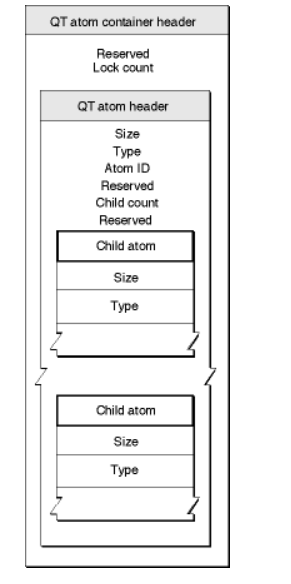
# Overview

* Quicktime movies are stored on disk using atoms and QT atoms. A quicktime file stores the description of the media separately from the media data. The description is called the movie resource or movie atom and contains information such as number of tracks, video compression format, and timing information.
* Each atom consists of a size and type field. The size field is 4 bytes and type field is 4 bytes. Size represents the size of the atom including the size and type fields.
* If the size field is 0 it represents the last atom extending till end of the file. If the size field is 1, it represents the extended size is stored in 8 bytes just after the type field.
* An atom can be heirarchial. Example a movie atom contains one track atom for each track in the movie. The track atom can contain one media atom each, along with other atoms that define other track characteristics.
* An atom that does not contain other atoms is called a leaf atom and typically contains data as one or more fields or tables. Some leaf flags act as flags or placeholders, however, contain no data beyond their size and type fields.
* QT atoms are enhanced data structures that provide a more general purpose storage format. A QT atom has an extended header: the size and type fields followed by fields for an atom id and number of child atoms. This allows multiple child atoms of the same type to be specified through identification numbers. QT atoms are normally wrapped in an atom container, a data structure with a header containing a lock count. Each atom container contains exactly one root atom which is the QT atom. Atom containers are not atoms and are not found in the hierarchy of atoms that make up the Quicktime movie file. Atom containers may be found as data structures within some atoms (like the media input maps and media property atoms). Each QT atom starts with a QT atom header followed by the atom’s content. The contents are either child atoms or data, but never both.



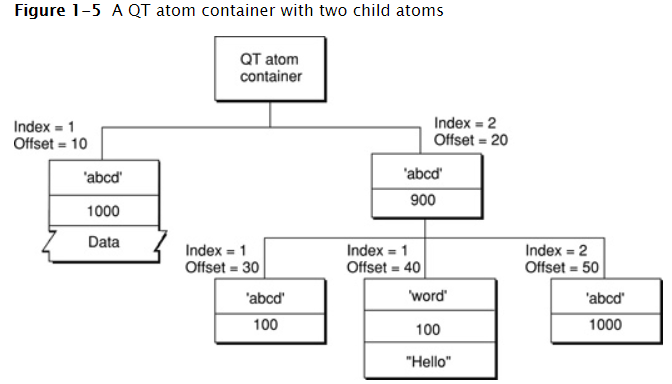
A QT atom container header contains the following:

1. Reserved: 10-byte element that must be set to 0.
2. Lock count: A 16byte integer that must be set to zero.

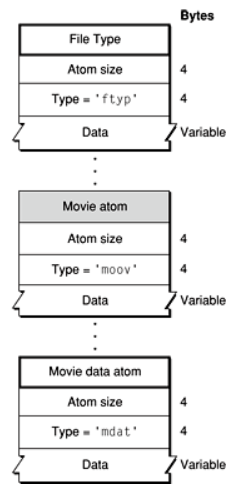
A QT atom header contains the following section:

1. Size: A 4 byte integer indicates the size of atom (including size of header and all constituent atoms).
2. Type: 4 byte element that contains the type of the atom.
3. Atom ID: 4 byte integer that contains the atoms ID value. The value must be unique among it’s siblings. The root atom always has an atom id 1.
4. Reserved: 2 byte integer that must be set to 0.
5. Child Count: A 2 byte integer that specifies the number of child atoms that an atom has. This count includes only immediate children.
6. Reserved: A 4 byte integer that must be set to 0.

An atom container has below structure:



* A quicktime movie file is structured as a collection of atoms. The file format is extensible and from time to time new formats are introduced. This allows file format to be extended without breaking existing applications. Generally atoms can be present in any order. An exception is the file type atom, which identifies the file as a Quicktime movie. If present this atom precedes any movie atom, movie data, preview or space atoms. If you find one of the other atom types prior to finding a file atom type it indicates the file atom type is not present. (All files created prior to 2004 do not contain this atom). While other atoms can be in any order, unless specified in the document, for practical reasons there is a recommended order that you should use when creating a QuickTime movie file. For example the atom containing the movie resource should precede any atoms containing the movie’s sample data. By this it is possible to play a movie over a network while the movie file is in the process of downloading. A quicktime movie file must contain a movie atom, which contains either the movie structure or a reference to one or more movie sources external to the file. These alternate sources will be quicktime movie files containing the movie structure. A quicktime movie file typically contains one or more movie data atoms, which contains media sample data such as video frames and group of audio samples. Below shows the essential atom types in a QuickTime movie file within which other atoms are stored. In addition, the file may contain free space atoms, preview atoms, and other atoms not enumerated in this file format specification.



|  |  |
| --- | --- |
| Basic atom types of a QuickTime file | |
| Atom type | Use |
| 'ftyp' | File type compatibility—identifies the file type and differentiates it from similar file types, such as MPEG-4 files and JPEG-2000 files. |
| 'moov' | Movie resource metadata about the movie (number and type of tracks, location of sample data, and so on). Describes where the movie data can be found and how to interpret it. |
| 'mdat' | Movie sample data—media samples such as video frames and groups of audio samples. Usually this data can be interpreted only by using the movie resource. |
| 'free' | Unused space available in file. |
| 'skip' | Unused space available in file. |
| 'wide' | Reserved space—can be overwritten by an extended size field if the following atom exceeds 2^32 bytes, without displacing the contents of the following atom. |
| 'pnot' | Reference to movie preview data. |

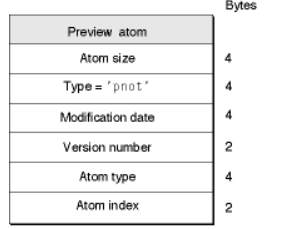
The file type atom has an atom type value as ‘ftyp’ and contains the following fields:

1. Size: 4 byte
2. Type: 4 byte element. Must be set to ‘ftyp’
3. Major Brand: A 4 byte integer that must be set to ‘qt ‘ for Quicktime movie files, ‘mmp4’ for MP4 movie files. If a file is compatible with multiple brands, all such brands are listed in the compatible brands section and major brand specifies the preferred brand to use.
4. Minor version: A 4 byte field that indicates the file format specification version. For quicktime movie files this needs to be set as the year and month of the quicktime file format specification followed by a binary coded decimal 0. Example for June 2004 minor version, this field is set as 20 04 06 00
5. Compatible brands[]: A series of unsigned 4 byte integers listing compatible file formats. The major brand must appear in the list of compatible brands.

