[Live document: last update 20200624]

"On date, time, location and other metadata in MP4/MOV files"

Leo van Stee

1 Introduction

During writing software to change dates in MP4/MOV files I had a lot of moments that I thought: how is this data stored? There are a lot of special cases that have probably grown during the development and merging with other standards. This document (for now) focuses on the atoms/boxes where metadata is stored. The idea behind this document to make things more clear and explain some of the knotty terminology with things often meaning the same (or not).

About this *webpage design*: there is none. I am currently updating this information in an HTML file that I edit MS Word, with the main aim to also make it printable on A4/letter format. There are Excel tables linked in this document that I want to 'keep on the page'. I realise that it looks sub-optimal in a browser… Please resize the browser to a small width, or you can view the pdf version index.pdf.

MP4 or Quicktime? What name to use: The MP4 format is based on the Quicktime format[1] but from here I will call it MP4 format since this has become more general.

2 Dates and data in MP4 files

This whole exercise started with the annoyance of having MP4 files wrongly dated because of wrong time-setting on the recording device and also by noticing strange sorting behavior during sorting in Google Photos and other applications. Simply, the first distinction is between the operating system file attributes (time created, modified, last opened) and the data stored within the MP4 file.

3 Atoms that contain metadata

3.1 Metadata atom

The 'meta' atom, full name 'metadata atom' is one container where metadata is stored in an MP4/MOV file. It can be either a so-called 'full atom' with version and flag bytes added, or a non-full atom without the latter two. The size of the atom is variable and depends on all the data inside of it (child atoms).

Metadata atom			
Variable	# Bytes	Type	Values
Size	4	uint32	*variable*
Type	4	uint32	'meta'
Version	1		1
Flags	3	3 bytes	

Position in file. The path of metadata atom can be either \moov\meta or \moov\udta\meta, or both. No more than one atom is allowed in each path. The metadata atom can also be present in :[trak ?].

The meta atom contains many childatoms and data stored in keys. I will not describe the structure of each separate atom inside as this has been often described elsewhere, e.g. (1) and references therein. The idea of this document is to show the structure of the atoms using data from real-life examples in illustrations such as in Table 1. In the top-line the hexadecimal byte values are shown and below descriptive information.

Table 1: Example of metadata atom that has no 'keys' section. (file had no metadata before, metadata inserted by Windows property editor). Note that metadata atom is a **full-atom here**. File:[2]

Metadata atom	00 00 00 C0 size	6D 65 74 61 type meta	00 v	00 00 00 flags						
Metadata handler atom	00 00 00 21 size	68 64 6C 72 type hdlr	00 v	00 00 00 flags	*00 00 00 00* *predefined=0*	6D 64 69 72 handler type mdir	*00 00 00 00* *Reserved=0*	*00 00 00 00* *Reserved=0*	*00 00 00 00* *Reserved=0*	00 Name
Metadata Item List Atom	00 00 00 93 size	69 6C 73 74 type ilst								
Metadata Item Atom	00 00 00 2E size	A9 77 72 74 type ©wrt								
Value atom	00 00 00 26 size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	00 00 00 00 locale indicator country language	57 6E 20 63 6F 6D 70 6F 73 65 72 … type=1: string Wn composer1/c..					
Metadata Item Atom	00 00 00 19 size	74 6D 70 6F type tmpo								
Value atom	00 00 00 11 size	64 61 74 61 type data	00 00 00 15 type indicator DF wknown types	00 00 00 00 locale indicator country language	00 type=$15=21: big-endian signed int in 1,2,3 or 4 bytes					
Metadata Item Atom	00 00 00 20 size	A9 6E 61 6D type ©nam								
Value atom	00 00 00 18 size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	00 00 00 00 locale indicator country language	57 69 6E 54 69 74 6C 65 type=1: string "WinTitle"					
Metadata Item Atom	00 00 00 24 size	A9 63 6D 74 type ©cmt								
Value atom	00 00 00 1C size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	00 00 00 00 locale indicator country language	57 69 6E 20 43 6F 6D 6D 65 6E 74 73 type=1: string "Win Comments"					

Table 2: Example of metadata atom that has a 'keys' section. Note that metadata atom is a **non-full**-atom here. File: [3]

