Adobe Flash Video File Format Specification
Version 10.1

Adobe Flash Video File Format Specification Version 10.1

Updated Information/Additional Third Party Code Information available at http://www.adobe.com/go/thirdparty.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Published August 2010

Edit 47 2010-09-03

Contents

Introduction

Flash® is the de facto standard for dynamic media on the Web, supporting a number of media formats, including two core container formats for delivering synchronized audio and video streams:

- F4V, for H.264/AAC–based content, and
- FLV, for other supported codecs such as Sorensen Spark and On2 VP6.

This document provides the technical format information for the F4V and FLV video file formats supported by Adobe® products.

Adobe seriously considers all feedback to the specification of the video file format. E-mail any unclear or potentially erroneous information within the specification to Adobe at flashformat@adobe.com. All such email submissions shall be subject to the Submitted Materials guidelines in the Terms of Use at www.adobe.com/misc/copyright.html.

The F4V Video File Format

The open specification of the F4V video file format builds on the standard IEC 14496-12 (MPEG-4 Part 12) ISO base media file format. It has a flexible structure and defines specific supported codecs and extensions. The F4V video file format thus simplifies the implementation of dynamic media software, facilitating interoperability across tools, services, and clients.

Starting with Flash Player 9 Update 3 (9,0,115,0), Flash Player can play F4V files.

For details on the F4V video file format, see Section 1. The F4V File Format

For the use of metadata in F4V files, see Section 3. F4V Metadata.

New in the F4V Video File Format

The release of Flash Player 10,1,53,64 added support of the following features and boxes in the Flash F4V video file format. Boxes in *italic* are defined by Adobe Systems.

New Features Cue points, Encryption, Hinting, HTTP streaming

New Boxes

abst	*adaf*	*adkm*	*aeib*	*afra*	*afrt*	*ahdr*	*akey*	*amhp*	*amto*	*aprm*	*aps*
asig	*asrt*	dinf	dref	edit	elst	enca	encr	encv	*flxs*	frma	hdlr
hmdh	mehd	mfhd	mfra	mfro	moof	mvex	nmhd	*rtmp*	schi	schm	sdtp
sinf	smhd	tfhd	tfra	traf	trex	trun	url	vmhd			

The FLV Video File Format

An FLV file encodes synchronized audio and video streams. The audio and video data within FLV files are encoded in the same way as audio and video within SWF files.

This document describes FLV version 1. See Annex E. The FLV File Format

1 The F4V File Format

1.1 Overview

Flash Player 9 Update 3 (9,0,115,0) and higher can play F4V files. The F4V format is based on the ISO/IEC 14496-12:2008 ISO base media file format.

A large part of what distinguishes the F4V format from the ISO base media file format involves the metadata formats that F4V can store. This chapter discusses all aspects of the F4V format except metadata, which is covered in Section 3 F4V Metadata.

1.2 Simple data types

This following data types are used in F4V files.

Type	Definition
0x…	Hexadecimal value …
4CC	Four-character ASCII code, such as 'moov', encoded as UI32
SI8	Signed 8-bit integer
SI8.8	Signed 16-bit fixed point number having 8 fractional bits
SI16	Signed 16-bit integer
SI16.16	Signed 32-bit fixed point number having 16 fractional bits
SI24	Signed 24-bit integer
SI32	Signed 32-bit integer
SI64	Signed 64-bit integer
STRING	Sequence of Unicode 8-bit characters (UTF-8), terminated with 0x00 (unless otherwise specified)
UI8	Unsigned 8-bit integer
UI16	Unsigned 16-bit integer
UI16.16	Unsigned 32-bit fixed point number having 16 fractional bits
UI24	Unsigned 24-bit integer
UI32	Unsigned 32-bit integer
UI64	Unsigned 64-bit integer
UIn	Bit field with unsigned n-bit integer, where n is in the range 1 to 31, excluding 8, 16, 24
xxx [ ]	Array of type xxx. Number of elements to be inferred, for example from box size.
xxx [n]	Array of n elements of type xxx

Multi-byte integers shall be stored in big-endian byte order, in contrast with SWF, which uses little-endian byte order. For example, as a UI16 in SWF file format, the byte sequence that represents the number 300 (0x12C) is 0x2C 0x01; as a UI16 in F4V file format, the byte sequence that represents the number 300 is 0x01 0x2C.

1.3 F4V box format

The fundamental building block of an F4V file is a box that has the following BOX format:

F4V box

Field	Type	Comment
Header	BOXHEADER	A consistent header that all boxes have
Payload	UI8 [ ]	A number of bytes, the length of which is defined by the BOXHEADER

Each box structure begins with a BOXHEADER structure:

BOXHEADER

Field	Type	Comment
TotalSize	UI32	The total size of the box in bytes, including this header. 0 indicates that the box extends until the end of the file.
BoxType	UI32	The type of the box, usually as 4CC
ExtendedSize	IF TotalSize == 1 UI64	The total 64-bit length of the box in bytes, including this header

Many boxes are well under 4 gigabytes in length and can store their size in the TotalSize field. The format also supports very large boxes by setting the 32-bit TotalSize field to 1 and storing a 64-bit size in ExtendedSize.

Each box is identified with a 32-bit type. For most boxes, this 32-bit type doubles as a human-readable four-character ASCII code or 4CC, such as 'moov' (0x6D6F6F76) and 'mdat' (0x6D646174).

The box payload immediately follows the box header. The size of the payload in bytes is equal to the total size of the box minus either 8 bytes or 16 bytes, depending on the size of the header.

For more information, see section 4.2 of ISO/IEC 14496-12:2008.

1.4 F4V Box Hierarchy

Table 1 shows boxes that the Flash Player recognizes in an F4V file, their nesting, required presence, and recommended order. The hierarchy within the Sample Description box is shown separately in Table 2.

Boxes in *italic* are defined by Adobe Systems. The other boxes are specified in ISO/IEC 14496-12:2008.

Some boxes contain additional boxes not listed here. Consult the box definitions for further details.

Table 1. The F4V Box Hierarchy

Box Type	Required?	Short Description
ftyp	Y	File type and compatibility
pdin	N	Progressive download information
afra	*See HTTP streaming*	*Fragment random access for HTTP streaming*
abst	*See HTTP streaming*	*Bootstrap info for HTTP streaming*
asrt	*Y*	*Map fragment to segment*
afrt	*Y*	*Map time to fragment*
moov	Y	Container for structural metadata
mvhd	Y	Movie header, overall declarations
trak	Y	Container for an individual track
tkhd	Y	Track header, main properties of a track
edts	N	Edit list container
elst	N	Time-line mapping
mdia	Y	Container for media track properties
mdhd	Y	Media track properties
hdlr	Y	Handler, declares the media type
minf	Y	Media information container
vmhd	Y, one of these according to media type	Video media header
smhd		Sound media header
hmhd		Hint media header
nmhd		Null media header
dinf	Y	Data information container
dref	Y	Data reference
url	Y	URL reference
stbl	Y	Container for sample properties
stsd	Y	Sample descriptions (codec types etc.)
stts	Y	Map decoding time to sample
ctts	N	Map composition time to sample
stsc	Y	Map sample to chunk

			stsz	N	Sample sizes
			stco co64	Y, one of stco or co64	Chunk offsets
			stss	N	Sync sample table
			sdtp	N	Independent and disposable samples
	mvex			N	Movie extends
		mehd		N	Movie extends header
		trex		Y	Track extends defaults
	auth			*N*	*Author metadata tag*
	titl			*N*	*Title metadata tag*
	dscp			*N*	*Description metadata tag*
	cprt			*N*	*Copyright metadata tag*
	udta			N	User data
uuid				*N*	*XMP Metadata*
moof				N	Movie fragment
	mfhd			Y	Movie fragment header
	traf			N	Track fragment
		tfhd		Y	Track fragment header
		trun		N	Track fragment run
mdat				Y for other than HTTP streaming	Media data container
meta				N	Container for metadata boxes
	ilst			*N*	*Metadata tags*
free				N	Free space
skip				N	Free space
mfra				N	Movie fragment random access
	tfra			N	Track fragment random access
	mfro			Y	Movie fragment random access offset

1.5 Sample Description Box Hierarchy

Table 2 shows the hierarchy within the Sample Description box.

Table 2. The box hierarchy in the Sample Description box

Box Name	Required?	Short Description
stsd	Y	Sample descriptions
Mediatype-specific sample entry boxes	Y for other than encryption	Sample description for this track
rtmp	*Y for HTTP streaming*	*Adobe Mux Hint Sample Entry*
amhp	*Y*	*Adobe Mux Hint Process*
amto	*N*	*Adobe Mux Time Offset*
encv enca encr	Y for encryption	Sample descriptions entry for encrypted tracks
sinf	Y	Protection Scheme Information
frma	Y	Original Format
schm	Y	Scheme Type
schi	Y	Scheme Information
adkm	*Y*	*Adobe's DRM Key Management System*
ahdr	*Y*	*Adobe DRM Header*
aprm	*Y*	*Standard Encryption Params*
aeib	*Y*	*Encryption Information*
akey	*Y*	*Key Information*
aps	*N*	*FMRMS v1.x Params*
flxs	*N*	*Flash Access v 2.0 Params*
asig	*N*	*Adobe Signature*
adaf	*Y*	*Adobe DRM Access Unit Format*

1.6 Handling Unsupported Boxes

The ISO specification ISO/IEC 14496-12:2008 and the Apple QuickTime specification define additional box types, not included in this specification. These box types are not part of the F4V file format, and F4V players need not support them. The F4V player shall disregard unsupported boxes and their contents and continue playing the file.