The free and skip atoms designate unused space in the movie data file. These atoms consist of only an atom header (size and type fields) followed by the appropriate number of bytes for free space. When reading a quicktime movie the application can safely ignore these atoms.

The movie data atom consists of the atom header followed by the movie’s media data. Your application can understand the data in this atom only by using the metadata stored in the movie atom.

The preview atom contains information that allows you to find preview image associated with a quicktime movie. Below is the structure for a preview atom



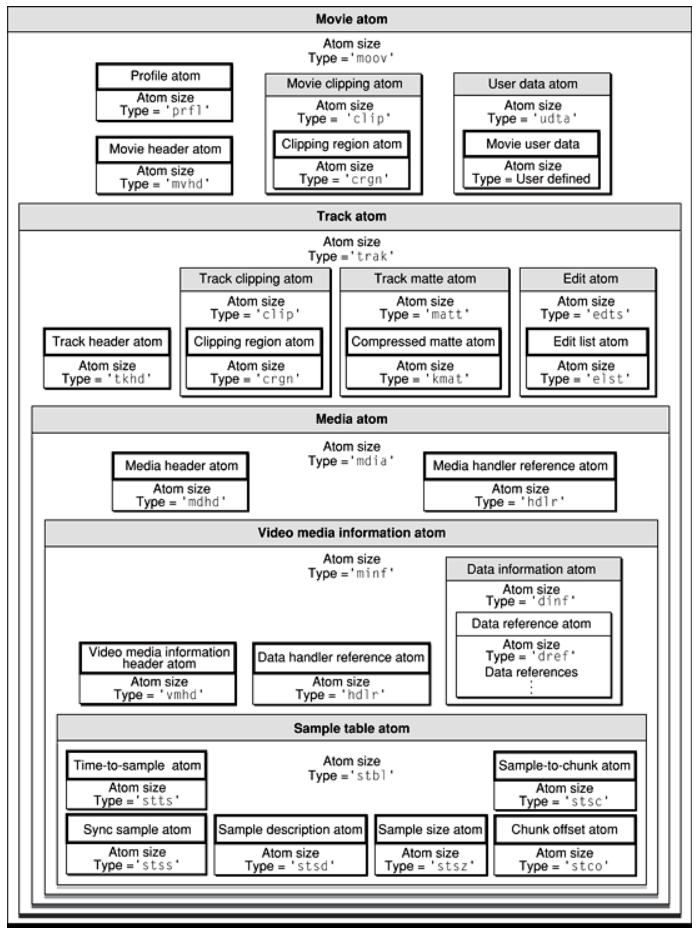
The preview atom has below fields:

1. Size: 4 byte
2. Type: 4 byte. Should be set to ‘pnot’
3. Modification Date: A 4byte integer containing date that indicates when the preview was last updated. The data is in standard Machintosh format.
4. Version number: A 2 byte integer that must be set to 0.
5. Atom type: A 4 byte integer that indicates the type of atom that contains the preview data. Typically this is set to ‘PICT’ to indicate a quickdraw picture.
6. Atom index: A 2 byte integer that identifies which atom of the specified type to be used as a preview. Typically set as 1.

# Movie Atoms

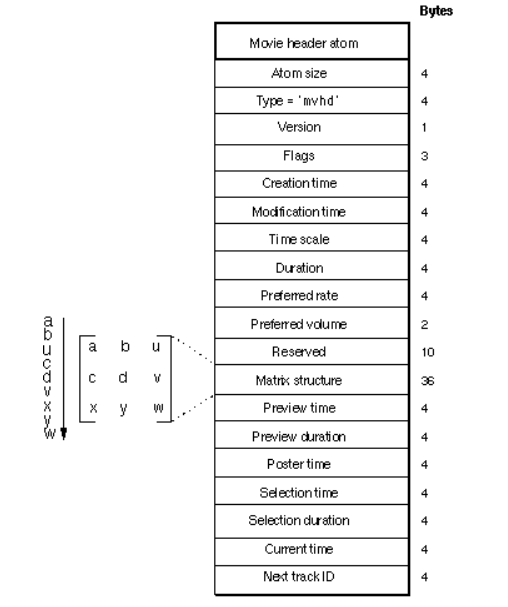
The quicktime movie atoms have an atom type of ‘moov’. They act as containers for the movie information that described a movie’s data. This information or metadata is stored in a number of different types of atoms. Only metadata is stored in a movie atom. Sample data for the movie such as the audio and video samples are referenced in a movie atom but not contained in it. The movie atom is essentially a container for other atoms. These atoms taken together describe the contents of a movie. At the highest level, a movie atom consist of track atoms which inturn contains media atoms. At the lowest level are the leaf atoms which contain non-atom data, usually in the form of a table or a set of data elements. Movie atom allows your application to interpret the sample data that is stored elsewhere.

Below is a sample movie atom.



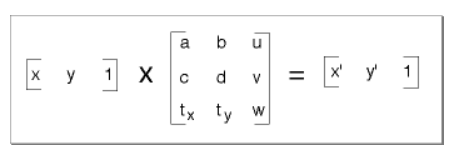
The movie atom consists of:

1. Movie header atom: defines the time scale and duration information for the entire movie, as well as its display characteristics. The movie atom contains a track atom for each track of the movie. It has an atom type of ‘mvhd’. The movie header atom is a leaf atom. Below is the structure of a movie header atom.

