



# **Understanding and Selecting a File Format**

Peter Bubestinger-Steindl

2021-11-18

# Why bother? - Let's just have:

- Best quality
- Preserve original properties
- Lowest size
- Fast and easy to open/use
- Lasts forever
- +cherries  & ice cream  on top!
- ...

## Speaker notes

That list actually makes sense:

They define the desired properties that you have to choose from, which ones and how well you want them implemented.

# Digital Video Trinity

Container

Videocodec

Audiocodec

**METADATA** \*

## Speaker notes

The uncomfortable truth. It's more than “1 format” to choose/consider for AV.

A/V media files: “The” format of video files usually consists of 3 different formats:

- Container
- Video
- Audio      Container is the one you see in the filename as suffix: mov, mkv, mxf, avi, flv, 3gp, wav, etc.

# Format Naming

Triplet notation greatly helps reducing confusion:

- **H.264 / AAC** in **MP4**
- **FFV1 / PCM** in **MKV** (Matroska)
- **ProRes / PCM** in **MOV**
- **DPX / WAV** (PCM) in a **folder**
- etc

## Speaker notes

When communicating “which video format”, please consider using a triplet-notation, like: “video/audio in container”.

It is unfortunately most common to simply quote the file suffix as “the format”.

# What's a Codec?

*“A codec is a device or computer program which encodes or decodes a data stream or signal.”*

Source: [Wikipedia: Codec](#)



# What's a Codec?

Think of a human language...

- It's coded information.
- There may be dialects.
- Different people may  
“speak / understand” differently.

# What's a Container?

*“A container format (informally, sometimes called a wrapper) [...] is a file format that allows multiple data streams to be embedded into a single file, usually along with metadata for identifying and further detailing those streams.”*

Source: [Wikipedia: Container format \(computing\)](#)

# What's a Container?

Think of a regular paper folder...

- It's a wrapper around content.
- Contains Metadata.
- Structures the content streams.

## Speaker notes

Metadata: \* Index (which streams are contained, etc) \* Descriptive MD (title, language, etc) \* as well as technical MD (fps, aspect ratio, color handling, etc) \* NOTE: Some technical MD may be stored in the container *and* the codec/stream. This may be a blessing and a curse... Example: Aspect ratio or fps.

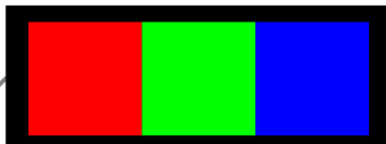
**Let's look inside! :)**

# Data Structure (Hex Editor)

# **(Documented?) Data Structure**

# PORTABLE NETWORK GRAPHICS

ANGE ALBERTINI  
<http://www.corkami.com>



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00:	89	.P	.N	.G	0D	0A	1A	0A	00	00	00	0D	.I	.H	.D	.R
10:	00	00	00	03	00	00	00	01	08	02	00	00	00	94	82	83
20:	E3	00	00	00	15	.I	.D	.A	.T	08	1D	01	0A	00	F5	FF
30:	00	FF	00	00	00	FF	00	00	00	FF	0E	FB	02	FE	E9	32
40:	61	E5	00	00	00	00	.I	.E	.N	.D	AE	42	60	82		

## SIGNATURE

FIELDS

VALUES

signature

\x89 PNG  
\r\n \x1a \n

## HEADER

size

0x0000000D

id

IHDR

width

0x00000003

height

0x00000001

bpp

0x08

color

0x02 RGB

compression

0x00 DEFLATE

filter

0x00

interlace

0x00

CRC32

0x948283E3

## DATA

size

0x00000015

id

IDAT

ZLIB

window size

0b00001000

method

0b00001000 DEFLATE

level / dict.

0b00011101

checksum

0x081D % 31 = 0

DEFLATE

last block

0b00000001 FINAL

block type

0b00000001 RAW

data length

0x000A

!length

0xFFFF5

PIXELS

line filter

0x00 NONE

FF 00 00 00 FF 00 00 00 FF

adler32

0x0EFB02FE

CRC32

0xE93261E5

## END

size

0x00000000

id

IEND

CRC32

0xAE426082



# VLC / MediaInfo

Website: [videolan.org/vlc](http://videolan.org/vlc)

Website: [mediaarea.net/MediaInfo](http://mediaarea.net/MediaInfo)

## Speaker notes

“MediaInfo is a convenient unified display of the most relevant technical and tag data for video and audio files.”

btw: A VLC-related WARNING: There’s a major fraud out there: “[www.vlc.de](http://www.vlc.de)” - aka “VLC Plus Player”. It contains the original VLC, but with unknown - possibly malicious - additions/modifications. Stay away from it.

# Characteristics / Properties

	File 1	File 2	File 3
Container	MOV	MOV	MOV
Videocodec	UYVY	H.264	XviD
Resolution	720 x 576px	1920 x 1080	640 x 480
FPS	25	24	30000/1001
-			
Audiocodec	PCM	AAC	MP3
Samplerate	48 kHz	48 kHz	44.1 kHz

	File 1	File 2	File 3
Channels	Stereo	Surround 5.1	Mono

## Speaker notes

This is just a random example to show that from the “outside” (=file explorer) you would only see that all files are “.mov” - whereas their actual audio/video codecs, as well as their technical properties are completely different. It therefore always makes sense to use proper tools to “look inside” (MediaInfo, etc).

# Format choice = A balance of ...

## Sustainability:

1. Disclosure?
2. Open reference  
implementation/libs?
3. Adoption/popularity?
4. Complexity?
5. Independence vs external  
contingencies?
6. Artificial restrictions?
7. Self descriptive?