1.7 Ordering of Boxes

For best player performance, the required top-level boxes in an F4V file should be in the following order:

1. File Type (ftyp),
2. Movie (moov),
3. Media Data (mdat).

The ftyp box shall be before the moov and mdat boxes, and any other "significant" variable-size boxes.

The ftyp box should be the first box in the file, or as early as possible in the file.

While the Flash Player can play both orderings of the moov and mdat boxes, the moov box should always be before the mdat box, as this provides for faster startup and progressive streaming.

Many F4V file creation tools place boxes in suboptimal order for playing, in which case, a post-processing step should be applied to place the boxes in the recommended order.

For required boxes and box order for HTTP streaming support, see Annex C. HTTP Streaming: File Structure.

1.8 Supported Media Types

The following tables describe the media types that can be encapsulated inside an F4V file.

1.8.1 Supported audio types

Media type	Comments
MP3	A media type of .mp3 (0x2E6D7033) indicates that the track contains MP3 audio data. The dot character, hex 0x2E, is included to make a complete four-character code.
AAC	A media type of mp4a (0x6D703461) indicates that the track is encoded with AAC audio. Flash Player supports the following AAC profiles, denoted by their object types: - 1 = main profile - 2 = low complexity, a.k.a. LC - 5 = high efficiency/scale band replication, a.k.a. HE/SBR When the audio codec is AAC, an esds box occurs inside the stsd box of a sample table. This box contains initialization data that an AAC decoder requires to decode the stream. See ISO/IEC 14496-3 for more information about the structure of this box.

1.8.2 Supported video types

Media type	Comments
GIF	A media type of gif (0x67696620) denotes a still frame of video data compressed using the CompuServe GIF format. The space character, hex 0x20, is included to make a complete four-character code.
PNG	A media type of png (0x706E6720) denotes a still frame of video data compressed using the standard PNG format. The space character, hex 0x20, is included to make a complete four-character code.
JPEG	A media type of jpeg (0x6A706567) denotes a still frame of video data compressed using the standard JPEG format.
H.264	A media type of H264 (0x48323634), h264 (0x68323634), or avc1 (0x61766331) indicates that the track is encoded with H.264 video. Flash Player supports the following H.264 video profiles: - 0 = supported for older media that neglects to set profile - 66 = baseline - 77 = extended - 88 = main - 100 = YUV 4:2:0, 8 bits/sample, a.k.a. "High" - 110 = YUV 4:2:0, 10 bits/sample, a.k.a. "High 10" - 122 = YUV 4:2:2, 10 bits/sample, a.k.a. "High 4:2:2" - 144 = YUV 4:4:4, 12 bits/sample, a.k.a. "High 4:4:4" When the video codec is H.264, an avcC box occurs inside the stsd box of a sample table. This box contains initialization data that an H.264 decoder requires to decode the stream. Bytes 1 and 3 after the BOXHEADER contain the profile and level, respectively, for the AVC data. For more information about the remainder of the avcC box, see section 5.3.4.1 of ISO/IEC 14496-15.
VP6	The following media types indicate that the track is encoded with On2 VP6 video. - VP6F (0x56503646) - VP6A (0x56503641) - VP60 (0x56503630) - VP61 (0x56503631) - VP62 (0x56503632)

1.8.3 Supported data types

Media type	Comments
Text	A media type of either text (0x74657874) or tx3g (0x74783367) indicates that the track contains textual data that is made available via ActionScript.
AMF0	A media type of amf0 (0x616D6630) indicates that the track contains data corresponding to the original version of the ActionScript Message Format (AMF).
AMF3	A media type of amf3 (0x616D6633) indicates that the track contains data corresponding to the ActionScript Message Format (AMF) version 3.

2 F4V Box Definitions

This section defines the boxes supported by the F4V file format.

2.1 File Type box

Box type: 'ftyp'

Container: File

Mandatory: Yes

Quantity: One

The F4V format is based on the ISO MPEG4 format, which in turn is based on the Apple QuickTime container format. The subsets of the format support different features. The File Type (ftyp) box helps identify the features that a program needs to support to play a particular file.

The ftyp box should be placed as early as possible, and shall be before any variable length box.

Flash Player does not enforce any restrictions with respect to ftyp boxes. If the file contains data that Flash Player can decode, Flash Player tries to play it.

ftyp box

Field	Type	Comments
Header	BOXHEADER	BoxType = 'ftyp' (0x66747970)
MajorBrand	UI32	The primary brand identifier. For an F4V file, MajorBrand is 'f4v' (0x66347620).
MinorVersion	UI32	MinorVersion is informative only. It shall not be used to determine the conformance of a file to a standard. It may allow for more precise identification of the major brand for inspection, debugging, or improved decoding.
CompatibleBrands	UI32 [ ]	Arbitrary number of compatible brands, until the end of the box

For more information, see section 4.3 of ISO/IEC 14496-12:2008.

2.2 Progressive Download Information box

Box type: 'pdin'

Container: File

Mandatory: No

Quantity: One

The Progressive Download Information (pdin) box defines information about progressive download. The payload of a pdin box provides hints about how much data to download before a player can safely begin playback.

The pdin box should be placed as early as possible in the file, after the File Type (ftyp) box, for maximum utility.

pdin box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'pdin' (0x7064696E)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
RateDelay	RATEDELAY [ ]	Arbitrary number of RATEDELAY records, until the end of the box

Each RATEDELAY record has the following layout:

RATEDELAY

Field	Type	Comment
Rate	UI32	The rate (in bytes/second) to be considered for this record
InitialDelay	UI32	The number of milliseconds to delay before beginning playback at this rate

For more information, see section 8.1.3 of ISO/IEC 14496-12:2008.

2.3 Movie box

Box type: 'moov'

Container: File

Mandatory: Yes

Quantity: One

The Movie (moov) box is effectively the "header" of an F4V file. The moov box itself contains one or more other boxes, which in turn contain other boxes, which define the structure of the F4V data.

moov box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'moov' (0x6D6F6F76)
Boxes	BOX [ ]	Arbitrary number of boxes that define the file structure

For more information, see section 8.2.1 of ISO/IEC 14496-12:2008.

2.4 Movie Header box

Box type: 'mvhd'

Container: Movie box ('moov')

Mandatory: Yes

Quantity: One

The Movie Header (mvhd) box defines playback information that applies to the entire F4V file. The mvhd box should be placed first in its container.

mvhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mvhd' (0x6D766864)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
CreationTime	IF Version == 0 UI32 IF Version == 1 UI64	The creation time of the F4V file, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)
ModificationTime	IF Version == 0 UI32 IF Version == 1 UI64	The last modification time of the F4V file, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)

TimeScale	UI32	The time coordinate system for the entire F4V file, in number of time units per second. For example, 100 indicates the time units are 1/100 second each.
Duration	IF Version == 0 UI32 IF Version == 1 UI64	The total length of the F4V file presentation, in TimeScale units. This is also the duration of the longest track in the file.
Rate	SI16.16	The preferred rate of playback, expressed as a fixed point 16.16 number (commonly 0x00010000 = 1.0, or normal playback rate)
Volume	SI8.8	The master volume of the file, expressed as a fixed point 8.8 number (commonly 0x0100 = 1.0, or full volume)
Reserved	UI16	Reserved. Set to 0
Reserved	UI32 [2]	Reserved. Set to 0
Matrix	SI32 [9]	Transformation matrix for the F4V file, shall be {0x00010000, 0, 0, 0, 0x00010000, 0, 0, 0, 0x40000000}
Reserved	UI32 [6]	Reserved. Set to 0
NextTrackID	UI32	The ID of the next track to be added to the presentation. This value shall not be 0 but may be all 1's to indicate an undefined state

For more information, see section 8.2.2 of ISO/IEC 14496-12:2008.

2.5 Track box

Box type: 'trak'

Container: Movie box ('moov')

Mandatory: Yes

Quantity: One or more

Each Track (trak) box corresponds to an individual media track within the F4V file and contains boxes that further define the properties of the media track.

trak box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'trak' (0x7472616B)
Boxes	BOX [ ]	Arbitrary number of boxes that define the media track

For more information, see section 8.3.1 of ISO/IEC 14496-12:2008.