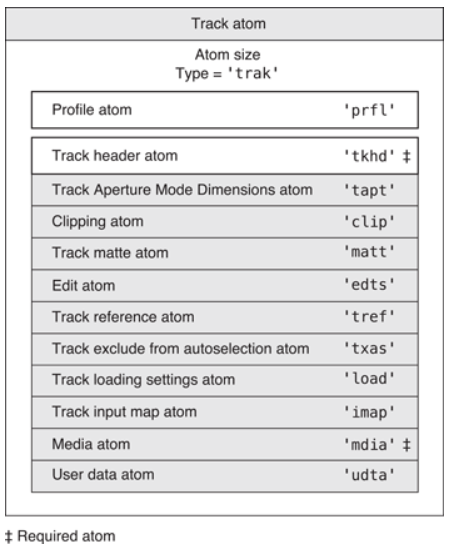


Below is it’s specification:

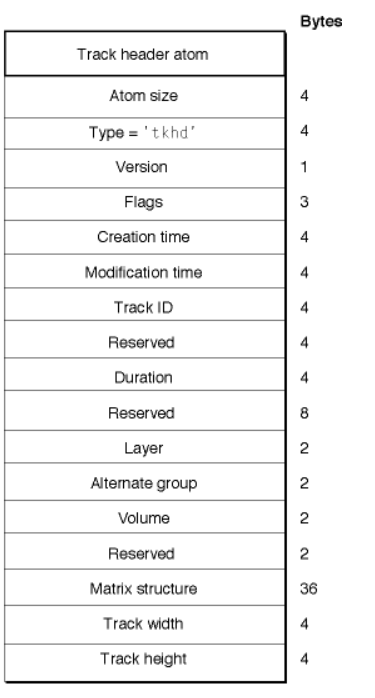
1. Size: A 4 byte element
2. Type: 4 byte element. Should be set to ‘mvhd’
3. Version: 1 byte specification of the version of this movie header.
4. Flags: Three bytes of space for future movie header flags.
5. Creation Time: A 4 byte element that specifies calendar date and time (in seconds since midnight January 1, 1904) when the movie atom was created. It should be specified in UTC.
6. Modification Time: A 4 byte integer that specifies the calendar date and time (in seconds since midnight January 1, 1904) when the movie was modified. It should be in UTC.
7. Time scale: A 4 byte time value that indicates the time scale for the movie – that is the number of time units that pass per second in its time coordinate system. A time coordinate system that measures time in sixtieths of a second, for example, has a time scale of 60. (600 for the monsters inc and 4100 for the lync recording. A standard value for this field is 600, which permits non-fractional values for most of the common movie rates. (For example a movie playing at 30 frames per second has 20 units per frame, if the time scale is 600 – “https://books.google.com/books?id=WwzK0JopNNAC&pg=PA68&lpg=PA68&dq=quicktime+movie+time+scale+how+is+it+determined&source=bl&ots=hPRdKGDzrU&sig=sUmNSHK3etSgAa6sXQ7ZMWmiMqw&hl=en&sa=X&ved=0ahUKEwj4pby7uf\_XAhXJz4MKHZ9dDikQ6AEITDAG#v=onepage&q=quicktime%20movie%20time%20scale%20how%20is%20it%20determined&f=false”)). NTSC color video is not 30 frames-per-second (fps), but actually 29.97 fps. By setting the media’s time scale to 2997 units per second and setting the frame durations to 100 units each, the effective rate is 29.97 fps exactly. (<https://developer.apple.com/standards/qtff-2001.pdf>). For audio tracks at 44.1khz, the time scale can be set as 44100.
8. Duration: A 4 byte time value that indicates the duration of the movie in time scale units. Note that this property is derived from the movie tracks. The value of this field corresponds to the duration of the longest track in the movie.
9. Preferred Rate: A 4 byte fixed point number that indicates the rate at which to play this movie. A value of 1 indicates a normal rate.
10. Preferred Volume: A 2 byte fixed point number that indicates how loud to play this movie’s sound. A value of 1 indicates full volume.
11. Reserved: 10 bytes reserved for use by Apple. Set to 0
12. Matrix structure: The matrix structure associated with the movie. A matrix shows how to map points from one coordinate space to another. All values in the matrix are 32 bit fixed point numbers divided as 16.16, except for {u, v, w} column which contains 32 bit fixed point numbers divided as 2.30.



1. Preview time: The 4 byte time value in the movie at which the preview begins.
2. Preview duration: The 4 byte duration of the movie preview in movie time scale units.
3. Poster time: The 4 byte time value of the time of the movie poster.
4. Selection time: The 4 byte time value for the start time of the current selection.
5. Selection duration: The 4 byte duration of the current selection in movie time scale units.
6. Current time: The 4 byte time value for the current time position within the movie.
7. Next track id: A 4 byte integer that indicates a value to use for the track id number of the next track added to the movie. Note that 0 is not a valid track id value. (Set as 3 in sample movie)
8. Track atom: Track atom defines a single track of the movie. Each track atom consists of its associated media atom. Track atoms have an atom type of ‘trak’. The track atom requires a track header ‘tkhd’ and a media atom ‘mdia’. Other child atoms are optional, and may include track clipping atom ‘clip’, a track matte atom ‘matt’, an edit atom ‘edts’, a track reference atom ‘tref’, a track load settings atom ‘load’, a track input map atom ‘imap’, and a user data atom ‘udta’. Below is the structure of a track atom

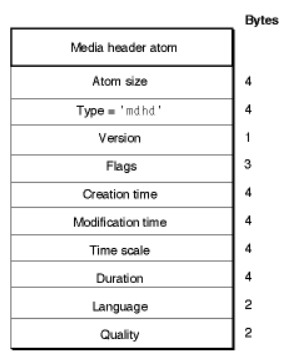


1. Track header atom: It specifies the characteristics of a single track within the movie. Track header atom consists of the following data elements:

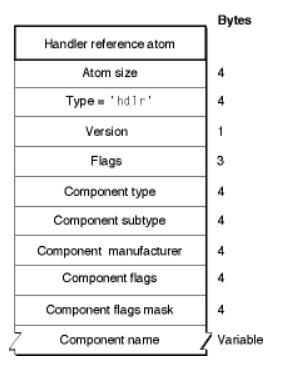


1. Size: 4 byte
2. Type: 4 byte. Must be set to ‘tkhd’
3. Version: A 1 byte specification of the version of this track header.
4. Flags: Three bytes that are reserved for the track header flags. These flags indicate how the track is used in the movie. The following flags are valid (all flags are enabled when set to 1):
   1. Track enabled: Indicates whether the track is enabled.
   2. Track in movie: Indicates that the track is used in the movie. Flag value is 0x0002.
   3. Track in preview: Indicates that the track is used in movie’s preview. Flag value is 0x0004.
   4. Track in poster: Indicates that the track is used in the movie’s poster. Flag value is 0x0008.
5. Creation time: A 4 byte integer that indicates the calendar date and time (expressed as seconds since midnight, January 1, 1904) when the track header was created. Should be in UTC
6. Modification time: 4 byte. All times can be kept same as movie.
7. Track Id: A 4 byte integer that uniquely identifies the track. This value cannot be set to 0.
8. Reserved: A 4 byte integer that is reserved for future use.
9. Duration: A 4 byte time value that indicates the duration of this track (in the movie’s time coordinate system). Note that this value is derived from the track edits. The value of this field is equal to the sum of the durations of all the track edits. If there is no edit list, then the duration is the sum of the sample durations, converted into the movie timescale.
10. Reserved: An 8 byte field that is reserved for future use.
11. Layer: A 2 byte integer that indicates the tracks spatial priority in its movie. The quicktime movie toolbox uses this value to determine how tracks overlay one another. Tracks with lower layer values are displayed in front of the tracks with higher layer values.
12. Alternate group: A 2 byte integer that identifies a collection of movie tracks that contain alternate data for one another. The same identifier appears in each ‘tkhd’ atom of the other tracks in the group. Quicktime chooses one track from the group to be used when the movie is played. The choice may be based on the considerations as playback quality, language or the capabilities of the computer. A value of 0 indicates that the track is not in an alternate group. The most common reason for having alternate tracks is to provide versions of the same track in different languages.
13. Volume: A 2 byte fixed point value that indicates how loud the track’s sound is to be played. A value of 1.0 indicates normal volume.
14. Reserved: A 2 byte integer reserved for future use.
15. Matrix structure: The matrix structure associated with this track.
16. Track width: A 4byte fixed point number that specifies the width of this track in pixels.
17. Track height: A 4 byte fixed point number that specifies the height of this track in pixels. (Both can be set to 0)
18. Media Atom: Media atom describes a track’s media type and sample data. It specifies the media type (sound, video or timed metadata), the media handler component used to interpret the sample data, the media timescale and duration. The media atom has an atom type of ‘mdia’. It must contain a media header atom (‘mdhd’). Do not confust the media atom (‘mdia’) with the media data atom (‘mdat’). The media atom contains only references to media data, the media data atom contains the actual media samples.

The media header atom has below layout:



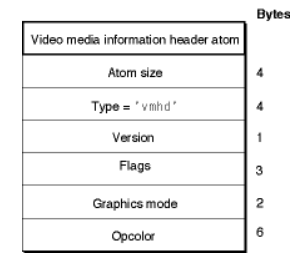
1. Size: 4 byte
2. Type: 4 byte. Must be set to ‘mdhd’.
3. Version: 1 byte that specifies the version of the header atom.
4. Flags: Three bytes of space to set the media header flags. Set this to 0.
5. Creation time: 4 bytes
6. Modification time: 4 bytes
7. Time scale: 4 bytes
8. Duration: 4 bytes duration in its time scale.
9. Language: A 2 byte integer that specifies the language code for the media. (55 c4 or 15 c7)
10. Quality: A 2 byte integer that specifies the media playback quality.
11. Handler Reference atom: The handler reference atom specifies the media handler component that is to be used to interpret the media’s data. It has atom type as ‘hdlr’. It has following components:



1. Size
2. Type: ‘hdlr’
3. Version: 1 byte
4. Flags: 3 byte. Set this to 0.
5. Component type: 4 bytes that specify the type of the handler. Only two values are allowed for this field: ‘mhlr’ for media handler and ‘dhlr’ for data handler. This field can be left 0.
6. Component subtype: 4 byte that specifies the type of the media handler or data handler. For media handler the field defines the type of data – ex. ‘vide’ for video data, ‘soun’ for sound data or ‘subt’ for subtitle. For data handlers, this field defines the data reference type – Ex. A component subtype value of ‘alis’ identifies a file alias.
7. Component manufacturer: 4 byte. Reserved. Set to 0.
8. Component flags: 4 byte. Reserved. Set to 0.
9. Component flags mask: 4 byte. Reserved. Set to 0.
10. Component name: A counted string that specifies the name of the component (the media handler used when the media was created). This field may contain a zero length string. (Sample value: VideoHandler)
11. Media information atom: Media information atoms (‘minf’) stores handler specific information for a track’s media data. The media handler uses this information to map from media time to media data and to process the media data. These atoms contain information that is specific to the type of data defined by the media.

The video media information atoms are the highest level atoms in video media. These atoms contain a number of other atoms that define specific characteristics of the video media data.

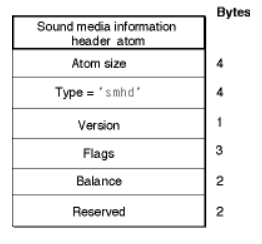
The video media information header atom define specific color and graphics mode information.

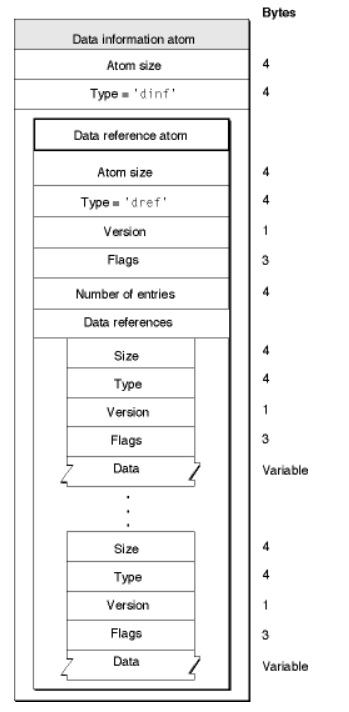


It contains following data elements:

1. Size
2. Type: vmhd
3. Version: 1 byte
4. Flags: 3 bytes for video media information flags. There is one defined flag: No lean ahead (This is a compatibility flag that allows quicktime to distinguish between movies created with Quicktime 1.0 and newer movies. You should always set this flag to 1, unless you are creating a movie intended to playback with Quicktime 1.0.
5. Graphics mode: A 2 byte integer that specifies the transfer mode (Boolean operation quickdraw should perform when drawing or transferring an image from one location to another).
6. OpColor: Three 2 byte values that specify the red, green and blue colors for the transfer mode operation indicated in the graphics mode field.

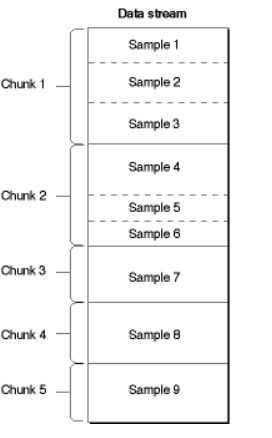
The sound media information header atoms contain following data elements:



1. Size
2. Type: smhd
3. Version: 1 byte
4. Flags: 3 bytes. Set this field to 0.
5. Balance: A 2 byte integer that specifies the sound balance of this sound media. Sound balance is the setting that controls the mix of sound between the two speakers of a computer. This field is normally set to 0.
6. Reserved: 2 bytes reserved for future.
7. Data information atom: The data handler component uses the data information atom to interpret the media’s data. It has an atom type of ‘dinf’. 