## Quality and functionality:

1. Preserve “original”?
2. Image/sound quality?
3. Interoperability?
4. Editing?
5. Support for (additional/expected)  
properties?
6. Performance & data size?

## Speaker notes

One reason why there is no “one-size-fits-all” file format for AV, is that there are different use cases, each having a different focus.

These lists are a good reference for giving you questions to ask about formats that may be considered as an option.

The next slide translates these buzzwords into more concrete questions/situations.

# ★ Translates to...

## Sustainability:

1. Documentation openly accessible?
2. Open reference implementation?
3. How likely is it to be supported in tools/devices for which userbase?
4. Which features are implemented/tested/stable?
5. Which choice/requirements do I have to handle it beyond “shelf life”?
6. Is it legal/possible to handle it in future/different situations?
7. Can it contain proper metadata?

## Quality and functionality:

1. Preserve significant properties?
2. Sufficient image/sound quality and robustness to multi-generation copies?
3. Interoperability / ease of usage & access?
4. Direct use for editing?
5. How many different formats will I need (pile up)?
6. Handle performance / data size requirements?



1. Documentation openly accessible? Without documentation (or access to it), a file format is the equivalent of a secret code/language. You can imagine how well “secret code” satisfies preservation requirements?
2. Open reference implementation? Any code that implements a file format can be called an “implementation” of that format. A “reference implementation” is an application that is able to read/write that format in a proper way, used to check the validity of other implementations that read/write the same format. With proprietary (=closed source) formats it is not uncommon that there is no real “official” reference implementation - but merely a black-box binary provided by the vendor, not necessarily freely accessible and possibly subject to change at their will. Without an open reference implementation (=including source code), interoperability issues are more likely to happen.
3. How likely is it to be supported in tools/devices & for which userbase?

Adoption of a format plays a major role in how easy (=cheaper) vs hard (=more expensive) it is to use a format. It may be necessary/useful to engage with vendors of developers and ask for supporting a certain format in tools you would like to use.

The userbase is important, as it has a great impact on where a format is supported, and for which use-cases it is real-world tested. If a format is common in a userbase that covers your use cases, it's good. If a format was designed/intended for a userbase further away from your use cases, you may find that you get less support (or understanding) for your needs.

#### 4. Which features are implemented/tested/stable?

Developing, testing and supporting a file format is a tricky task. Therefore it is common that the “most popular/important” features are the ones most likely to be tested and working reliably. Other functions that may also be defined (even in a standardized format), may not receive the same attention by its supporters, often leading to (interoperability-)issues with not-so-popular functions. This is especially important for us, since certain features required for preservation are often less important for the main intended use-case of a format.

A popular example is “lossless”: both, JPEG2000 as well as H.264 can produce lossy as well as lossless encodings. However in practice, the lossless feature is often neither tested (and sometimes not even supported) in many of their implementations.

5. Which choice/requirements do I have to handle it beyond “shelf life”? For digital formats, this can be reduced to: If you have an Open Source version that is able to encode/decode a format, you can most likely continue using it under future (=unknown), possibly changing conditions.
6. Is it legal/possible to handle it in future/different situations? Patents, proprietary licensing and copy protection mechanisms are merely artificial restrictions that must be considered when choosing a file format for preservation. Especially since access to the material should not be hindered by them.
7. Can it contain proper metadata? It is good practice to include at least rudimentary metadata that allows to “identify” a file if found in the wild. Typical fields are: title, unique-identifier, institution name, etc.

Depending on your collection and users, different metadata may be useful or required to create a proper AIP.

#### **Quality and functionality:**

1. Preserve significant properties? Can the format maintain certain, relevant properties of the source as close to the original as possible?
2. Sufficient image/sound quality and robustness to multi-generation copies? Can the format provide sufficient quality, and how much impact would repeated encoding (lossy) have on it?
3. Interoperability / ease of usage & access? Can the format easily be used with different tools, under different operating systems, environments, devices, etc?
4. Direct use for editing? Can the format be used directly for non-linear editing or extracting parts, or does it require pre-transcoding in order to do so?
5. How many different formats will I need (pile up)? Can the format handle and depict the majority of my collection, or would I need different formats for other source material? It is intended to have a small number of different file formats in an archive, as maintaining workflows/tools becomes more difficult (=expensive) with more formats. However, it is also not advisable to simply choose a “can do anything and everything” format, as mentioned above: More features = harder to implement, support, maintain, etc.
6. Handle performance / data size requirements? Can the required (daily) workflows be supported in due time, or are there bottlenecks, budget constraints, etc?

**Your format(s)...?**

## Speaker notes

At this point, please provide suggestions for any AV data format that we shall test to check against the above mentioned list(s).

In this case, merely answer “yes/no” if you think you (or someone else in your institution) may know an answer to the question or has checked it thoroughly.

# Your use cases/focus?

- Who will want/need to work with these files?
- Under which conditions?
- For how long?
- Digitization vs Production vs Preservation vs Access?
- Which properties are significant *to you*?

## Speaker notes

It's good to define (and write down) what you actually need from a file format in which use-case.

**“Different strokes for different folks”** 😏



## Speaker notes

Sometimes one format can be used for all these cases (merely lower quality/bitrate for access or mezzanine), but with AV it's not uncommon that these are different codecs (or even containers).

It is perfectly normal to have a preservation copy that requires transcoding before going into other workflows.

The formats for access and editing are more likely to be changed more often than the preservation format.