Metadata atom	00 00 xx xx size	6D 65 74 61 type meta								
Metadata handler atom	00 00 00 22 size	68 64 6C 72 type hdlr	00 v	00 00 00 flags	*00 00 00 00* *predefined=0*	6D 64 74 61 handler type mdta	*00 00 00 00* *Reserved=0*	*00 00 00 00* *Reserved=0*	*00 00 00 00* *Reserved=0*	00 00 Name
Metadata Item Keys Atom	00 00 00 93 size	6B 65 79 73 type keys	00 v	00 00 00 flags	00 00 00 03 entry_count					
Key value 1	00 00 00 28 key_size	6D 64 74 61 key_namespace mdta	63 6F 6D 2E 61 70 70 6C 65 2E 71 75 69 63 6B 74 69 6D … key_name (Apple calls it key_value) com.apple.quicktime.creationdate							
Key value 2	00 00 00 21 key_size	64 61 74 61 key_namespace mdta	63 6F 6D 2E 61 70 70 6C 65 2E 71 75 69 63 6B 74 69 6D … key_name com.apple.quicktime.model							
Key value 3	00 00 00 24 key_size	A9 6E 61 6D key_namespace mdta	63 6F 6D 2E 61 70 70 6C 65 2E 71 75 69 63 6B 74 69 6D … key_name com.apple.quicktime.location.ISO6709							
Metadata Item List Atom	00 00 00 xx size	69 6C 73 74 type ilst								
Metadata Item Atom	00 00 00 30 size	00 00 00 01 type =key 1								
Value atom	00 00 00 28 size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	46 52 1A 41 locale indicator country language	32 30 31 34 2D 30 37 2D 30 35 54 … type=1: string 2014-07-05T13:02:04+0200					
Metadata Item Atom	00 00 00 21 size	00 00 00 02 type =key 2								
Value atom	00 00 00 19 size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	46 52 1A 41 locale indicator country language	69 50 68 6F 6E 65 20 35 73 type=1: string iPhone 5s					
Metadata Item Atom	00 00 00 32 size	00 00 00 03 type =key 3								
Value atom	00 00 00 2A size	64 61 74 61 type data	00 00 00 01 type indicator DF wknown types	46 52 1A 41 locale indicator country language	2B 34 33 2E 36 35 32 31 2B 30 30 … type=1: string +43.6521+003.3638+148.202/					
Free Atom	00 00 04 00 size	66 72 65 65 type free	00 00 00 00 00 00 00 00							

3.2 User data atom

A User Data Atom whose immediate parent is a movie atom contains data relevant to the movie as a whole. A User Data Atom whose parent is a track atom contains information relevant to that specific track. A QuickTime movie file may contain many user data atoms, but only one User Data Atom is allowed as the immediate child of any given movie atom or track atom.

A User Data Atom whose immediate parent is a movie atom contains data relevant to the movie as a whole. A User Data Atom whose parent is a track atom contains information relevant to that specific track. A QuickTime movie file may contain many user data atoms, but only one user data atom is allowed as the immediate child of any given movie atom or track atom (2).

The series of atoms inside a User Data Atom is also referred to as 'user data list'. Each data element in the user data list contains size and type information along with its data. An example is shown in Table 3.

Table 3: Example of user data atom that also contains a meta data atom. File: [4]