2.5.1 Track Header box

Box type: 'tkhd'

Container: Track box ('trak')

Mandatory: Yes

Quantity: One

The Track Header (tkhd) box describes the main properties of a track. The tkhd box should be placed first in its container.

tkhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'tkhd' (0x746B6864)
Version	UI8	Either 0 or 1
Flags	UI24	Bit 0 = the track is enabled Bit 1 = the track is part of the presentation Bit 2 = the track should be considered when previewing the F4V file
CreationTime	IF Version == 0 UI32 IF Version == 1 UI64	The creation time of the track, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)
ModificationTime	IF Version == 0 UI32 IF Version == 1 UI64	The last modification time of the track, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)
TrackID	UI32	The track's unique identifier
Reserved	UI32	Reserved. Set to 0
Duration	IF Version == 0 UI32 IF Version == 1 UI64	The duration of the track, in TimeScale units defined in the Movie Header box, section 2.4.
Reserved	UI32 [2]	Reserved. Set to 0
Layer	SI16	The position of the front to back ordering of tracks, expected to be 0 for F4V files
AlternateGroup	SI16	0
Volume	SI8.8	0x0100 (fixed point 8.8 number representing 1.0) for audio track, otherwise 0
Reserved	UI16	Reserved. Set to 0
TransformMatrix	SI32[9]	A matrix of fixed point values defining a perspective transform, which shall be {0x00010000, 0, 0 0, 0x00010000, 0 0, 0, 0x40000000}
Width	UI16.16	Width expressed as a fixed point 16.16 number
Height	UI16.16	Height expressed as a fixed point 16.16 number

For more information, see section 8.3.2 of ISO/IEC 14496-12:2008.

2.5.2 Edit box

Box type: 'edts'

Container: Track box ('trak')

Mandatory: No

Quantity: One

The Edit (edts) box maps the presentation timeline to the media timeline, as it is stored in F4V file. The edts box is a container for the edit lists.

If the file does not contain an edts box, there is an implicit one-to-one mapping of these timelines.

The edts box should precede the Media (mdia) box.

edit box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'edts' (0x65647473)
Edit list box	BOX	An explicit time-line map

For more information, see section 8.6.5 of ISO/IEC 14496-12:2008.

2.5.2.1 Edit List box

Box type: 'elst'

Container: Edit box ('edts')

Mandatory: No

Quantity: One

The Edit List (elst) box contains an explicit time line map in the form of edit list entries. Each entry in the elst box defines a part of the presentation timeline through one of the following means:

- By mapping a part of the media timeline
- By indicating an empty time (that is, a gap)
- By defining a dwell (that is, the location at which a single point of time is held for a period)

An empty edit in an edit list box indicates a gap. To specify a starting offset for a track, insert an empty edit at the beginning of the track.

An empty edit shall not be the last edit in a track. If the duration specified in the movie header box is different from the track's actual duration, an implicit empty edit is placed at the end of the track.

The media should have a key frame immediately after the gap. Alternately, Flash Player can use the data in the Sync Sample box to find a key frame after a gap.

elst box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'elst' (0x656C7374)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
EntryCount	UI32	Number of entries in the edit list entry table
EditListEntryTable	ELSTRECORD [EntryCount]	An array of ELSTRECORD structures

Each ELSTRECORD has the following format:

ELSTRECORD		
Field	**Type**	**Comment**
SegmentDuration	IF Version == 0 UI32 IF Version == 1 UI64	Duration of this edit segment, in TimeScale units defined in the Movie Header (moov) box
MediaTime	IF Version == 0 SI32 IF Version == 1 SI64	Starting time within the media of this edit segment as composition time, in TimeScale units defined in the mdhd? box. A value of -1 specifies an empty edit.
MediaRateInteger	SI16	Relative rate at which to play the media of this edit segment. The default value is 1. A value of 0 specifies dwell editing.
MediaRateFraction	SI16	Reserved. Set to 0

For more information, see section 8.6.6 of ISO/IEC 14496-12:2008.

2.6 Media box

Box type: 'mdia'

Container: Track box ('trak')

Mandatory: Yes

Quantity: One

The Media (mdia) box contains boxes that define media track properties.

mdia box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'mdia' (0x6D646961)
Boxes	BOX [ ]	Arbitrary number of boxes that define media track properties

For more information, see section 8.4 of ISO/IEC 14496-12:2008.

2.6.1 Media Header box

Box type: 'mdhd'

Container: Media box ('mdia')

Mandatory: Yes

Quantity: One

The Media Header (mdhd) box describes properties about a media track. The mdhd box should be placed first in its container.

mdhd box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'mdhd' (0x6D646864)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0

CreationTime	IF Version == 0 UI32 IF Version == 1 UI64	The creation time of the box, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)
ModificationTime	IF Version == 0 UI32 IF Version == 1 UI64	The last modification time of the box, expressed as seconds elapsed since midnight, January 1, 1904 (UTC)
TimeScale	UI32	The time coordinate system for this track, in number of time units per second
Duration	IF Version == 0 UI32 IF Version == 1 UI64	The total duration of this track, in TimeScale units
Pad	UI1	Padding, set to 0
Language	UI5 [3]	3-character code specifying language (see ISO 639-2/T), each character interpreted as 0x60 + (5 bit) code to yield an ASCII character
Reserved	UI16	Reserved. Set to 0

For more information, see section 8.4.2 of ISO/IEC 14496-12:2008.

2.6.2 Handler Reference box

Box type: 'hdlr'

Container: Media box ('mdia')

Mandatory: Yes

Quantity: One

A Handler Reference (hdlr) box declares the nature of the media data inside the track. The hdlr box should precede the Media Information (minf) box.

hdlr box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'hdlr' (0x68646C72)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0.
Predefined	UI32	Set to 0.
HandlerType	UI32	An integer containing the following 4CC values: 'vide' = Video track 'soun' = Audio track 'data' = Data track 'hint' = Hint track Other track types are ignored.
Reserved	UI32 [3]	Set to 0.
Name	String	Null terminated UTF-8 string that names the track type, used for debugging purposes.

For more information, see section 8.4.3 of ISO/IEC 14496-12:2008.

2.7 Media Information box

Box type: 'minf'

Container: Media box ('mdia')

Mandatory: Yes

Quantity: One

The Media Information (minf) box contains boxes that define the track's media information.

The minf box contains one Media Header box, the type of which corresponds to the track's HandlerType.

minf box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'minf' (0x6D696E66)
Boxes	BOX [ ]	Arbitrary number of boxes that define the track's media information

For more information, see section 8.4.4 of ISO/IEC 14496-12:2008.

2.7.1 Video Media Header box

Box type: 'vmhd'

Container: Media Information box ('minf')

Mandatory: Yes for a video track, otherwise no.

Quantity: One for a video track, otherwise zero.

The Video Media Header (vmhd) box contains general information for video media, independent of the coding used. The vmhd box should be placed first in its container.

vmhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'vmhd' (0x766D6864)
Version	UI8	Expected to be 0
Flags	UI24	Set to 1.
GraphicsMode	UI16	Composition mode for video track. The default value is 0, which means copy over the existing image.
OpColor	UI16 [3]	Set of 3 RGB color values to be used by graphics modes. Default values: (0, 0, 0)

For more information, see section 8.4.5.2 of ISO/IEC 14496-12:2008.

2.7.2 Sound Media Header box

Box type: 'smhd'

Container: Media Information box ('minf')

Mandatory: Yes for an audio track, otherwise no.

Quantity: One for an audio track, otherwise zero.

The Sound Media Header box contains general information for audio media, independent of the coding used. The smhd box should be placed first in its container.

smhd

Field	Type	Comment
Header	BOXHEADER	BoxType = 'smhd' (0x736D6864)
Version	UI8	Expected to be 0.
Flags	UI24	Set to 0.
Balance	SI8.8	A fixed point 8.8 number. Maps mono audio tracks to the stereo space, as follows: -1.0 = full left 0.0 = center 1.0 = full right
Reserved	UI16	Set to 0.

For more information, see section 8.4.5.3 of ISO/IEC 14496-12:2008.

2.7.3 Hint Media Header box

Box type: 'hmhd'

Container: Media Information box ('minf')

Mandatory: Yes for a hint track, otherwise no.

Quantity: One for a hint track, otherwise zero.

The Hint Media Header (hmhd) box contains the general information for hint tracks, independent of the protocol used. The hmhd box should be placed first in its container.

hmhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'hmhd' (0x686D6864)
Version	UI8	Expected to be 0.
Flags	UI24	Set to 0.
MaxPDUSize	UI16	Size (in bytes) of the largest PDU in hint stream.
AvgPDUSize	UI16	Average size (in bytes) of a PDU in entire presentation
MaxBitRate	UI32	Maximum rate (in bits per second) over an interval of 1 second.
AvgBitRate	UI32	Average rate (in bits per second) over entire presentation.
Reserved	UI32	Set to 0.

For more information, see section 8.4.5.4 of ISO/IEC 14496-12.

2.7.4 Null Media Header box

Box type: 'nmhd'

Container: Media Information box ('minf')

Mandatory: Yes for metadata track, otherwise no.

Quantity: Zero for video, audio, and hint tracks, otherwise, one.

The Null Media Header (nmhd) box contains the general information for tracks other than video and audio. The nmhd box should be placed first in its container.

nmhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'nmhd' (0x6E6D6864)
Version	UI8	Expected to be 0.
Flags	UI24	Reserved. Set to 0

For more information, see section 8.4.5.5 of ISO/IEC 14496-12.