# Significant properties

*Knowing and deciding which properties to safeguard and which are allowed to change.*

See:

LoC FADGI: DRAFT Significant Properties for Digital Video  
Nestor (DE): Leitfaden DLTP AV Medien

## Speaker notes

For some it's the resolution, color information, audio quality - for others it's sufficient to see/understand it “good enough”, or to be able to quickly edit-and-broadcast as the main focus.

Depends.

However, be aware that your recording may possibly be used in a different context in the future, so if possible don't aim “too low”.

But please make an active decision and possibly document it (which ones and why) somewhere.

# Significant properties

Depend on media type (and use case).

## Video

- resolution
- framerate
- aspect ratio
- colorspace
- subsampling
- ...

## Audio

- “resolution”  
(= samplerate, bit-depth)
- channels
- channel layout

## Speaker notes

Definition fuzziness in the preservation community:

Some say “Significant” are the properties that must be maintained as-is and kind of “must never change”, whereas others define it as “should be aware of and decided how to deal with them”.

Further properties: \* scan type (interlaced / progressive) \* field order \* color information \* ...

# Quality

- Avoid generation loss. (if possible)
- Avoid resizing. (=rescaling)
- Don't invent more bits. (e.g. DV as v210)
- Preserve colorspace / bit-depth, etc.
- More headroom for lower quality. (e.g. 24bit/96kHz for Shellack)
- Select high enough bitrate. (lossy)  
(Or proper “[Constant Rate Factor \(CRF\)](#)”)

## Speaker notes

These are some guidelines for maintaining the highest (reasonable) quality for preservation copies.

# Bitrate = Data per Time

- $\text{Mbps} / 8 = \text{MB} / \text{second}$
- $\text{MB/s} * 60 = \text{MB} / \text{minute}$
- $\text{MB/min} * 60 = \text{MB} / \text{hour}$

*btw: Constant (CBR) vs Variable (VBR)  
bitrate?*



## Speaker notes

If you know the bitrate, you can calculate the size of your files, by multiplying the bitrate by the runtime duration (time).

A fixed bitrate however is only available for:

- constant bitrate (lossy) compression.
- uncompressed.

For lossy compression, bitrate is an encoding parameter, but for uncompressed we'll show later in this session how to calculate the size.

For lossless or variable bitrate, the exact size cannot be pre-calculated - only estimated. The actual size can still vary, since the size of these kind of encodings greatly depends on the content being encoded. Rule of thumb: Less motion = smaller size and vice versa. Noise make files larger.

There's also the choice with some formats between constant bitrate and variable bitrate. Constant bitrate is simpler, more robust, straight to calculate/estimate. Simpler=preserves better. Variable bitrate is another way to compress space, but makes decoding/interpreting the data more complicated and may lead to different behavior in different tools/setups.

# Size

- Bitrate = Size vs Quality  
(bitrate as parameter exists *only* for lossy encoding)
- Uncompressed > lossless > lossy

## Speaker notes

- Bitrate as tuning parameter only applies to lossy codecs, because lossless/uncompressed are not allowed to throw anything away - therefore their bitrate cannot be adjusted by definition.
  - Only reason for compression: data size.
  - Any form of compression adds complexity to a format. Complexity in terms of understanding (=implementing) a codec. Complexity in terms of performance requirements. Uncompressed is always the least-CPU-requirement.
  - Why “compressed” is often equated as “lossy”. Because moving images produce a lot of data and lossy compression produces the best compromise between quality/size/performance - lossy is the de-facto default when producing videos.

# Performance

Often a tradeoff between:

**Processing power** <sup>(CPU/RAM)</sup>  
(format/algorithm  
complexity)

**I/O bandwidth**  
(disk/network)  
(data size)

## Speaker notes

Better compressed, usually meaning more complex codec - resulting in smaller files. These files travel lighter, but require more processing power (CPU/RAM) to encode/decode. If these requirements reach the limits of currently available processing power, realtime issues may occur. This also means: can the daily workflows be processed in due time?

Oh, btw: Hardware acceleration plays an important role when it comes to codec performance. However, hardware acceleration is not necessarily available or implemented for all formats. Usually you will find this for “popular” formats or ones that are used in professional domains.

# Format Examples

## Video

## Audio

## in Container

### Preservation:

- V210
- FFV1, J2K
- High-bitrate lossy
- ...

- PCM
- FLAC

- MOV
- MKV
- MXF

### Mezzanine:

- ProRes
- H.264
- DVCPRO50
- ...

### Access:

- H.264
- VP9
- DVD
- BluRay
- ...

- MP3
- AAC
- Opus
- ...

- MP4 (M4V)
- MKV (WebM)
- MPG

## Speaker notes

I've intentionally used fuzzy wording for access formats, since it better depicts the perception and wording used in daily life.

Later in this session, we'll get to how to select properties for these use cases, which helps to decide which format to use for which case.

# **Lossy, Lossless, Uncompressed?**

How it affects quality and preservation.



# Lossy



Zlad! Elektronik Supersonik

## Speaker notes

“Lossy compression” literally means: some parts of the content are “lost” (=thrown away) in order to gain more compression (=smaller files). This process is irreversible and accumulates (=generation loss).

Of course the above image is exaggerated, but it shows pretty well what lossy compression is and what artefacts a typical MPEG-like compression algorithm produces.

(btw: This is a snapshot image of the highest-quality version of the video on the original website (around 2009)).