Atom	Size	Type	Data
User data atom	00 00 01 CA size	75 64 74 61 type udta	
User data Item Atom	00 00 00 10 size	53 44 4C 4E type SDLN	53 45 51 5F 50 4C 41 59 SEQ_PLAY
User data Item Atom	00 00 00 10 size	73 6D 72 64 type smrd	54 52 55 45 42 4C 55 45 TRUEBLUE
User data Item Atom	00 00 00 1E size	A9 78 79 7A type ©xyz	00 12 (size 18) · 15 C7 (langua. eng') · 2B 35 30 2E 39 36 37 38 2D 31 31 34 2E 30 36 39 30 2F +50.9678-114.0690/
User data Item Atom	00 00 00 1A size	73 6D 74 61 type smta	00 00 00 00 00 00 00 0E 73 61 75 74 00 00 00 00 00 00 saut
Metadata atom	00 00 00 97 size	6D 65 74 61 type meta	00 (v) · 00 00 00 (flags)
Metadata handler atom	00 00 00 20 size	68 64 6C 72 type hdlr	00 (v) · 00 00 00 (flags) · *00 00 00 00 predefined=0* · 6D 64 69 72 (handler type mdir) · *00 00 00 00 Reserved=0* · *00 00 00 00 Reserved=0* · *00 00 00 00 Reserved=0*
Metadata Item List Atom	00 00 00 6B size	69 6C 73 74 type ilst	
Metadata Item Atom	00 00 00 35 size	A9 6E 61 6D type ©nam	
Value atom	00 00 00 2D size	64 61 74 61 type data	00 00 00 01 (type indicator: DF, wknown types) · 00 00 00 00 (locale indicator: country, language) · 41 6E 64 72 6F 69 64 20 36 2E .. .. type=1: string "Android 6.0.1 tags by Windows"
Metadata Item Atom	00 00 00 2E size	A9 63 6D 74 type ©cmt	
Value atom	00 00 00 26 size	64 61 74 61 type data	00 00 00 01 (type indicator: DF, wknown types) · 00 00 00 00 (locale indicator: country, language) · 63 6F 6D 6D 65 6E 74 20 73 68 .. .. type=1: string "comment should be here"

A User Data Atom can also be the container of a Meta Data Atom as is shown in Table 3. Now, an important part that is *different* from the data inside the Meta Data Atom (that also can contain atomtypes starting with ©) is the following:

All user data list entries whose type begins with the © character (ASCII 169) are defined to be international text. These list entries must contain a list of text strings with associated language codes. By storing multiple versions of the same text, a single user data text item can contain translations for different languages. The list of text strings uses a small integer atom format, which is identical to the QuickTime atom format, except that it uses 16-bit values for size and type instead of 32-bit values. **The first value is the size of the string, including the size and type***, and the second value is the language code for the string.

What is meant is probably the following: In the 'Quicktime atom format' (Table 3c) the data is stored in a Value Atom with type 'data' including a 4-byte type indicator (1 for string) and a 4-byte locale indicator. When you reshuffle the bytes of the User Data Item Atom you can place the data in a 'pseudo' Value Atom like in (Table 3b).

***The first value is the size of the string, including the size and type**: type indicator is 1 by definition, type of atom is 'data' by definition.

Table 4: Difference between an Item Atom in User data and Meta data (see text)

a. atom rooted in \udta\

User data Item Atom	00 00 00 1E	A9 78 79 7A	00 12	15 C7	2B 35 30 2E 39 36 37 38 2D 31 31 34 2E 30 36 39 30 2F
	size	type ©xyz	size 18	langua. eng'	+50.9678-114.0690/

b. same atom rooted in \udta\ , but data reorganised to match pattern of 'Quicktime atom format' shown under c.

User data Item Atom	00 00 00 1E	A9 78 79 7A			
	size	type ©xyz			
Pseudo Value atom	- - 00 12	- - - -	- - - -	- - 15 C7	2B 35 30 2E 39 36 37 38 2D 31 .. ..
	size	type 'data' by defin.	type indicator 1 by defin.	locale indicator	type=1: string +50.9678-114.0690/

c. another atom rooted in \udta\meta\ilst\ which is referred to as 'Quicktime atom format'

Metadata Item Atom	00 00 00 35	A9 6E 61 6D			
	size	type ©nam			
Value atom	00 00 00 2D	64 61 74 61	00 00 00 01	00 00 00 00	41 6E 64 72 6F 69 64 20 36 2E .. ..
	size	type data	type indicator (DF, wknown types)	locale indicator (country, language)	type=1: string "Android 6.0.1 tags by Windows"

User data text strings may use either Macintosh text encoding or Unicode text encoding. The format of the language code determines the text encoding format. Macintosh language codes are followed by Macintosh-encoded text. If the language code is specified using the ISO language codes listed in specification ISO 639-2/T, the text uses Unicode text encoding. When Unicode is used, the text is in UTF-8 unless it starts with a byte-order-mark (BOM, 0xFEFF), in which case the text is in UTF-16. Both the BOM and the UTF-16 text should be big-endian. Multiple versions of the same text may use different encoding schemes.