2.7.5 Data Information box

Box type: 'dinf'

Container: Media Information box ('minf')

Mandatory: Yes

Quantity: One

The Data Information (dinf) box contains a Data Reference (dref) box, which declares the location of the media data in a track. The dinf box should precede the Sample Table (stbl) box.

dinf box

Field	Type	Comments
Header	BOXHEADER	BoxType = 'dinf' (0x64696E66)
Data Reference box	BOX	Table of data references, used to locate media data.

For more information, see section 8.7.1 of ISO/IEC 14496-12.

2.7.5.1 Data Reference box

Box type: 'dref'

Container: Data Information box ('dinf')

Mandatory: Yes

Quantity: One

A Data Reference (dref) box contains a table of data references, normally URLs, which declare the location of the media data used within the presentation. The data is in the same file as this box. The data reference index in the Sample Description (stsd) box ties entries in this table to the samples in the track. A track may be split over several sources in this way. If the data is in the same file as this box, then no string, not even an empty one, shall be supplied in the entry field.

The DataEntryBox within the dref box shall be a DataEntryUrlBox.

dref box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'dref' (0x64726566)
Version	UI8	Expected to be 0.

Flags	UI24	Reserved. Set to 0
EntryCount	UI32	Number of entries.
DataEntry	DataEntryBox [EntryCount]	An array of data entry boxes. While ISO 14496-12 permits several box types, the only box type allowed here for F4V is DataEntryURLBox (url).

For more information, see section 8.7.2 of ISO/IEC 14496-12.

The DataEntryBox within the dref box is a DataEntryUrlBox (url) defined as follows.

url box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'url ' (0x75726C20)
Version	UI8	Expected to be 0
Flags	UI24	Set to 1. 1 indicates the media data is in the same file as the Movie box containing this box.

2.7.6 Sample Table box

Box type: 'stbl'

Container: Media Information box ('minf')

Mandatory: Yes

Quantity: One

The Sample Table (stbl) box contains boxes that define properties about the samples that make up a track.

The boxes within the stbl box should be in the following order: Sample Description (stsd), Decoding Time to Sample (stts), Sample to Chunk (stsc), Sample Size (stsz), Chunk Offset (stco or co64).

The Sample Description (stsd) box and its contained boxes are specified in section 2.8 Sample Description Box Structure.

stbl box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stbl' (0x7374626C)
Boxes	BOX [ ]	Arbitrary number of boxes that define properties about the track's constituent samples.

For more information, see section 8.5 of ISO/IEC 14496-12:2008.

2.7.6.1 Decoding Time to Sample box

Box type: 'stts'

Container: Sample Table box ('stbl')

Mandatory: Yes

Quantity: One

The Decoding Time to Sample (stts) box defines the time-to-sample mapping for a sample table.

stts box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stts' (0x73747473)
Version	UI8	Expected to be 0

Flags	UI24	Reserved. Set to 0
Count	UI32	The number of STTSRECORD entries
Entries	STTSRECORD [Count]	An array of STTSRECORD structures

Each STTSRECORD has the following format:

STTSRECORD

Field	Type	Comment
SampleCount	UI32	The number of consecutive samples that this STTSRECORD applies to
SampleDelta	UI32	Sample duration in TimeScale units defined in the mdhd box

For more information, see section 8.6.1.2 of ISO/IEC 14496-12:2008.

2.7.6.2 Composition Time to Sample box

Box type: 'ctts'

Container: Sample Table box ('stbl')

Mandatory: No

Quantity: One

The Composition Time to Sample (ctts) box defines the composition time to sample mapping for a sample table.

ctts box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'ctts' (0x63747473)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
Count	UI32	The number of CTTSRECORD entries
Entries	CTTSRECORD [Count]	An array of CTTSRECORD structures

Each CTTSRECORD has the following structure:

CTTSRECORD

Field	Type	Comment
SampleCount	UI32	The number of consecutive samples that this CTTSRECORD applies to
SampleOffset	UI32	For each sample specified by the SampleCount field, this field contains a positive integer that specifies the composition offset from the decoding time in TimeScale units defined in the mdhd box

Samples are not always composed (presented to the user) at the time of decoding. The ctts box contains offsets from the decoding time to the time when samples are to be presented to the user.

For more information, see section 8.6.1.3 of ISO/IEC 14496-12:2008.

2.7.6.3 Sample to Chunk box

Box type: 'stsc'
Container: Sample Table box ('stbl')
Mandatory: Yes
Quantity: One
The Sample To Chunk (stsc) box defines the sample-to-chunk mapping in the sample table of a media track.

stsc box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stsc' (0x73747363)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
Count	UI32	The number of STSCRECORD entries
Entries	STSCRECORD [Count]	An array of STSCRECORD structures

Each STSCRECORD has the following format:

STSCRECORD

Field	Type	Comment
FirstChunk	UI32	The first chunk that this record applies to
SamplesPerChunk	UI32	The number of consecutive samples that this record applies to
SampleDescIndex	UI32	The sample description that describes this sequence of chunks

For more information, see section 8.7.4 of ISO/IEC 14496-12:2008.

2.7.6.4 Sample Size box

Box type: 'stsz'
Container: Sample Table box ('stbl')
Mandatory: No
Quantity: One
The Sample Size (stsz) box specifies the size of each sample in a sample table.

stsz box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stsz' (0x7374737A)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
ConstantSize	UI32	If all samples have the same size, this field is set with that constant size, otherwise it is 0
SizeCount	UI32	The number of samples in the track
SizeTable	IF ConstantSize == 0 UI32 [SizeCount]	A table of sample sizes. If ConstantSize is not 0, this table is empty

For more information, see section 8.7.3.2 of ISO/IEC 14496-12:2008.

2.7.6.5 Chunk Offset box

Box type: 'stco' or 'co64'

Container: Sample Table box ('stbl')

Mandatory: Yes

Quantity: One

Each Sample Table box shall contain one Chunk Offset box of either the stco or the co64 type. The stco and co64 boxes define chunk offsets for each chunk in a sample table.

stco and co64 boxes

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stco' (0x7374636F) or 'co64' (0x636F3634)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
OffsetCount	UI32	The number of offsets in the Offsets table
Offsets	IF BoxType == 'stco' UI32 [OffsetCount] ELSE IF BoxType == 'co64' UI64 [OffsetCount]	A table of absolute chunk offsets within the file

For more information, see section 8.7.5 of ISO/IEC 14496-12:2008.

2.7.6.6 Sync Sample box

Box type: 'stss'

Container: Sample Table box ('stbl')

Mandatory: No

Quantity: One

The Sync Sample (stss) box specifies which samples within a sample table are sync samples. Sync samples are samples that are safe to seek to. If the track is a video track, sync samples are the keyframes or intraframes that do not rely on any data from any other frames.

If the Sample Table (stbl) box does not contain an stss box, all samples in the track shall be treated as sync samples.

stss box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stss' (0x73747373)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
SyncCount	UI32	The number of entries in SyncTable
SyncTable	UI32 [SyncCount]	A table of sample numbers that are also sync samples, sorted in ascending order of sample numbers

For more information, see section 8.6.2 of ISO/IEC 14496-12:2008.

2.7.6.7 Independent and Disposable Samples box

Box type: 'sdtp'

Container: Sample Table box ('stbl') or Track Fragment 'traf'

Mandatory: No

Quantity: One in each of the stbl and traf boxes

An stbl or traf box may each contain one Independent and Disposable Samples (sdtp) box. The sdtp box helps in implementing features such as fast-forward and random access. The sdtp box states whether a sample is an I-picture, and provides information about frame dependencies and redundant coding that are present in a sample. The number of entries in the table is the same as the value of SampleCount in the Sample Size box.

sdtp box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'sdtp' (0x73647470)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
SampleDependency	SAMPLEDEPENDENCY [ ]	Dependency information for each sample, to the end of the box

Each SAMPLEDEPENDENCY record has the following structure:

SAMPLEDEPENDENCY

Field	Type	Comment
Reserved	UI2	Reserved. Set to 0.
SampleDependsOn	UI2	0 = the sample dependency is unknown 1 = this sample does depend on others (not an I picture) 2 = this sample does not depend on others (I picture) 3 = reserved
SampleIsDependedOn	UI2	0 = the dependency of other samples on this sample is unknown 1 = other samples may depend on this one (not disposable) 2 = no other sample depends on this one (disposable) 3 = reserved
SampleHasRedundancy	UI2	0 = it is unknown whether there is redundant coding in this sample 1 = there is redundant coding in this sample 2 = there is no redundant coding in this sample 3 = reserved

When redundant coding is present, the value of 'SampleDependsOn' corresponds only to the primary coding. The parameter 'SampleIsDependedOn' is independent of the presence of redundant coding.

For more information, see section 8.6.4 of ISO/IEC 14496-12:2008.