Due to the vast data sizes of AV, lossy encoding is the current de-facto default in most environments that handle digital AV (except film). High quality lossy codecs may sometimes be called “visually lossless” (e.g. ProRes). Which only means that they’re pretty good at hiding their artefacts from your eyes/ears. High-bitrate lossy formats are standard in broadcast and production.

For audio this is different, merely because the data size is so “trivial” for modern computer systems that the studio standard is uncompressed (PCM) in WAV. Suggesting lossy encoding to audio engineers as production source format would probably get you kicked out of their studio ;)

# Generation Loss

# Generation Loss Comparison

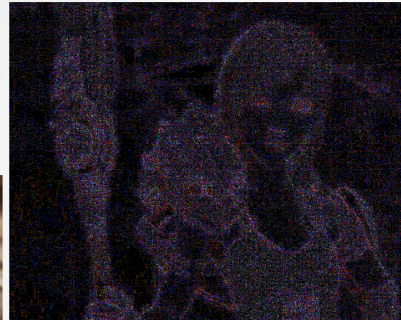
(color/contrast increased for better visibility)

Original

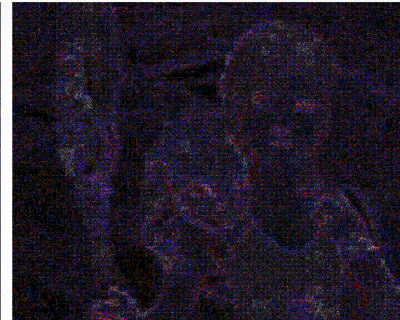


RAW (YUV422)

IMX50 D10 (YUV422)



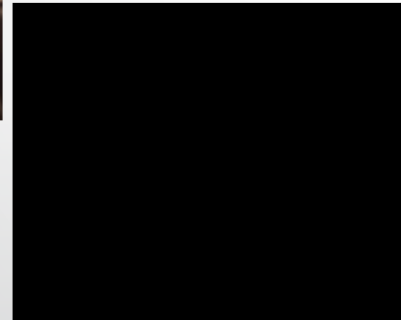
DVCPro50 (YUV422)



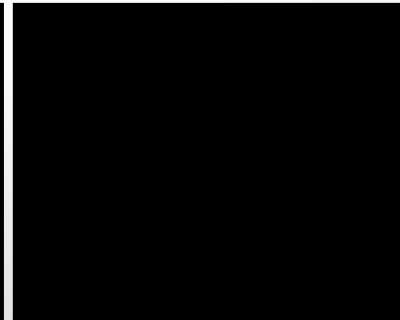
IMX50 D10 (YUV422)



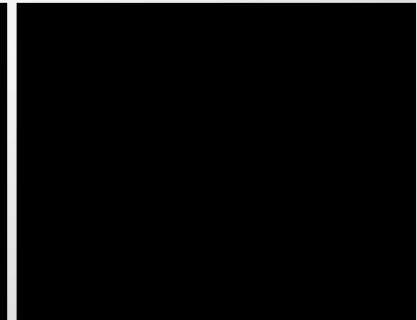
Delta Original / Gen1



Delta Gen1 / Gen2



Delta Gen2 / Gen3



FFv1 (YUV422)

FFv1 (YUV422)

FFv1 (YUV422)

## Speaker notes

Here's an example where one can see the difference between each encoded version in popular broadcast codecs.

Since it's almost certain that there's not "evergreen format", any format will sooner or later have to be migrated to another one. The longer an item is preserved, the more often it will encour such a format change (transcoding). Therefore, generation loss should be considered.

Also: Usage of archive material: What happens if you hand out your material to an editor, say on DVD or a mezzanine format? The editor then edits and exports to another format, the broadcaster or cinema then to another? ...and in the end this work is then also converted to an access format.

And theeeen, years later that access copy is used as a source for a documentary or online edit.

How many generation losses accumulate here? How could they be reduced?

Generally: please try to avoid unnecessary generation losses. Why not? :)

# Lossless

*“It’s like ZIP for film!”*

- No generation loss
- Way larger than lossy
- Smaller than uncompressed





# Uncompressed

- No generation loss
- Dead simple (=preserves well)
- The largest possible version
- Uncompressed != Uncompressed?

There's *more than just 1* “uncompressed”

(Ex: RGB, BGR, UYVY, YUY2, V210, etc)

## **Uncompressed != Uncompressed:**

There are many variations, and each one is a separate (not necessarily compatible) format. However, they're all dead-simple, easy to implement and cross-convert without any loss. (Except if changing color spaces/models - but that's another story...)

Examples: \* YVYU / YUYV: stored information is identical except the U and V samples are swapped. \* RGB / BGR: stored information is identical, except their “order of appearance” in the bitstream.

# Uncompressed - Think of it as:

**4px RGB Image:** (8bpc = 24bit/pixel)

RGB RGB RGB RGB (4\*3 = 12 byte)

**4px YUV Image:** (8bpc, 4:2:2, 16bit/pixel)

UYVY UYVY (2\*4 = 8 byte)

**2 samples audio:** (2ch, 16bits)

LL RR LL RR (4\*2 = 8 byte)

## Speaker notes

Each uppercase letter represents 1 byte (=8 bits).

Image: RGB: 8 bits-per-component = 24 bytes per pixel

- Red, Green, Blue, Alpha

YUV:

- Subsampling 4:2:2 uses only half the bits for U and V. Therefore, the proper notation would be: U0Y0V0Y1  
U2Y2V2Y3 See: <https://www.fourcc.org/pixel-format/yuv-uyvy/>

Audio: 16 bits per sample, 2 channels (left/right).