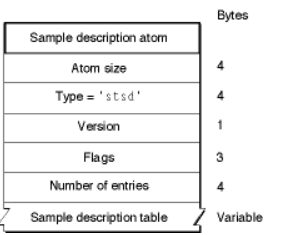
Below is the structure of the data reference atom:

1. Size
2. Type: Must be set to ‘dref’.
3. Version: 1 byte
4. Flags: 3 bytes. Must be set to 0.
5. Number of entries: 4 byte integer that specifies the count of data references that follow.
6. Data references: An array of data references. Each data reference is formatted like an atom and contains the following data elements.
   1. Size
   2. Type: Can be set to ‘alis’ (The data reference is a Macintosh alias. An alias contains information about the file it refers to, including its full file name), ‘rsrc’ (deprecated), ‘url ‘ (A C string that specifies a URL, there may be additional data after the C string).
   3. Version: 1 byte
   4. Flags: 3 byte. There is one defined flag: Self reference (It indicates that the media’s data is in the same file as the movie atom.)
   5. Data: The data reference information.
7. Sample table atom: Quicktime stores media data in samples. A sample is a single element in a sequence of time-ordered data. Samples are stored in the media and they have varying durations. Samples are stored in a series of chunks in a media. Chunks are a collection of data samples in a media that allow optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the individual samples within a chunk may have different sizes from one another.



One way to describe a sample is using a sample table atom. It allows the media handler to parse the samples in proper order. The sample table atom contains information for converting from media time to sample number to sample location. This atom also indicates how to interpret the sample (for example whether to decompress the video data, and if so, how). The sample table atom has an atom type of ‘stbl’. If the track that contains the sample table atom references no data, then the sample table atom does not need to contain any child atoms (not a very useful media track). If the track does reference data, then the following child items are required: sample description, sample size, sample to chunk and chunk offset. All of the subtables of the sample table use the same total sample count. A sample description atom is required because it contains a data reference index field which indicates which data reference atom to use to retrieve the media samples. Without the sample description, it is not possible to determine where the media samples are stored. The sync sample atom is optional. If the sync sample atom is not present all the samples are implicitly sync samples.

Sample description atom stores information that allows you to decode samples in the media. The data stored in the sample description varies, depending on the media type. For example, in case of video media, the sample descriptions are image description structures. Below is the layout of sample description atom:



1. Size
2. Type: must be set to ‘stsd’.
3. Version: 1 byte
4. Flags: 3 byte. Set this flag to 0.
5. Number of entries: A 32-bit integer containing the number of sample descriptions that follow.
6. Sample description table: An array of sample descriptions. While the exact format of sample description varies by media type, the first four fields are the same.
   1. Size: 4byte integer
   2. Data Format: A 4 byte integer indicating the format of stored data. This depends on the media type but is usually the compression format or the media type. (mp4a, mp4v, avc1)
   3. Reserved: 6 bytes. Must be set to 0.
   4. Data reference index: A 2 byte integer that contains the index of the data reference to use to retrieve data associated with the samples that use this sample description.

These four fields may be followed by additional data specific to the media type and data format.

Video Sample Description: The data format field for video sample description indicates the type of compression that was used to compress the image data, or the color space representation of uncompressed video data. Some image compression format are cvid, jpeg, ‘smc ‘, mp4v (MPEG-4 video), avc1 (H.264 video). The video media sample description adds the following fields to the general sample description:

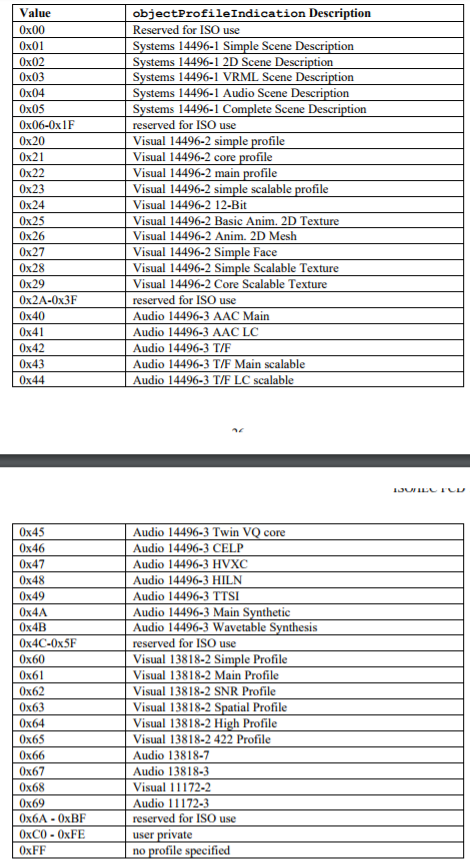
1. Version: 2 byte integer version of the compressed data. This should be set to 0, unless a compressor has changed it’s data format.
2. Revision level: 2 byte integer that must be set to 0.
3. Vendor: A 4 byte integer that specifies the developer of the compressor that generated the compressed data. Can be set to 0.
4. Temporal Quality: A 4 byte integer containing values from 0 to 1023 indicating the degree of temporal compression.
5. Spatial quality: A 4 byte integer containing values from 0 to 1023 indicating the degree of spatial compression.
6. Width: A 2 byte integer that specifies the width of the source image in pixels.
7. Height: A 2 byte integer that specifies the height of the source image in pixels.
8. Horizontal resolution: A 4 byte fixed point number containing the horizontal resolution of the image in pixels per inch.
9. Vertical resolution: A 4 byte fixed point number containing the vertical resolution of the image in pixels per inch.
10. Data size: A 4 byte integer that must be set to 0.
11. Frame count: A 2 byte integer that indicates how many frames of compressed data are stored in each sample. Usually set to 1.
12. Compressor name: A 4 byte Pascal string containing the name of the compressor that created the image, such as ‘jpeg’
13. Depth: A 2 byte integer that indicates the pixel depth of the compressed image. Values of 1, 2, 4, 8, 16, 24 and 32 indicate the depth of color images. Values of 34, 36 and 40 indicate 2-, 4- and 8-bit gray scale, for grayscale images.
14. Color table id: A 2 bit integer that specifies which color table to use. If this field is set to -1, the default color table should be used for the specified depth. For all depths below 16 bits per pixel, this indicates a standard machintosh color table for the specified depth. Depth of 16, 24 and 32 have no color table. If the color table id is set to 0, a color table is contained within the sample description itself. The color table immediately follows the color table id field in the sample description.