2.8 Sample Description Box Structure

2.8.1 Sample Description box

Box type: 'stsd'

Container: Sample Table box ('stbl')

Mandatory: Yes

Quantity: One

The Sample Description (stsd) box defines the sample description for a sample table. The stsd box can contain multiple descriptions for a track, one for each media type contained in the track. The sample description table gives detailed information about the coding type used, and any initialization information needed for that coding.

Table 2 shows the hierarchy within the Sample Description box.

For more information, see section 8.5.2 of ISO/IEC 14496-12.

stsd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'stsd' (0x73747364)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
Count	UI32	Number of entries, one for each media type contained in the track
Descriptions	DESCRIPTIONRECORD [Count]	An array of boxes, one for each media type contained in the track

Each DESCRIPTIONRECORD shall be one of the following boxes:

- VisualSampleEntry, for HandlerType == 'vide' [video track],
- AudioSampleEntry, for HandlerType == 'soun' [audio track],
- MetaDataSampleEntry, for HandlerType == 'meta' [timed metadata track],
- SampleEntry, for HandlerType == 'data' [data track],
- HintSampleEntry, for HandlerType == 'hint' [hint track], or
- AdobeMuxHintSampleEntry, for HandlerType == 'hint' [Adobe Multiplexed Hint Track]

2.8.2 VisualSampleEntry box

Box type: one of the video media types specified in Section 1.8.2 Supported video types

Container: Sample Table box ('stsd')

Mandatory: Yes for video tracks

Quantity: One for each video track

The VisualSampleEntry box contains detailed information about the video coding type used, and any initialization information needed for that coding. For more information, see section 8.5.2 of ISO/IEC 14496-12:2008.

VisualSampleEntry box

Field	Type	Comment
Header	BOXHEADER	BoxType is one of the video media types specified in 1.8.2
Reserved	UI8 [6]	Set to 0
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with

		samples that use this sample description. Data references are stored in Data Reference (dref) boxes.
Predefined	UI16	Set to 0
Reserved	UI16	Set to 0
Predefined	UI32 [3]	Set to 0
Width	UI16	Max visual width (in pixels) from codec
Height	UI16	Max visual height (in pixels) from codec
HorizResolution	UI16.16	Resolution of the image pixels/inch, default value 0x00480000 (72 dpi)
VertResolution	UI16.16	Resolution of the image pixels/inch, default value 0x00480000 (72 dpi)
Reserved	UI32	Set to 0
FrameCount	UI16	How many frames are stored in each sample, default value 1 (one frame per sample)
CompressorName	UI8 [32]	Name of the compressor (for informative purpose only). First byte is set to the number of bytes of displayable data that follows first byte.
Depth	UI16	Bit depths. Default value 0x0018 (colors w/o alpha)
Predefined	SI16	Set to -1
Boxes	BOX [ ]	Additional boxes as specified for the media type, or encryption

2.8.3 AudioSampleEntry box

Box type: one of the audio media types specified in Section 1.8.1 Supported audio types

Container: Sample Table box ('stsd')

Mandatory: Yes for audio tracks

Quantity: One for each audio track

The AudioSampleEntry box contains detailed information about the audio coding type used, and any initialization information needed for that coding. For more information, see section 8.5.2 of ISO/IEC 14496-12:2008.

AudioSampleEntry box

Field	Type	Comment
Header	BOXHEADER	BoxType is one of the audio media types specified in 1.8.1
Reserved	UI8 [6]	Set to 0
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in DataReference (dref) boxes.
Reserved	UI32 [2]	Set to 0
ChannelCount	UI16	Number of channels. Default value is 2. 1 = Mono 2 = Stereo
SampleSize	UI16	Size of sample. Default value is 16
Predefined	UI16	Set to 0
Reserved	UI16	Set to 0

SampleRate	UI16.16	Sampling rate, fixed point 16.16 number
Boxes	BOX [ ]	Additional boxes as specified for the media type, or encryption

2.8.4 MetaDataSampleEntry box

Box type: depends on protocol used

Container: Sample Table box ('stsd')

Mandatory: Yes for metadata tracks

Quantity: One for each metadata track

The MetaDataSampleEntry box contains detailed information about the metadata coding type used, and any initialization information needed for that coding. For more information, see section 8.5.2 of ISO/IEC 14496-12:2008.

MetaDataSampleEntry box

Field	Type	Comment
Header	BOXHEADER	BoxType depends on protocol used
Reserved	UI8 [6]	Set to 0
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in DataReference (dref) boxes.
Data	UI8 [ ]	Additional contents as specified in ISO/IEC 14496-12:2008

2.8.5 SampleEntry box

Box type: one of the data media types specified in Section1.8.3 Supported data types

Container: Sample Table box ('stsd')

Mandatory: Yes for data tracks

Quantity: One for each data track

The SampleEntry box contains detailed information about the coding type used, and any initialization information needed for that coding. For more information, see section 8.5.2 of ISO/IEC 14496-12:2008.

SampleEntry box

Field	Type	Comment
Header	BOXHEADER	BoxType is one of the data media types specified in Supported data types.
Reserved	UI8 [6]	Set to 0.
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in Data Reference (dref) boxes.
Boxes	BOX [ ]	Additional boxes as specified for the media type, or encryption

2.8.6 HintSampleEntry box

Box type: 'hint'

Container: Sample Table box ('stsd')

Mandatory: Yes for hint tracks

Quantity: One for each hint track

The HintSampleEntry (hint) box contains appropriate declarative data for the streaming protocol being used, and the format of the hint track. For more information, see section 8.5.2 of ISO/IEC 14496-12:2008.

hint box

Field	Type	Comment
Header	BOXHEADER	In general, the BoxType depends on the protocol used. In an F4V file, only the 'hint' box type is allowed.
Reserved	UI8 [6]	Set to 0
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in Data Reference (dref) boxes.
Data	UI8 [ ]	Arbitrary number of bytes until end of box

2.8.7 Sample Descriptions for HTTP Streaming with Fragments

2.8.7.1 Adobe Mux Hint Sample Entry box

Box type: 'rtmp'

Container: Sample Table box ('stsd')

Mandatory: Yes for HTTP streaming support with F4V fragments

Quantity: One for the hint track for HTTP streaming support with F4V fragments

The Adobe Mux Hint Sample Entry (rtmp) box describes the hint track used in HTTP streaming with F4V fragments. See Annex C. HTTP Streaming: File Structure. A hint track contains AdobeMuxHintSamples.

rtmp box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'rtmp' (0x72746D70)
Reserved	UI8 [6]	Set to 0
DataReferenceIndex	UI16	Index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in Data Reference (dref) boxes.
HintTrackVersion	UI16	The version of the hint track definition being used. Set to 1.
HighestCompatibleVersion	UI16	Specifies compatibility with older versions.
MaxPacketSize	UI16	The largest Adobe Multiplexed Hint track sample packet size, in bytes.
AdditionalData	BOX [ ]	One Adobe Mux Hint Process box and zero or one Adobe Mux Time Offset boxes

2.8.7.2 Adobe Mux Hint Process box

Box type: 'amhp'

Container: Adobe Mux Hint Sample Entry ('rtmp')

Mandatory: Yes

Quantity: One

The Adobe Mux Hint Process (amhp) box contains descriptions of the hint modes used in this track.

amhp box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'amhp' (0x616D6870))
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
ModeCount	UI8	The number of mode configurations supported in this box. This value is also the number of modes supported in the corresponding hint track.
ENTRIES	MuxHintProcessEntry [ModeCount]	An array of MuxHintProcessEntry

Each MuxHintProcessEntry has the following format:

MuxHintProcessEntry

Field	Type	Comment
HintTrackMode	UI8	The mode (sample or immediate) that the entry corresponds to. For more information about the modes, see Annex C.5 Adobe Multiplexed Hint Track Format.
TrailerLengthField	UI1	1 indicates the presence of the TrailerLength field in the AdobeMuxHintSample for this mode. If 0, then TrailerDefaultSize is used.
LengthField	UI1	1 indicates the presence of the hint sample Length field in the AdobeMuxHintSample for this mode. LengthField shall be 1 for mode == 2.
ModeField	UI1	1 indicates the presence of the Mode field in the AdobeMuxHintSample for this mode. When multiple modes are used, ModeField shall be 1.
ConstructorCountField	UI1	1 indicates the presence of the ConstructorCount field in the AdobeMuxHintSample for this mode. If 0, there is one constructor
PacketCountField	UI1	1 indicates the presence of the PacketCount field in the AdobeMuxHintSample for this mode.
Reserved	UI3	Set to 0
TrailerDefaultSize	UI8	The default size of trailer data, in bytes, after the hint sample payload. Used when the TrailerLengthField is not present.

For the FLV compatible mode, TrailerLengthField, LengthField, ModeField, ConstructorCountField, and PacketCountField shall be 0. In this case, the Immediate noDuplication hinting mode is used.

2.8.7.3 Adobe Mux Time Offset box

Box type: 'amto'

Container: Adobe Mux Hint Sample Entry ('rtmp')

Mandatory: No

Quantity: One

The Adobe Mux Time Offset (amto) box stores the timestamp of the first hint sample in this file.

amto box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'amto' (0x616D746F)
TimeOffset	UI32	The timestamp of the first hint sample in the file. This timestamp is the offset that is added to the presentation time of each sample derived from the cumulative sample durations.