- Left
- Right

All you need to know to manually reverse engineer (decode) an uncompressed image bitstream would be (somewhat):

- Color model
- Image size (width=line length, height=number of lines per field/frame)
- Sample size/bit depth

# Paper analogy

Table of Contents

Metadata.....2

Video Track 1.....3

    Format: Uncompressed

Video Track 2.....4

    Format: Lossy compression

Video Track 3.....5

    Format: Proprietary

Audio Track 1.....6

    Format: Uncompressed

    Language: English (en)

Audio Track 2.....7

    Format: Uncompressed

    Language: German (de)

Subtitle Track 1.....8

    Format: SubRip (srt)

    Language: Netherlands (nl)

Metadata

Title: Video Mockup

Author: Peter B.

Date: 03.01.2017

Duration: 13m37s

License: CC-BY-SA

Video Track 3

Format: Proprietary

426B426B426B426BE280A6426B47726E

426B426C426BE280A6E280A647726E47

726E47726E426B426B5959426B595942

6B426B47726E47726E47726E426B426B

5959426B5959426B426B47726E47726E

47726E47726E426B426BE280A6E280A6

426B426B426B426B426C426B426B426B

426C426B426B426B426B426B426B426C

426B426BE280A60A

Audio Track 1

Format: Uncompressed

Language: English (en)

Channels	
1. Left	2. Right
00000000000000	00000000000000
00000000000000	00000000000000
00000000000775	00000000000921
5746717633042	9889453841869
7260752435502	9229751162070
5534694932924	8268766924819
0142834161271	8542239617845
2493252116899	5678678789261
3343806495658	2571072497935
5869056...	4839133...

Video Track 1

Format: Uncompressed

Black Black Black Black ...

Black Green Black Blue1 Black ...

...

Green Green Green Black Black

Yellow Yellow Black Yellow Yellow

Black Black Green Green Green

Black Black Yellow Yellow Black

Yellow Yellow Black Black Green

Green Green Green Black Black ...

...

Black Black Black Black Blue2 Black

Black Black Blue2 Black Black Black

Black Black Black Blue2 Black Black

...

Video Track 2

Format: Lossy compression

Bk Bk Bk Bk ...

Bk Grn Bk Bl Bk ...

...

Grn Grn Grn Bk Bk Y Y Bk Y Y Bk Bk

Grn Grn Grn Bk Bk Y Y Bk Y Y Bk Bk

Grn Grn Grn Grn Bk Bk ...

...

Bk Bk Bk Bk Bl Bk

Bk Bk Bl Bk Bk Bk

Bk Bk Bk Bl Bk Bk

...

Audio Track 2

Format: Uncompressed

Language: German (de)

Channels	
1. Left	2. Right
00000000000000	00000000000000
00000000000000	00000000000000
00000000000036	00000000000633
5847401428341	0427260752435
6127124932521	5025534694932
1689933438064	1316257598310
4186992297511	0546213491131
6207082687669	5371923103658
2481904480771	4740142834161
5489261...	2787458...

Subtitle Track 1

Format: SubRip (srt)

Language: Netherlands (nl)

1

00:01:47,585 --> 00:01:49,585

Ik weet het zeker. Ik ben zwanger.

2

00:01:51,014 --> 00:01:54,529

Wat voor soort vader zou ik zijn?

Ik bedoel, ik dood mensen.

3

00:01:56,615 --> 00:01:59,701

Ik?

4

00:02:01,076 --> 00:02:02,281

Dit kind komt eraan.

5

00:02:02,381 --> 00:02:05,469

Maar jouw rol in zijn leven

is helemaal jouw keus.

...

# Risks to format longevity

- Data errors
- Obsolescence
- Vendor lock-in
- Format complexity
- Interoperability issues

Countermeasures?

- Data errors: Files are corrupt. These errors may include filenames or filesystem tech-MD.
  - Obsolescence: Not supported anymore by accessible tools. This is neither God-given, nor irreversible, unless it's a black-box format. Format obsolescence/support is a human-made decision. Documentation/schematics are an important game changer.
  - Vendor lock-in: For long-term preservation, vendor- and technology-neutrality is a must: They will come and go, and with lock-in situations Eternal Migration is hindered or even impossible. Format normalization helps here.
  - Interoperability issues: If a format is read and written differently by different applications, it might “morph”. This morphed version might work fine with the tools used in a certain environment, but might be completely broken if read/written by another tool that misunderstands the “dialect”.



