 = 1.77777777777777777777777777778.

Picture aspect ratio (PAR)

Internally, all videos are a just series of pictures. Let's take a look at one such picture.

That looks odd, right? Indeed. The pictures that make up a video may have an aspect ratio that are different from the DAR, most often for algorithmic reasons (e.g. only sizes that are a multiple of 16 can be compressed by the chosen algorithm). This is called *picture aspect ratio* or PAR.

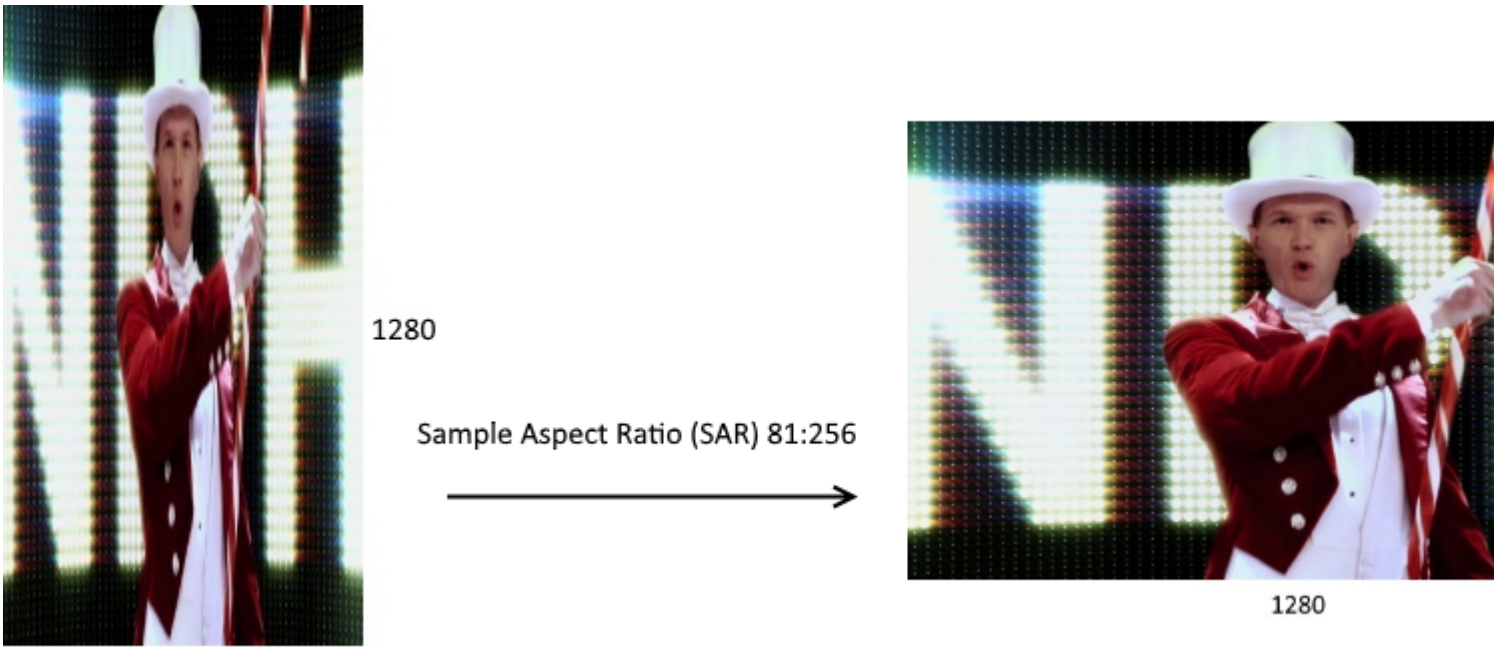
In this example, we have the picture dimensions exactly reversed between the displayed form (16:9) and the actual picture, so the PAR is 9:16. Normally, the differences are smaller but this example is exaggerated for clarity.

From the illustrated parameters, we see that PAR = 720:1280 = 9:16 = 0.5625

Sample aspect ratio (SAR)

As the *picture aspect ratio* example indicates, videos are series of pictures that do not necessarily have the same aspect ratio as the final result to be displayed to the user.

So how do you get from those stretched pictures to the normally displayed output? You need a stretching factor! This stretching factor is applied to the picture in order to bring it to the correct aspect ratio for display. This is the *sample aspect ratio* or SAR.



The stretching factor is often expressed as a ratio of two integers. You calculate it as SAR = PAR / DAR.

From the illustrated parameters, we see that SAR = 9:16 / 16:9 = (9/16)/(16/9) = 81/256 = 3.160493827160493827160493827160 5

Pixel aspect ratio

This is another name for *sample aspect ratio* and should be avoided, as the natural acronym (PAR) conflicts with *picture aspect ratio*.

Read Video aspect ratios online: https://riptutorial.com/video/topic/5713/video-aspect-ratios

Credits

S. No	Chapters	Contributors
1	Getting started with video	Community, Sander
2	Understanding a media presentation	Sander
3	Video aspect ratios	Sander