2.8.8 Sample Descriptions for Protected Contents

2.8.8.1 Encrypted Video box

Box type: 'encv'

Container: Sample Table box ('stsd')

Mandatory: Yes for encrypted video tracks

Quantity: One for each encrypted video track

The Encrypted Video (encv) box shall be the original VisualSampleEntry with a 'sinf' box appended, as described in Annex D.2. For more information, see section 8.12 of ISO/IEC 14496-12:2008.

2.8.8.2 Encrypted Audio box

Box type: 'enca'

Container: Sample Table box ('stsd')

Mandatory: Yes for encrypted audio tracks

Quantity: One for each encrypted audio track

The Encrypted Audio (enca) box shall be the original AudioSampleEntry with a 'sinf' box appended, as described in Annex D.2. For more information, see section 8.12 of ISO/IEC 14496-12:2008.

2.8.8.3 Encrypted Data box

Box type: 'encr'

Container: Sample Table box ('stsd')

Mandatory: Yes for encrypted data tracks

Quantity: One for each encrypted data track

The Encrypted Data (encr) box shall be the original SampleEntry with a 'sinf' box appended, as described in Annex D.2. For more information, see section 8.12 of ISO/IEC 14496-12:2008.

2.8.8.4 Protection Scheme Information box

Box type: 'sinf'

Container: 'encv' or 'enca' Sample Description Entry for the protected track in 'stsd' box

Mandatory: Yes

Quantity: One

The Protection Scheme Information (sinf) box is a container box with all the information required both to understand the encryption transform applied and its parameters, and to find other information such as the kind and location of the key management system. It also documents the original (unencrypted) format of the media. It shall be appended to any sample entry that has a four-character code ('encv', 'enca', or 'encr') indicating a protected stream.

sinf box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'sinf'
OriginalFormatBox	OriginalFormatBox	The format of the original sample
SchemeTypeBox	SchemeTypeBox	The DRM type. Required. (This is optional in ISO 14496-12)
SchemeInformationBox	SchemeInformationBox	DRM details. Required. (This is optional in ISO 14496-12)

For more information, see section 8.12.1 of ISO/IEC 14496-12:2008.

2.8.8.5 Original Format box

Box type: 'frma'

Container: Protection Scheme Information box ('sinf')

Mandatory: Yes

Quantity: One

The Original Format (frma) box specifies the format of the original sample, e.g. "mp4v" if the stream contains protected MPEG-4 visual material.

frma box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'frma'
UnencryptedDataFormat	UI32	The four-character-code of the original un-transformed sample entry

For more information, see section 8.12.2 of ISO/IEC 14496-12:2008.

2.8.8.6 Scheme Type box

Box type: 'schm'

Container: Protection Scheme Information box ('sinf')

Mandatory: Yes

Quantity: One

The Scheme Type (schm) box specifies the DRM system used to manage keys and decryption of the content. As the media file format may support other key management systems other then Adobe's DRM, the key management system in use shall be indicated by a four-character code (4CC) in the SchemeType field.

schm box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'schm'
Version	UI8	Shall be 1
Flags	UI24	Shall be 0 or 1
SchemeType	UI32	The scheme type. Shall be 'adkm', indicating that the content is protected using Adobe's DRM system
SchemeVersion	UI32	Shall be 1
SchemeUri	IF Flags == 1 STRING	Browser URI

For more information, see section 8.12.5 of ISO/IEC 14496-12:2008.

2.8.8.7 Scheme Information box

Box type: 'schi'

Container: Protection Scheme Information box ('sinf')

Mandatory: Yes

Quantity: One

The Scheme Information (schi) box is a container box carrying DRM key/rights management system specific information. For Adobe's DRM, this box shall include one Adobe DRM Key Management System box. There may be other boxes present. For interoperability with other DRMs, the Adobe DRM Key Management System box may be located anywhere in the Scheme Information box.

schi box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'schi'
OtherDRMSpecificData	BOX [ ]	(Optional) Boxes containing other DRM specific key management information
SchemeSpecificData	AdobeDRMKMSBox	Adobe DRM Key Management System box specifying key management information
OtherDRMSpecificData	BOX [ ]	(Optional) Boxes containing other DRM specific key management information

For more information, see Section 8.12.6, ISO 14496-12:2008

2.8.8.8 Boxes for Adobe's Protection Scheme

The following boxes are defined by Adobe, and are not documented in ISO 14496-12:2008.

2.8.8.8.1 Adobe DRM Key Management System box

Box type: 'adkm'

Container: Scheme Information box ('schi')

Mandatory: Yes

Quantity: One

The Adobe DRM Key Management System (adkm) box specifies encryption and sample formatting.

adkm box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'adkm'
Version	UI8	Shall be 1
Flags	UI24	Shall be 0
Header	AdobeDRMHeaderBox	Adobe DRM Header box specifying how to retrieve the key and how to use it to decrypt the content
AUFormat	AdobeDRMAUFormatBox	Adobe DRM Access Unit Format box specifying the formatting prepended to each sample

2.8.8.8.2 Adobe DRM Header box

Box type: 'ahdr'

Container: Adobe DRM Key Management System box ('adkm')

Mandatory: Yes

Quantity: One

The Adobe DRM Header (ahdr) box specifies the version of the encryption format and methods.

ahdr box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'ahdr'
Version	UI8	Shall be 1 or 2, indicating the version of the encryption format. 1 = FMRMS v1.x products. 2 = Flash Access 2.0 products. Contents protected using either version are in existence, hence applications shall be able to consume both versions of the content
Flags	UI24	Shall be 0
StdEncryptionBox	StandardEncryptionParamsBox	Standard Encryption Params box containing the encryption method used to encrypt the samples is of type 'Standard Encryption'
Signature	IF Version == 1 AdobeSignatureBox	AdobeSignatureBox is not described in this document

2.8.8.8.3 Standard Encryption Params box

Box type: 'aprm'

Container: Adobe DRM Header box ('ahdr')

Mandatory: Yes

Quantity: One

The Standard Encryption Params (aprm) box contains parameters for the encryption method 'Standard'.

aprm box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'aprm'
Version	UI8	Shall be 1
Flags	UI24	Shall be 0
EncInfoBox	EncryptionInfoBox	Encryption Information box specifying the encryption algorithm used to encrypt the samples
KeyInfoBox	KeyInfoBox	Key Information box specifying how to retrieve the key for the decryption of samples

2.8.8.8.4 Encryption Information box

Box type: 'aeib'

Container: Standard Encryption Params box ('aprm')

Mandatory: Yes

Quantity: One

The Encryption Information (aeib) box specifies the encryption algorithm used to encrypt the samples.

aeib box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'aeib'
Version	UI8	Shall be 1
Flags	UI24	Shall be 0
EncryptionAlgorithm	STRING	The encryption algorithm. Shall be 'AES-CBC', specifying that the encryption used is 'AES-CBC' with padding as per RFC 2630
KeyLength	UI8	Key length of encryption/decryption algorithm in bytes. Shall be 16 (i.e. 128 bits)

2.8.8.8.5 Key Information box

Box type: 'akey'

Container: Standard Encryption Params box ('aprm')

Mandatory: Yes

Quantity: One

The Key Information (akey) box contains information for retrieving the key for decryption of samples.

The details of the entries contained in these boxes, and the mechanism used by the DRM client to retrieve the keys are outside the scope of this specification.

akey box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'akey'
Version	UI8	Shall be 1
Flags	UI24	Shall be 0
Params	IF AdobeDRMHeaderBox.Version == 1 APSParamsBox ELSE FMRMSv2ParamsBox	APSParamsBox is not described in this document as it will no longer be produced by conforming applications

2.8.8.8.6 Flash Access Params box

Box type: 'flxs'

Container: Key Info box ('akey')

Mandatory: Yes, if AdobeDRMHeaderBox.Version == 2, else No

Quantity: One, if AdobeDRMHeaderBox.Version == 2, else Zero

The Flash Access Params (flxs) box contains information for retrieving the key for decryption of samples.

flxs box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'flxs'
FmrmsV2Metadata	STRING	Base64-encoded metadata used by the DRM client to retrieve decryption key

2.8.8.8.7 Adobe DRM Access Unit Format box

Box type: 'adaf'

Container: Key Info box ('adkm')

Mandatory: Yes

Quantity: One

The Access Unit Format (adaf) box specifies the format of the headers placed on the samples.

adaf box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'adaf'
Version	UI8	Shall be 0
Flags	UI24	Shall be 0
SelectiveEncryption	UI1	Indicates use of Selective Encryption. Shall be 1. 1 = Selective encryption is turned on, i.e. only some samples are encrypted, not all. 0 = Selective encryption is turned off, and all samples are encrypted
Reserved	UI7	Shall be 0
Reserved	UI8	Shall be 0

IVLength	UI8	The size of the initialization vector in bytes. This length should be consistent with the algorithms used. Shall be 16 (128 bits)

2.9 Movie Extends box

Box type: 'mvex'

Container: Movie box ('moov')

Mandatory: No

Quantity: One

If the F4V file contains fragments, then the Movie (moov) box contains one Movie Extends (mvex) box, otherwise there is no mvex box. For fragments, an F4V file shall contain one and only one mvex box. The mvex box tells readers that this file might contain Movie Fragment (moof) boxes.

mvex box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mvex' (0x6D766578)
Boxes	BOX [ ]	Boxes defining the track defaults values for the fragments

For more information, see section 8.8.1 of ISO/IEC 14496-12:2008.