Now I ask myself: what happens if you put a ‘Quicktime format’ value atom in the root of a udta atom? For this I created a testfile using a hexeditor with that content. Most software like Ffprobe, Mediainfo, Exiftool and Online MP4 parser don’t read it correctly. As they should. By definition the atoms with the root \udta\ should be in the short format it seems.

3.3 Microsoft Xtra atom

When an MP4 or MOV file is edited by Windows Properties in the ‘Details’ tab of (right mouse), an ‘Xtra’ atom is added or changed as \\moov\udta\Xtra. Besides this, several values are also added or changed in the \\moov\udta\meta atom.

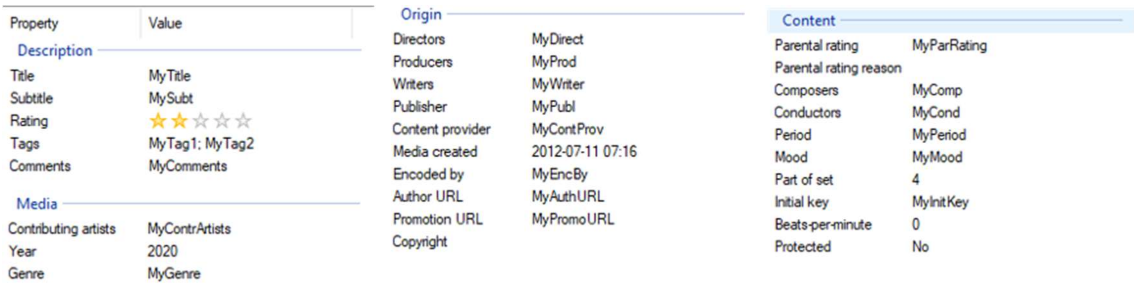

Figure 1: Properties displayed in the Windows property editor by Windows.

Table 5: Properties displayed in the Windows property editor and the atoms where they are stored.

Section	Property	Xtra key	moov/udta/meta key
Description			
	Title	-	©nam
	Subtitle	WM/SubTitle	-
	Rating	WM/SharedUserRating	-
	Tags	-	-
	Comments	-	©cmt
Media			
	Contributing artists	-	©ART
	Year	-	©day
	Genre	-	©gen
Origin			
	Directors	WM/Director	-
	Producers	WM/Producer	-

	Writers	WM/Writer	-
	Publisher	WM/Publisher	-
	Content provider	WM/ContentDistributor	-
	Media created	*cannot be set*	-
	Encoded by	WM/EncodedBy	-
	Author URL	WM/AuthorURL	-
	Promotion URL	WM/PromotionURL	-
	Copyright	*cannot be set*	-
Content			
	Parental rating	WM/ParentalRating	-
	Parental rating reason	*cannot be set*	-
	Composers	-	©wrt
	Conductors	WM/Conductor	-
	Period	WM/Period	-
	Mood	WM/Mood	-
	Part of set	-	disk
	Initial key	WM/InitialKey	-
	Beats-per-minute	-	tmpo
	Protected	*cannot be set*	-

Information is not easy to find. The data can be stored in various types that are indicated by a type enumeration (like well-know types in the meta atom).

I cannot find this information as you would expect it on the Microsoft website. Although many 'keys' that are used in the Xtra atom (e.g. WM/Composer) are described in the WMF Attribute list (3) including a type enumeration (WMT_ATTR_DATATYPE enumeration); part of this enumeration is as follows: WMT_TYPE_DWORD = 0, WMT_TYPE_STRING = 1, WMT_TYPE_BINARY = 2 etc. However, this is not the enumeration used in the Xtra atom.

The actual enumeration used I found in Exiftool source code (4) and in an Xtrabox Java script (5), and is shown in Table 6. As noted in (4), an implementation has existed in a branch of mp4v2 but has been removed. This is discussed in (6).