# Data errors: Error resilience?

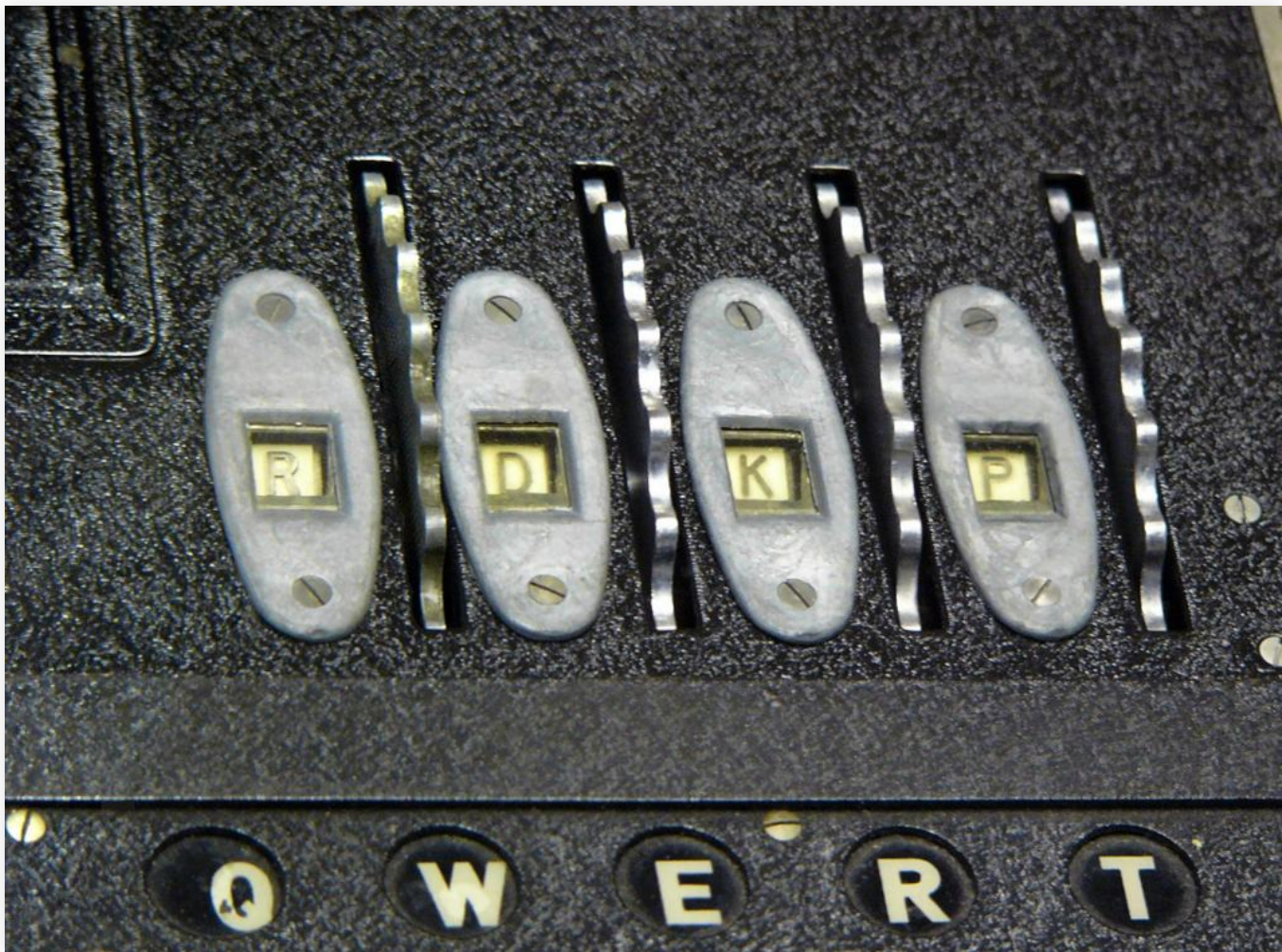
- **Bitstream checksums:**  
Ability to *know* if the content is intact.
- **Error concealment:** Optional choice of decoder to “mask” decoding issues.  
(decoder specific)
- **Make backup copies!** 😊

## Speaker notes

Error resilience of a format is nice, but don't solely rely only on it: Make backups! :)

# **Obsolescence / Vendor Lock-in**

**Open vs Closed**

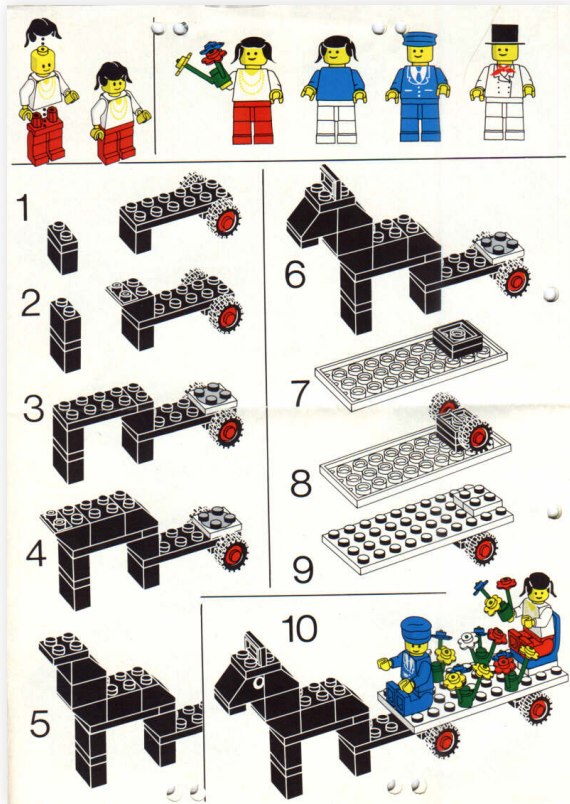


Enigma encryption rotor windows

## Speaker notes

What will be easier (=more likely) to be understood/accessible now and in the future: Documented language or secret code?

# Theory vs Practice



“Implementation overrules paper specs. Always.”

## Speaker notes

If the implementation is open: you can use, study, share and improve if necessary.

If it's closed/proprietary then there's nothing you can do about it: Black box. Vendor lock-in. With digital, this means that it's quite likely that this as-is binary may not be able to keep running/functioning when its environment changes.