2.9.1 Movie Extends Header box

Box type: 'mehd'

Container: Movie Extends box ('mvex')

Mandatory: No

Quantity: One

The Movie Extends Header (mehd) box provides the duration of the fragmented movie. If the Movie Extends (mvex) box does not contain an mehd box, the overall duration is computed by examining all the fragments.

mehd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mehd' (0x6D766578)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
FragmentDuration	If Version==0 UI32 If Version ==1 UI64	Duration of the longest track in TimeScale units defined in the mvhd box

For more information, see section 8.8.2 of ISO/IEC 14496-12:2008.

2.9.2 Track Extends box

Box type: 'trex'

Container: Movie Extends box ('mvex')

Mandatory: Yes

Quantity: One for each track in the Movie box

The Track Extends (trex) box defines the default values for the movie fragments.

trex box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'trex' (0x74726578)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
TrackID	UI32	Identity of the associated track
DefaultSampleDescriptionIndex	UI32	Default SampleDescriptionIndex to be used in track fragments
DefaultSampleDuration	UI32	Default SampleDuration to be used in track fragments
DefaultSampleSize	UI32	Default SampleSize to be used in track fragments
DefaultSampleFlags	SAMPLEFLAGS	Default SampleFlags to be used in track fragments

Each SAMPLEFLAGS record has the following layout:

SAMPLEFLAGS

Field	Type	Comment
Reserved	UI6	Reserved. Set to 0.
SampleDependsOn	UI2	0 = the sample dependency is unknown 1 = this sample does depend on others (not an I picture) 2 = this sample does not depend on others (I picture) 3 = reserved
SampleIsDependedOn	UI2	0 = the dependency of other samples on this sample is unknown 1 = other samples may depend on this one (not disposable) 2 = no other sample depends on this one (disposable) 3 = reserved
SampleHasRedundancy	UI2	0 = it is unknown whether there is redundant coding in this sample 1 = there is redundant coding in this sample 2 = there is no redundant coding in this sample 3 = reserved
SamplePaddingValue	UI3	Reserved. Set to 0
SampleIsDifferenceSample	UI1	0 = a key or sync sample 1 = a non-key or non-sync sample
SampleDegradationPriority	UI16	Reserved. Set to 1.

For more information, see section 8.8.3 of ISO/IEC 14496-12:2008.

2.10 User Data box

Box type: 'udta'

Container: Movie box ('moov') or Track box ('trak')

Mandatory: No

Quantity: One at each movie level or track level

The User Data (udta) box is contained within the Movie (moov) box or Track (trak) box. At most, one udta box may occur at each movie level or track level. The udta box should be placed last in its containing box.

The udta box declares free-form user information about the containing box and its data (presentation or track). Flash Player ignores the contents of udta boxes.

udta box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'udta' (0x75647461)
UserData	BOX [ ]	Arbitrary number of boxes with free-form user data

For more information, see section 8.10.1 of ISO/IEC 14496-12:2008.

2.11 F4V Boxes for HTTP Streaming

2.11.1 Fragment Random Access box

Box type: 'afra'

Container: File

Mandatory: Yes for HTTP streaming support with F4V fragments, otherwise no.

Quantity: One per fragment for HTTP streaming support with F4V fragments, otherwise zero.

The Fragment Random Access (afra) box provides random access information to one or more fragments.

For HTTP streaming support with F4V fragments, the F4V file can contain one afra box for each fragment. The afra box shall be located before the fragment's Media Data (mdat) and Movie Fragment (moof) boxes. The afra box can be used to seek to the exact point in the F4V file that contains the closest random access sample to a given time.

The afra box is associated with a given fragment (here referred to as “the associated fragment”). The afra box also provides random access to information in other fragments in the same segment or different segments.

The afra box contains arrays of entries. Each entry contains the location and the presentation time of a random access sample. If a random access sample is not within the associated fragment, the entry also provides the following information:

- Segment identifying information
- Fragment identifying information
- The byte offset from the beginning of the containing segment to the ‘afra’ box associated with this random access point
- The byte offset from the associated ‘afra’ box to the sample

Note: Every random access sample in a fragment does not necessarily have an array entry.

The absence of the afra box does not mean that all the samples are sync samples. Random access information in the 'trun', 'traf', and 'trex' are set appropriately regardless of the presence of this box.

afra box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'afra' (0x61667261)

Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
LongIDs	UI1	Controls the size of the Segment and Fragment fields of a GLOBALAFRAENTRY.
LongOffsets	UI1	Controls the size of the Offset field of an AFRAENTRY. Also controls the size of the AfraOffset and OffsetFromAfra fields of a GLOBALAFRAENTRY.
GlobalEntries	UI1	The value 1 indicates that GlobalEntryCount is present
Reserved	UI5	Set to 0
TimeScale	UI32	The number of time units per second, used in the Time field of AFRAENTRY and GLOBALAFRAENTRY.
EntryCount	UI32	The number of entries in LocalAccessEntries
LocalAccessEntries	AFRAENTRY [EntryCount]	Random access to points in this fragment. This array does not necessarily contain an entry for every random access sample in this fragment.
GlobalEntryCount	IF GlobalEntries == 1 UI32	The number of entries in GlobalAccessEntries. If GlobalEntries == 0, then this field is not present and GlobalEntryCount is 0.
GlobalAccessEntries	GLOBALAFRAENTRY [GlobalEntryCount]	Random access to points outside this fragment.

Each AFRAENTRY points to a sample within this fragment, and has the following format:

AFRAENTRY

Field	Type	Comment
Time	UI64	The presentation time of the random access sample, in TimeScale units
Offset	IF LongOffsets == 0 UI32 ELSE UI64	The byte offset from the beginning of this Fragment Random Access box to the sample

Each GLOBALAFRAENTRY points to a sample outside this fragment, and has the following format:

GLOBALAFRAENTRY

Field	Type	Comment
Time	UI64	The presentation time of the random access sample, in TimeScale units
Segment	IF LongIDs == 0 UI16 ELSE UI32	The number of the segment containing this random access point

Fragment	IF LongIDs == 0 UI16 ELSE UI32	The number of the fragment containing this random access point
AfraOffset	IF LongOffsets == 0 UI32 ELSE UI64	The byte offset from the beginning of the containing segment to the afra box associated with this random access point
OffsetFromAfra	IF LongOffsets == 0 UI32 ELSE UI64	The byte offset from the associated afra box to the sample

2.11.2 Bootstrap Info box

Box type: 'abst'

Container: File

Mandatory: Yes for HTTP streaming support with F4V fragments, otherwise no.

Quantity: One or more for HTTP streaming support with F4V fragments, otherwise zero.

A Bootstrap Info (abst) box contains the information necessary to bootstrap the media-presentation URL requests RFC1630 from the media client to the HTTP server. The media presentation can be either a live or a video-on-demand scenario. This box contains basic information about the server, movie, and segment information. It also contains one or more segment run tables and fragment run tables.

In the HTTP streaming segment, the abst box is optional and precedes the Movie (moov) box. In the HTTP streaming fragment, the abst box is required. For a description of the boxes and structure required for HTTP streaming, see Annex C. HTTP Streaming: File Structure.

abst box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'abst' (0x61627374)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved. Set to 0
BootstrapinfoVersion	UI32	The version number of the bootstrap information. When the Update field is set, BootstrapinfoVersion indicates the version number that is being updated.
Profile	UI2	Indicates if it is the Named Access (0) or the Range Access (1) Profile. One bit is reserved for future profiles.
Live	UI1	Indicates if the media presentation is live (1) or not.