Sound Sample Description: Sound media is used to store compressed and uncompressed audio data in quicktime movies. Sound sample description contains information that defines how to interpret sound media. The data format field specifies the format of the audio data. It can be ‘.mp3’, ‘mp4a’ (MPEG-4 Advanced audio encoding), ‘ac-3’, ‘raw ’ etc. The sound sample description (version 0) contains following fields:

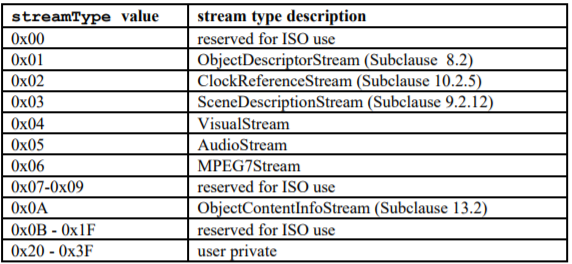
1. Version: 2 byte integer version (currently 0 or 1)
2. Revision level: 2 byte integer that must be set to 0.
3. Vendor: A 4 byte integer that must be set to 0.
4. Number of channels: A 2 byte integer that indicates the number of sound channels used by the sound sample. Set to 1 for monoaural sound, 2 for stereo sounds.
5. Sample size (bits): A 2 byte integer that specifies the number of bits in each uncompressed sound sample. Allowable values are 8 or 16. Formats using more than 16 bits per sample set this field to 16 and use sound description version 1.
6. Compression id: The 2 byte integer that must be set to 0 for version 0. This may be set to -2 for some version 1 sound descriptions.
7. Packet Size: A 2 byte integer that must be set to 0.
8. Sample rate: A 4 byte unsigned fixed point number (16.16) that indicates the rate at which the sound sample was obtained. The integer portion of this value must match the media’s time scale. Most files have integer values such as 44100. Version 0 of the sound description assumes uncompressed audio in ‘raw’ or ‘twos’ format, 1 or 2 channels, 8 or 16 bits per sample, and a compression id of 0.

Sound Sample Description Extensions: All extensions to the SoundDescription record are made using atoms. That means that one or more atoms can be appended to the SoundDescription record. Extensions were first added with sound sample description v1.

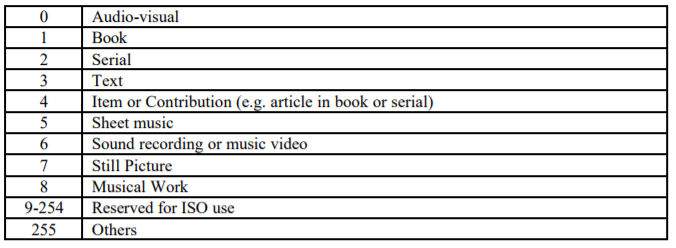
1. MPEG-4 Elementary stream descriptor atom (‘esds’): This atom is a required extension to the sound sample description for MPEG-4 audio. It contains following fields:
   1. Size
   2. Type: set to ‘esds’
   3. Version: 4 byte field set to 0.
   4. Elementary stream descriptor (keep is same as any file for lync): It includes information about the source of the stream data, in form of a unique identifier (the elementary stream id) or a URL pointing to a remote source for the stream. ES Id’s are resolved to particular delivery channels at the TransMux layer. ES descriptor also includes information about the encoding format, configuration information for the decoding process and the sync layer packetization, as well as quality of the service requirements for the transmission of the stream and intellectual property identification. Dependencies between streams can also be signalled. It consists of following fields:
      1. Length: 1 byte, Length of the remainder of this descriptor, excluding trailing embedded descriptor
      2. ES\_ID: 2 byte (Set to 80 80)
      3. Flags: 1 byte- Stream dependence flag(1 bit), URL flag (1 bit), Reserved (1 bit), Stream priority (5 bit)
      4. Depends\_On\_ES\_ID (Optional): 2 byte, present only if stream dependence flag is set.
      5. URLString (Optional): 1 byte
      6. ExtensionDescriptor[]
         1. Length: 1byte/2byte/4byte. Length of the remainder of the descriptor in bytes.
         2. DescriptionData[Length]: 1 byte. Data bytes
      7. LanguageDescriptor[] (0 or 1):
         1. Length: 1 byte
         2. LanguageCode: 3 byte
      8. DecoderConfigDescriptor: It specifies the information about the decoder type and the required decoder resources needed for the associated elementary stream. This is needed at the receiver to determine whether it is able to decode the elementary stream. A stream type identifies the category of the stream while the optional decoder specific information descriptor contains stream specific information for the setup of the decoder in a stream specific format that is opaque to this layer.
         1. Length: 1 byte, length of the remainder of this descriptor in bytes excluding trailing embedded descriptors.
         2. ObjectProfileIndication: 1 byte. An indication of th object profile, or scene description profile, if streamType=sceneDescriptionStream, that needs to be supported by the decoder for this elementary stream. Use one of the below values.



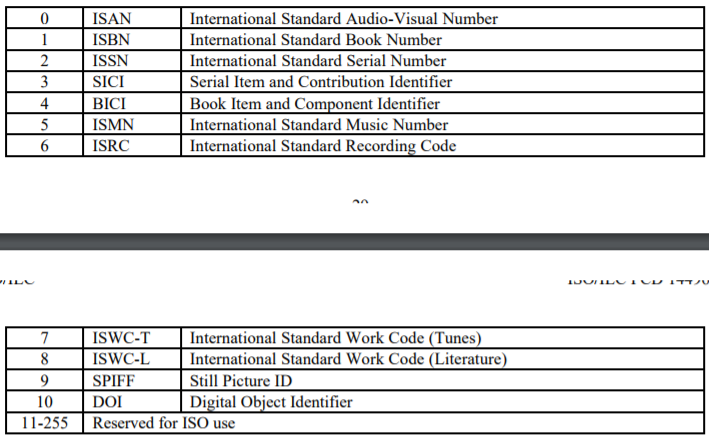
* + - 1. Flags: 1 byte – StreamType (6 bit, conveys the type of the elementary stream as per below table), UpStream (1 bit, indicates that this stream is used for upstream information), Reserved (1 bit, set to 1)