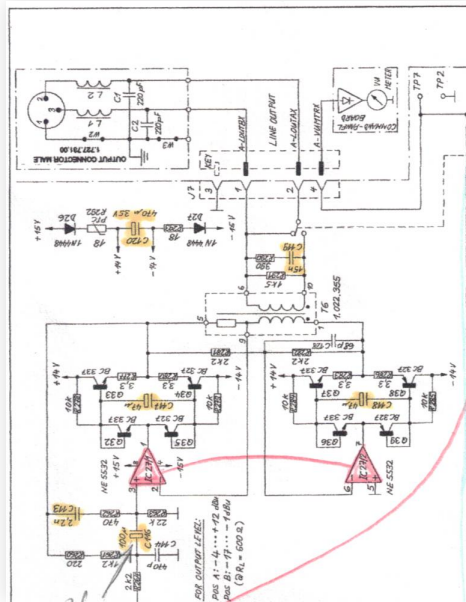
You've probably seen how fast even good and stable programs "age" until they stop working, once their native Windows version is "too long ago..."?

Even with official standards, their implementation is what creates the actual encoding/decoding. Regardless of what's written in any paper manual (=theory): The code implementation (=practice) is what counts.

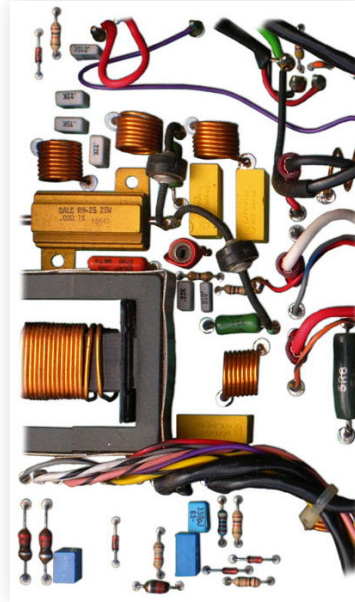
Therefore, it makes sense to demand an openly accessible reference implementation, with its source code under a license that allows to make use of it as you see fit.



# The Eternal Replayer



+



=





## Speaker notes

As an example:

- Schematics
- Building components
- +the right to use, study, share and improve them.

= “The Immortal Replayer”

Because it can be kept alive, or rebuilt or adapted to future needs or with future technology (whatever there may be).

Since software *is* the schematics and building components at the same time, having the source code and the right (=license) to use, adapt and share them - gives you an immortal file format. By definition.

This has been proven in the real world in many different digital domains outside AV or preservation many times already, with different tools and data formats from ancient computer systems. Even popular ones now dead (Atari, Amiga, C64, Amstrad Schneider, etc etc.)

# Format Complexity



## Speaker notes

Be careful with “one size fits all”: Sporks are good for camping, but there’s a reason why we still have separate tools for the job: spoon, knife and fork.

Considerations:

- More features = more complex / chance that only parts of specifications are supported by tool X.
- Can be non-trivial to judge what is “simple” and what is “complex”
- Find the sweet spot for your use case(s).