Table 6: Type enumeration of Xtra values

Const Name	Decimal	Hexadecimal
MP4_XTRA_BT_UNICODE	8	$8
MP4_XTRA_BT_INT64	19	$13
MP4_XTRA_BT_FILETIME	21	$15
MP4_XTRA_BT_GUID	72	$48

Table 7: Example of an Xtra atom

Element	Bytes	Field	Value
Xtra atom	00 00 03 14	size	788
	58 74 72 61	type	Xtra
Key 1	00 00 00 2D	key_size	45
	00 00 00 13	key_name_len	
	57 4D 2F 53 68 61 72 65 64 55 73 65 72 52 61 74 69 6E 67	key_name	WM/SharedUserRating
	00 00 00 01	entry_count	1
Value 1	00 00 00 0E	value_size	
	00 13	val_type	19
	19 00 00 00 00 00 00 00	value	
Key 2	00 00 00 2B	key_size	43
	00 00 00 0B	key_name_len	
	57 4D 2F 53 75 62 54 69 74 6C 65	key_name	WM/SubTitle
	00 00 00 01	entry_count	1
Value 1	00 00 00 14	value_size	
	00 08	val_type	8
	4D 00 79 00 53 00 75 00 62 00 74 00 00 00	value	M . y . S . u . b . t . . .
Key 3	00 00 00 3F	key_size	63
	00 00 00 0B	key_name_len	11
	57 4D 2F 43 61 74 65 67 6F 72 79	key_name	WM/Category
	00 00 00 02	entry_count	2
Value 1	00 00 00 14	value_size	20
	00 08	val_type	8
	4D 00 79 00 54 00 61 00 67 00 31 00 00 00	value	M . y . T . a . g . 1 . . .
Value 2	00 00 00 14	value_size	20
	00 08	val_type	8
	4D 00 79 00 54 00 61 00 67 00 32 00 00 00	value	M . y . T . a . g . 2 . . .
Key 4	00 00 00 27	key_size	
	00 00 00 07	key_name_len	
	57 4D 2F 4D 6F 6F 64	key_name	WM/Mood
	00 00 00 01	entry_count	1
Value 1	00 00 00 14	value_size	
	00 08	val_type	8
	4D 00 79 00 4D 00 6F 00 6F 00 64 00 00 00	value	M . y . M . o . o . d . . .

3.4 EXIF tag data in general

The next section will be about the Nikon NCTG atom that contains data stored in an EXIF format. Therefore first this section, because this is more general and will also exist in other atoms.

EXIF data is stored using a tag and then the data, like this:

tag	val_type	count	data

In this section the tag is not discusssed, only how the value is stored and how to read it.

Like in the other containers of data, EXIF also works with a type enumeration. I call these enumerators 'val_type' to keep consistency, but they are specific to EXIF and also introduce the "rational", wich is a fraction of two 4-byte integers stored inside an 8-byte variable (7). This enumeration is shown in Table 8.

The 2-byte 'type' is always followed by a 2-byte 'count'. In structures described until now, the length of the total data is usually indicated by one value. In case EXIF data you have to find this by multiplying:

BytesToRead=(# bytes specified by 'val_type' enumerator) x ('count').

As regards the 'tag'. EXIF data is stored using tags that are very specific for each case. So when inside an NCTG atom the tag 1 can mean the 'Make of the camera' while in a completely other case it can mean 'depth under sea level'. The example shown in the next section will demonstrate how this works.

Table 8: EXIF data type enumeration (7)

Enumerator	Format	# bytes	Comment
1	unsigned byte	1	
2	ascii strings	1	
3	unsigned short	2	
4	unsigned long	4	
5	unsigned rational	8	"rational" is fractional value, first (u)int32 is numerator, 2nd (u)int32 is denominator
6	signed byte	1	
7	undefined	1	
8	signed short	2	
9	signed long	4	
10	signed rational	8	
11	single float	4	
12	double float	8	

3.5 Nikon NCTG atom

The Nikon tags are stored in an EXIF format (see 3.4). As described, for EXIF you need specific tables to find the meaning of each tag. They are just stored as numbers and can mean anything in every different case.

An example of EXIF data is the NCTG atom. This clearly demonstrates that you need a table to correlate what each tag means. In this case 1 means the make of the camera.