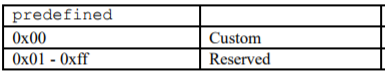
* + - 1. BufferSizeDb: 3 bytes. It is the size of the decoding buffer for this elementary stream in byte.
      2. MaxBitRate: 4 byte. The maximum bitrate of this elementary stream in any time window of one second duration.
      3. Average Bit rate: 4 byte. The average bitrate of this elementary stream. For streams with variable bitrate this value shall be set to 0.
      4. DecoderSpecificInfo[]:
         1. Length: 1 byte or 4 byte.
         2. SpecificInfo[length]
    1. SLConfigDescriptor:
       1. Length: 1 byte. Length of the remainder of this descriptor in bytes
       2. Compatibility: 2bit. Must be set to 0
       3. ContentTypeFlag: 1bit. Flag to indicate if a definition of the type of content is available.
       4. ContentIdentifierFlag: 1 bit. Flag to indicate presence of creation id.
       5. SupplementaryContentIdentifierCount: 1 byte. Since not all works follow a numbered identification scheme, non-standard schemes can be used. It indicates how many of these supplementary fields are following.
       6. ContentType (Optional): 1 byte. Present if ContentTypeFlag is set. Can be one of the following values:



* + - 1. ContentIdentifierType (Optional): 1byte. Defines the type of content identifier using one of below values:

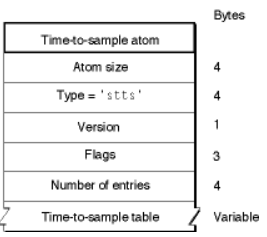


* + - 1. ContentIdentifierLength (Optional): 1 byte. Since the length of each of these identifiers can vary, a length identifier is needed to give the length in bytes.
      2. ContentIdentifier[contentIdentifierLength] (Optional): 1 byte
      3. LanguageCode: 3 byte. Three character iso code for language of the following text field.
      4. SupplementaryContentIdentifierTitleLength (Optional): 1 byte
      5. SupplementaryContentIdentifierTitle[] (Optional): 1 byte
      6. SupplementaryContentIdentifierValueLength (Optional): 1 byte
      7. SupplementaryContentIdentifierValue[] (Optional): 1 byte
    1. IPI\_DescPointer[] (0 or 1)
       1. Length: 1 byte. Length of remainder of this descriptor in bytes.
       2. IPI\_ES\_ID: The ES\_ID of the elementary stream that contains the IP information valid for this elementary stream.
    2. IP\_IdentificationDataSet[] (0 or 1)
    3. QoS\_Descriptor[] (0 or 1)
       1. Length: 1 byte. The length of the remainder of this descriptor in bytes.
       2. Predefined: 1 byte. A value different from zero indicates a predefined QoS profile according to table below:



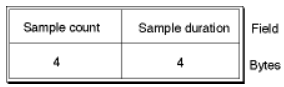
* + - 1. QoS\_QualifierCount (Optional): 1 byte. Present if Predefined is set to 0.
      2. QoS\_Qualifier\_Tag[] (Optional): 1 byte
      3. QoS\_Qualifier\_Length[] (Optional): 1 byte
      4. QoS\_Qualifier\_Data[] (Optional): 1 byte

1. Time to Sample Atoms: Time to sample atoms store duration information for a media’s samples, providing a mapping from a time in a media to the corresponding data sample. The time to sample atom has an atom type of ‘stts’. The atom contains a compact version of a table that allows indexing from time to sample number. Other tables provide sample size and pointers from the sample number. Each entry in the table gives the number of consecutive samples with the same time delta, and the delta of those samples. By adding the deltas, a complete time-to-sample map can be built. The atom contains deltas: DT(n+1) = DT(n) + STTS(n) where STTS(n) is the uncompressed table entry for sample n and DT is the display time for sample n. The sample entries are ordered by timestamps, therefore the deltas are all non-negative. The DT axis has zero origin; DT(i) = SUM (for j = 0 to i-1 or delta(j)), and the sum of all the deltas gives the length of the media in the track (not mapped to the overall time scale, and not considering any edit list). The edit list provides the initial DT value if it is non-empty (non-zero).



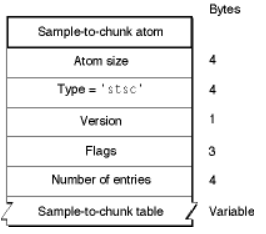
Below are the fields:

1. Size: 4 byte
2. Type: set to ‘stts’
3. Version: 1 byte
4. Flags: 3 byte. Set to 0
5. Number of entries: 4 byte integer containing the count of entries in the time to sample table.
6. Time-to-sample table: A table that defines the duration of each sample in the table. Each table entry contains a count field and a duration field.



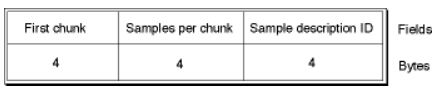
Sample count specifies the number of consecutive samples that have the same duration. Sample duration specifies the duration of each sample. For example, if a video media has a constant frame rate, this table would have one entry and the count would be equal to the number of samples.

1. Sample to Chunk Atom: As samples are added to the media, they are collected into chunks that allow optimized data access. A chunk contains one or more samples. Chunks in a media may have different sizes and the samples in the chunks may have different sizes. This atom stores chunk information for samples in a media. It has an atom type of ‘stsc’

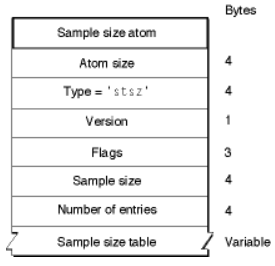


This atom contains following fields:

1. Size: 4 byte
2. Type: 4 byte. Must be set to ‘stsc’
3. Version: 1 byte
4. Flags: 3 byte. Set it to 0.
5. Number of entries: 4 byte. Specifies the count of entries in the sample to chunk table.
6. Sample to chunk table: Each entry in the table contains a first chunk field (the first chunk number using this table entry), a samples per chunk field (the number of samples in each chunk) and a sample description id fields (an identification number associated with the sample description for the sample.

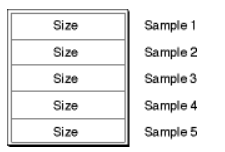


1. Sample Size atoms: You use sample size atoms to specify the size of each sample in the media. It has an atom type of ‘stsz’. It contains the sample count and a table giving the size of each sample.