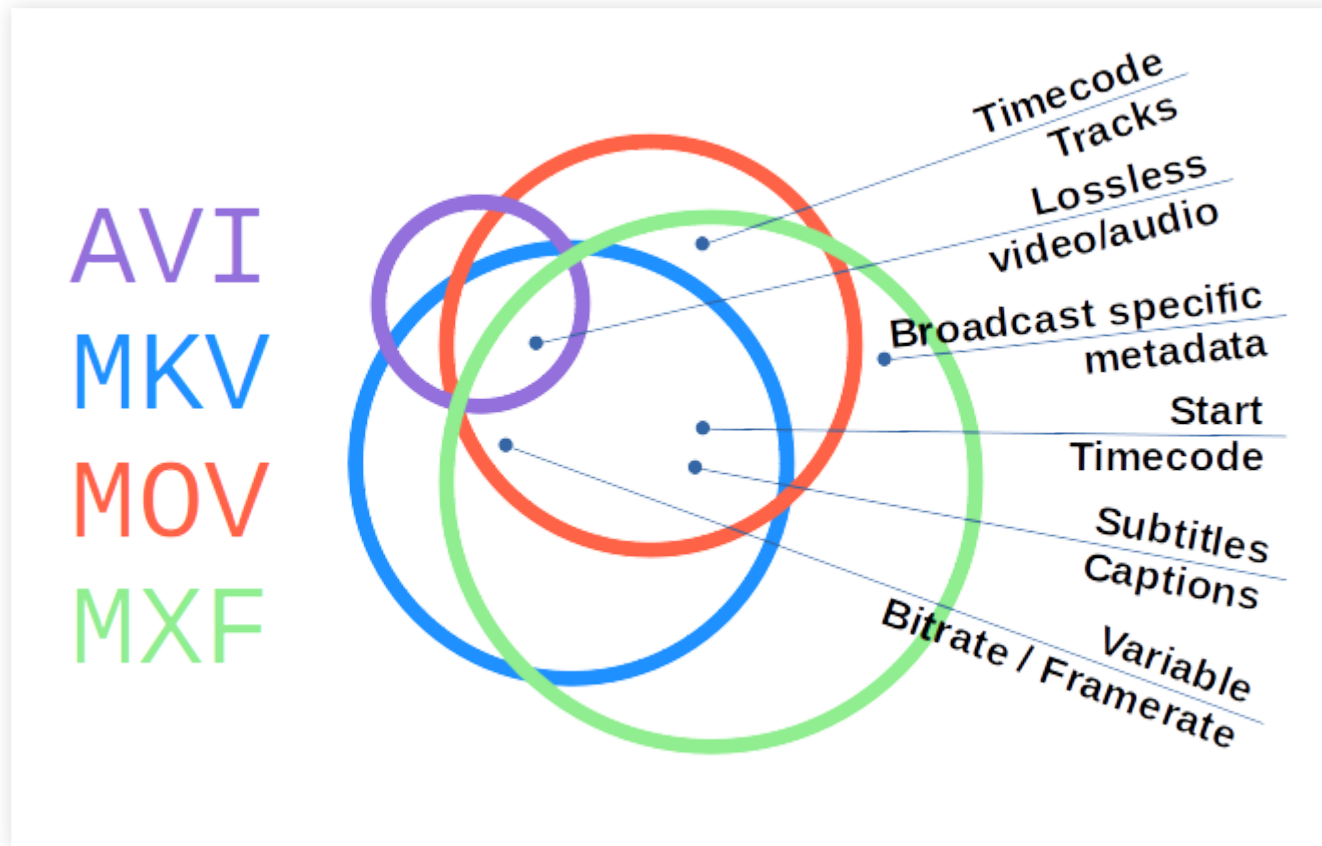
# Format Complexity: Less is More

Good rule = “Minimalistic Data Format”:

- As simple as possible
- As complicated as necessary

*Simpler = more stable, easier to use, keep alive, reconstruct or fix.*

# Yagni kiss Moscow?



YAGNI / KISS / MoSCoW

## Speaker notes

It may look cryptic, but is actually quite simple and useful:

It is an example for finding the “minimalistic data format” that suits your needs.

- **YAGNI = “You Ain’t Gonna Need It”:**

It’s supposed to prevent you from selecting a format that is possibly too bloated or unnecessarily complex, or less-well supported.

- **KISS = “Keep It Simple, Stupid”:** “The KISS principle states that most systems work best if they are kept simple rather than made complicated; therefore, simplicity should be a key goal in design, and unnecessary complexity should be avoided.”

- **MoSCoW = “Must, Should, Could, Won’t”:** “The Moscow method is a prioritization technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement”

The example shows 4 different container formats for AV. Each one has different number and kinds of supported features, represented by the size of its circle.

- Larger circle = more features.
- Smaller circle = less features.
- Overlapping circles = common features.

Now, according to the MoSCoW method, write down which features you:

- must have
- should have
- could have
- won't have (this time)

And check which format provides them, then draw a dot in the corresponding circle area.

**For example:** Only MXF may be able to provide support for broadcast-specific metadata/functions, therefore that feature will only have a dot in the MXF circle. Whereas, all (except) AVI can store aspect ratio - so that dot would go into overlap of all - except AVI. The feature of “extremely simple, well-documented and stable/unmodified for ages” would likely to be a dot in either AVI or MKV.

Use this to find out which format fits your needs, while being “as simple as possible and only as complicated as necessary”.



# Summary: Preservation Format

- Can be used to generate all other versions.
- Depicts the “original” source as accurately as possible.
- No artificial restrictions for using it.  
Now and under unknown future (=unknown) conditions.
- Well documented, no secrets, FOSS implementation exists.
- Bit error resilience would be nice.
- Consider GOP=1 (=Intraframe only).
- Audio format: Normalize to uncompressed PCM/WAV.
- For video container formats, consider using MKV or MOV.  
MXF only if really necessary because:
- **As simple as possible, as complicated as necessary.**

*summarizes as: “**preserves well**”*

# Best practices for ingest/digitization

- Capture analog video without adding generation loss.  
Uncompressed (v210) or lossless (FFV1, J2K).
- Or fallback option: high-quality lossy. At the highest quality (bitrate) you can store and manage well over time.
- Capture digital tape as “natively” as possible. (MiniDV, DAT, DigiBeta, etc.)
- Store already-digital files “as original” as possible.  
Transcode only if codec does not satisfy “sustainability” checklist. Rewrap/rewrite container. Always. Even if identical.

# Tempting...

- Hey, it's a standard!
- Hey, everyone's using it!
- Hey, the “big ones” are using it!
- Hey, it's from a major company!
- Hey, it can do *everything*!
- Hey, it's so easy to use!
- Hey, it's gratis!

## Speaker notes

Of course all of these are valid reasons to adopt a file format, but when it comes to long-term preservation requirements and independent sustainability, they may not be sufficient, and sometimes even misleading.

# Rather...

- ask, ask, ask.
- get documentation.
- get sample files.
- try handling/opening them *outside their “usual” bubble.*
- with at least 1 open implementation.

- Ask users with similar/identical use cases than yours how they're doing with that format.
  - Get file format data layout specifications and other related documentation that you can get your hands on. We've already profited from the ones who did this for past "data formats": analogue ones or even ancient writings on clay stone tablets, etc.
  - Get sample files and try using them as-if in your desired workflow. Ask manufacturers or service-providers to provide you samples, or show you how your samples behave in their application/environment.
  - Try sample files "outside their usual bubble": This means that sometimes certain format behave well within expected environments (e.g. ProRes on Apple stuff, certain MXF-flavors in broadcast tools, etc) - but when deviating from the tested grounds, things may look quite differently.
  - If you have at least 1 open implementation (=binary plus source code under a copyleft license), you have everything you need to get out of that format under any future conditions. Without artificial restrictions.

# **Comments?**

# **Questions?**

`p.bubestinger@av-rd.com`

# Links

- [Primer on Codecs for Moving Image and Sound Archives](#)
- [Hex Editing for Archivists](#)
- [Comparing Video Codecs and Containers for Archives](#)
- [Digital Media Primer for Geeks](#)
- [A short guide to choosing a digital format for video archiving masters](#)
- [Media Digitization and Preservation Initiative \(MDPI\)](#)
- [Understanding audio bitrate](#)