Table 9: Tag enumeration inside an NCTG atom (just parts; outtake of the total list)

1	Make
2	Model
3	Software
17	CreateDate
18	DateTimeOriginal
19	FrameCount
22	FrameRate
25	TimeZone
34	FrameWidth
17859226	ExposureTime
17859229	FNumber
17860642	ExposureProgram

A real-life example is shown in

Table 10. A simple example is Tag 1. The tag is 1. Lookup-value in Table 9 shows that it is the 'Make'. The val_type is 2, lookup-value in Table 8 shows that it is a string. The count is 6, so 6 characters to read (including \0, so it is a

null-terminated C type string). Strangely, Tag 2 is terminated by two zeros. In a similar way, as a string, the Createdate is stored (Tag 7). Notice that semicolons are use as separators.

An example of a ‘rational’ is shown in Tag 5. The tag enumerator is $17 (dec 23), which is not in tables I have found, but it looks like it is also the Framerate. The val_type is 5, a rational that is 8 bytes wide (Table 8). The four-byte numerator is 50000, the four-byte denominator is 1000, so the value is 50.

Table 10: Specific example of an NCTG atom (composed of different tags encountered)

NCTG atom	00 00 xx xx	4E 43 54 47		
	size	type		
		NCTG		
Tag 1	00 00 00 01	00 02	00 06	4E 49 4B 4F 4E 00
	tag	val_type	count	data
	1=Make	2	6	N I K O N .
Tag 2	00 00 00 02	00 02	00 0E	43 4F 4F 4C 50 49 58 20 50 39 30 30 00 00
	tag	val_type	count	data
	2=model	2	14	C O O L P I X P 9 0 0 . .
Tag 3	00 00 00 03	00 02	00 16	43 4F 4F 4C 50 49 58 20 50 39 30 30 20 56 65 72 2E 31 2E 34 31 00
	tag	val_type	count	data
	3=softw.	2	22	C O O L P I X P 9 0 0 V e r . 1 . 4 1 .
Tag 4	00 00 00 13	00 04	00 02	00 00 00 4B 00 00 00 00
	tag	val_type	count	data
	$13=framecount	4	2	. . . K . . . .
Tag 5	00 00 00 17	00 05	00 01	00 00 C3 50 00 00 03 E8
	tag	val_type	count	data
	$17=framerate?	5	1	d uint32: 50000 / d uint32:1000
Tag 6	00 00 00 18	00 03	00 01	00 01
	tag	val_type	count	data
	$18=unkn now	3	1	1
Tag 7	00 00 00 11	00 02	00 14	32 30 31 39 3A 30 38 3A 31 30 20 31 34 3A 34 32 3A 35 34 00
	tag	val_type	count	data
	17=Createdate	2	20	2 0 1 9 : 0 8 : 1 0 1 4 : 4 2 : 5 4 .

---

1 https://en.wikipedia.org/wiki/MPEG-4_Part_14#

2 MOV_0234-windowscomments.mp4

3 Apple-Iphone5s.mov

4 Android 6.0.1 - with tags added by Windows Explorer.mp4

1. **Wikipedia.** Wikipedia - QuickTime File Format. [Online] https://en.wikipedia.org/wiki/QuickTime_File_Format.

2. **Apple Computer, Inc.** *QuickTime File Format.* Cupertino, CA : s.n., 2002.

3. **Microsoft.** Windows Media Format 11 Attribute List. [Online] [Cited: 6 14, 2020.] https://docs.microsoft.com/en-us/windows/win32/wmformat/attribute-list.

4. **Harvey, Phil.** Perlscript "Microsoft.pm". [Online] [Cited: 6 14, 2020.] https://github.com/exiftool/exiftool/blob/master/lib/Image/ExifTool/Microsoft.pm.

5. "XtraBox.java" script. [Online] [Cited: 6 14, 2020.] http://www.java2s.com/example/java-src/pkg/com/googlecode/mp4parser/boxes/microsoft/xtrabox-3706e.html.

6. mp4v2 - issue #113. [Online] 8 5, 2011. [Cited: 6 14, 2020.] https://code.google.com/archive/p/mp4v2/issues/113.

7. Description of Exif file format. [Online] [Cited: 6 24, 2020.] https://www.media.mit.edu/pia/Research/deepview/exif.html.

8. Harvey, Phil. Perlscript "Quicktime.pm". [Online] https://github.com/alchemy-fr/exiftool/blob/master/lib/Image/ExifTool/QuickTime.pm.