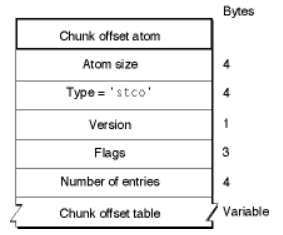


The following fields are present:

1. Size: 4 byte
2. Type: 4 byte. Must be set to stsz
3. Version: 1 byte
4. Flags: 3 byte. Set to 0.
5. Sample size: A 4 byte integer specifying the sample size. If all the samples are of the same size, this field contains the sample value. If it is set to 0 then the samples have different sizes, and those sizes are stored in the sample size table.
6. Number of entries: A 4 byte integer containing the number of entries in the sample size table.
7. Sample size table: A table containing the sample size information. It contains an entry for every sample in the media’s data stream. The table is indexed by sample number – the first entry corresponds to the first sample, the second entry for the second sample.

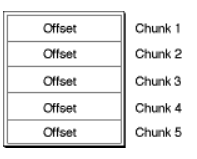


1. Chunk Offset Atom: This identifies the location of each chunk of data in the media’s data stream. It has an atom type of ‘stco’. The chunk offset table gives the index of each chunk into the containing file. There are two variants, permitting use of 4 byte or 8 byte offset. Note that offset are file offset and not offset in any atom in the file (for example mdat atom). The sample table atom can contain a 64-bit chunk offset atom. (STChunkOffset64AID = ‘co64’). When such an atom appears, it is used in place of the original chunk offset atom, which contains only 32 bit offsets.

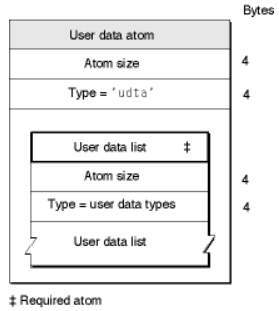


Below is the structure:

1. Size: 4 byte
2. Type: 4 byte. Set to ‘stco’
3. Version: 1 byte
4. Flags: 3 byte. Set to 0
5. Number of entries: A 4 byte integer containing count of entries in the chunk offset table.
6. Chunk offset table: It contains an array of offset values. The offset contains the offset from the beginning of the data stream to the chunk. The table is indexed by chunk number- the first entry corresponds to the first chunk.

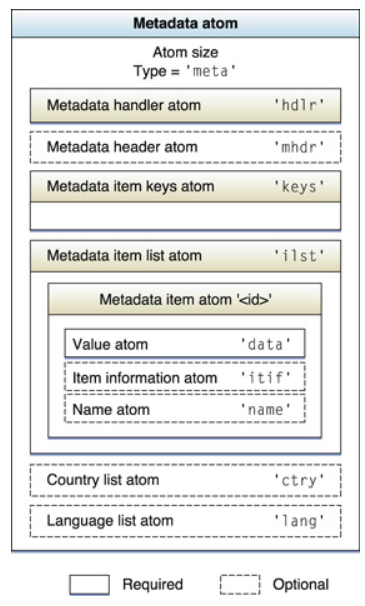


1. User Data Atoms: User data atoms allow you to define and store data associated with a quicktime object (moov, trak or mdia). Like copyright information or whether the movie should loop, and arbitrary information provided by and your application that quicktime simply ignores. A user data atom whose immediate parent is a movie contains data relevant to the movie as a whole. Same goes to track. A quicktime movie file may contain many user data atoms but only one user data atom is allowed as immediate child of any movie or track atom. It has an atom type of ‘udta’. Inside the user data atom, there is a list of atoms describing each piece of user data. You may create new data atom types that your own application recognizes.



Below is a structure:

1. Size: 4 byte
2. Type: 4 byte. Set to ‘udta’
3. User Data list: A user data list is formatted as a series of atoms. Each data element in the user data list contains the size and type information along with it’s payload data. For historical reasons the data list is terminated by a 4 byte integer set to 0.
4. Metadata atom: It is the container for carrying metadata. It has an atom type of ‘meta’. It must contain the following subatoms: metadata handler atom (‘hdlr’), metadata item keys atom (‘keys’), and metadata item list atom (‘ilist’)

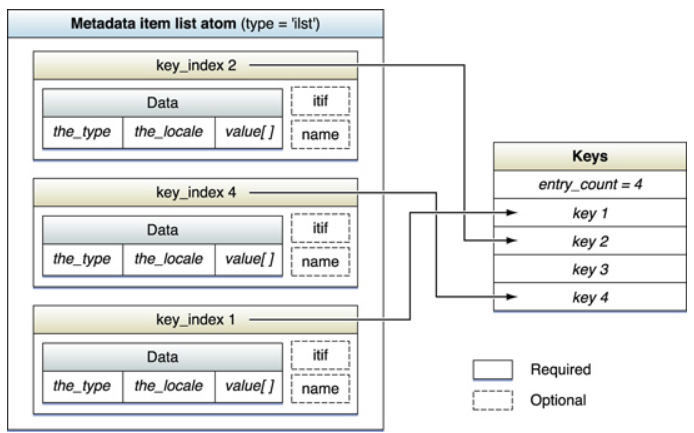


Below is structure of metadata atom:

1. Size: 4 byte
2. Type: 4 byte. Set to ‘meta’
3. Reserved: 4 byte. Set to 0.
4. Following atoms.

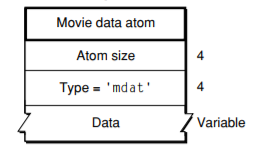
Below is the structure for metdata handler atom:

1. Size: 4 byte
2. Type: 4 byte. Set to ‘hdlr’
3. Version: 1 byte. Set to 0.
4. Flags: 3 byte. Set to 0
5. Predefined: 4 byte. Set to 0.
6. Handler type: A 4 byte integer that indicates the structure used in a metadata atom, set to ‘mdta’.
7. Reserved: An array of 3 const unsigned 4 byte integers. Set to 0.
8. Name: This is a null terminated string in UTF-8 characters which gives a human readable name for the metadata type, for debugging and inspection purposes. It may be empty or a single byte containing 0.
9. Metadata Item list atom: The metadata item list atom holds a list of actual metadata values that are present in the metadata atom. The metadata items are formatted as a list of items. It has a type of ‘ilst’ and contains a number of metadata items, each of which is an atom.



Each item in the metadata item list atom is identified by it’s key. The atom type for each metadata item atom should be set equal to the index of the key for the metadata within the item atom, taking this index from the metadata item keys atom.

1. Movie data atom: It consist of below structure:



1. Size: 4 byte
2. Type: ‘mdat’
3. Data