Update	UI1	Indicates if this table is a full version (0) or an update (1) to a previously defined (sent) full version of the bootstrap box or file. Updates are not complete replacements. They may contain only the changed elements. The server sends the updates only when the bootstrap information changes. The updates apply to the full version with the same BootstrapinfoVersion number. There may be more than one update for the same BootstrapinfoVersion number. If the server sends multiple updates, the updates apply to the full version with the same BootstrapinfoVersion number. Each update includes all previous updates to the same BootstrapinfoVersion. For multiple updates to a single full version, the latest update is determined based on the CurrentMediaTime.
Reserved	UI4	Reserved, set to 0
TimeScale	UI32	The number of time units per second. The field CurrentMediaTime uses this value to represent accurate time. Typically, the value is 1000, for a unit of milliseconds.
CurrentMediaTime	UI64	The timestamp in TimeScale units of the latest available Fragment in the media presentation. This timestamp is used to request the right fragment number. The CurrentMediaTime can be the total duration. For media presentations that are not live, CurrentMediaTime can be 0.
SmpteTimeCodeOffset	UI64	The offset of the CurrentMediaTime from the SMPTE time code, converted to milliseconds. This offset is not in TimeScale units. This field is zero when not used. The server uses the SMPTE time code modulo 24 hours to make the offset positive.
MovieIdentifier	STRING	The identifier of this presentation. The identifier is a null-terminated UTF-8 string. For example, it can be a filename or pathname in a URL. See Annex C.4 URL Construction for usage.
ServerEntryCount	UI8	The number of ServerEntryTable entries. The minimum value is 0.
ServerEntryTable	SERVERENTRY [ServerEntryCount]	Server URLs in descending order of preference
QualityEntryCount	UI8	The number of QualityEntryTable entries, which is also the number of available quality levels. The minimum value is 0. Available quality levels are for, for example, multi bit rate files or trick files.
QualityEntryTable	QUALITYENTRY [QualityEntryCount]	Quality file references in order from high to low quality

DrmData	STRING	Null or null-terminated UTF-8 string. This string holds Digital Rights Management metadata. Encrypted files use this metadata to get the necessary keys and licenses for decryption and playback.
MetaData	STRING	Null or null-terminated UTF-8 string that holds metadata
SegmentRunTableCount	UI8	The number of entries in SegmentRunTableEntries. The minimum value is 1. Typically, one table contains all segment runs. However, this count provides the flexibility to define the segment runs individually for each quality level (or trick file).
SegmentRunTableEntries	SegmentRunTable [SegmentRunTableCount]	Array of SegmentRunTable elements
FragmentRunTableCount	UI8	The number of entries in FragmentRunTable-Entries. The minimum value is 1.
FragmentRunTableEntries	FragmentRunTable [FragmentRunTableCount]	Array of FragmentRunTable elements

Each SERVERENTRY has the following format:

SERVERENTRY

Field	Type	Comment
ServerBaseURL	STRING	The server base url for this presentation on that server. The value is a null-terminated UTF-8 string, without a trailing "/".

Each QUALITYENTRY has the following format:

QUALITYENTRY

Field	Type	Comment
QualitySegmentUrlModifier	STRING	Name of the quality (segment) file that is used to construct the right URL for that quality media. The value is a null-terminated UTF-8 string, optionally with a trailing "/".

2.11.2.1 Segment Run Table box

Box type: 'asrt'

Container: Bootstrap Info box ('abst')

Mandatory: Yes

Quantity: One or more

The Segment Run Table (asrt) box can be used to locate a segment that contains a particular fragment.

There may be several asrt boxes, each for different quality levels.

The asrt box uses a compact encoding:

- A Segment Run Table may represent fragment runs for several quality levels.
- The Segment Run Table is compactly coded. Each entry gives the first segment number for a run of segments with the same count of fragments. The count of segments having this same count of fragments can be calculated by subtracting the first segment number in this entry from the first segment number in the next entry.

asrt box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'asrt' (0x61737274)
Version	UI8	Either 0 or 1
Flags	UI24	The following values are defined: 0 = A full table. 1 = The records in this table are updates (or new entries to be appended) to the previously defined Segment Run Table. The Update flag in the containing Bootstrap Info box shall be 1 when this flag is 1.
QualityEntryCount	UI8	The number of QualitySegmentUrlModifiers (quality level references) that follow. If 0, this Segment Run Table applies to all quality levels, and there shall be only one Segment Run Table box in the Bootstrap Info box.
QualitySegmentUrlModifiers	STRING [QualityEntryCount]	An array of names of the quality levels that this table applies to. The names are null-terminated UTF-8 strings. The array shall be a subset of the QualityEntryTable in the Bootstrap Info (abst) box. The names shall not be present in any other Segment Run Table in the Bootstrap Info box.
SegmentRunEntryCount	UI32	The number of items in this SegmentRunEntryTable. The minimum value is 1.
SegmentRunEntryTable	SEGMENTRUNENTRY [SegmentRunEntryCount]	Array of segment run entries

Each SEGMENTRUNENTRY has the following format:

SEGMENTRUNENTRY

Field	Type	Comment
FirstSegment	UI32	The identifying number of the first segment in the run of segments containing the same number of fragments. The segment corresponding to the FirstSegment in the next SEGMENTRUNENTRY will terminate this run.
FragmentsPerSegment	UI32	The number of fragments in each segment in this run.

2.11.2.2 Fragment Run Table box

Box type: 'afrt'

Container: Bootstrap Info box ('abst')

Mandatory: Yes

Quantity: One or more

The Fragment Run Table (afrt) box can be used to find the fragment that contains a sample corresponding to a given time.

Fragments are individually identifiable by the URL scheme. Fragments may vary both in duration and in number of samples. The Durations of the Fragments are stored in the afrt box.

The afrt box uses a compact encoding:

- A Fragment Run Table may represent fragments for more than one quality level.
- The Fragment Run Table is compactly coded, as each entry gives the first fragment number for a run of fragments with the same duration. The count of fragments having this same duration can be calculated by subtracting the first fragment number in this entry from the first fragment number in the next entry.

There may be several Fragment Run Table boxes in one Bootstrap Info box, each for different quality levels.

afrt box

Field	Type	Comment
Header	BOXHEADER	BoxType ='afrt' (0x61667274)
Version	UI8	Either 0 or 1
Flags	UI24	The following values are defined: 0 = A full table. 1 = The records in this table are updates (or new entries to be appended) to the previously defined Fragment Run Table. The Update flag in the containing Bootstrap Info box shall be 1 when this flag is 1.
TimeScale	UI32	The number of time units per second, used in the FirstFragmentTimestamp and FragmentDuration fields. Typically, the value is 1.
QualityEntryCount	UI8	The number of QualitySegmentUrlModifiers (quality level references) that follow. If 0, this Fragment Run Table applies to all quality levels, and there shall be only one Fragment Run Table box in the Bootstrap Info box.
QualitySegmentUrlModifiers	STRING [QualityEntryCount]	An array of names of the quality levels that this table applies to. The names are null-terminated UTF-8 strings. The array shall be a subset of the QualityEntryTable in the Bootstrap Info (abst) box. The names shall not be present in any other Fragment Run Table in the Bootstrap Info box
FragmentRunEntryCount	UI32	The number of items in this FragmentRunEntryTable. The minimum value is 1.
FragmentRunEntryTable	FRAGMENTRUNENTRY [FragmentRunEntryCount]	Array of fragment run entries

Each FRAGMENTRUNENTRY has the following format:

FRAGMENTRUNENTRY		
Field	**Type**	**Comment**
FirstFragment	UI32	The identifying number of the first fragment in this run of fragments with the same duration. The fragment corresponding to the FirstFragment in the next FRAGMENTRUNENTRY will terminate this run.
FirstFragmentTimestamp	UI64	The timestamp of the FirstFragment, in TimeScale units. This field ensures that the fragment timestamps can be accurately represented at the beginning. It also ensures that the timestamps are synchronized when drifts occur due to duration accuracy or timestamp discontinuities.
FragmentDuration	U32	The duration, in TimeScale units, of each fragment in this run
DiscontinuityIndicator	IF FragmentDuration == 0 UI8	Indicates discontinuities in timestamps, fragment numbers, or both. This field is also used to identify the end of a (live) presentation. The following values are defined: 0 = end of presentation. 1 = a discontinuity in fragment numbering. 2 = a discontinuity in timestamps. 3 = a discontinuity in both timestamps and fragment numbering. All other values are reserved. Signaling the end of the presentation in-band is useful in live scenarios. Gaps in the presentation are signaled as a run of zero duration fragments with both fragment number and timestamp discontinuities. Fragment number discontinuities are useful to signal jumps in fragment numbering schemes with no discontinuities in the presentation.

2.12 Movie Fragment box

Box type: 'moof'

Container: File

Mandatory: Yes for HTTP streaming support with F4V fragments, otherwise no.

Quantity: One per fragment for HTTP streaming support with F4V fragments, otherwise zero.

The Movie Fragment (moof) box provides segment-specific information that would otherwise be in the Media (moov) box. The moof boxes shall be in sequence order.

moof box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'moof' (0x6D6F6F66)
Boxes	BOX [ ]	A number of boxes that define the sample structure

For more information, see section 8.8.4 of ISO/IEC 14496-12:2008.

2.12.1 Movie Fragment Header box

Box type: 'mfhd'

Container: Movie Fragment box ('moof')

Mandatory: Yes

Quantity: One

The Movie Fragment Header (mfhd) box contains a sequence number to verify the integrity of the file.

mfhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mfhd' (0x6D666864)
Version	UI8	Expected to be 0
Flags	UI24	Reserved. Set to 0
SequenceNumber	UI32	Starts at 1 and increments in the order of occurrence for each movie fragment in the file.

For more information, see section 8.8.5 of ISO/IEC 14496-12:2008.

2.12.2 Track Fragment box

Box type: 'traf'

Container: Movie Fragment box ('moof')

Mandatory: No

Quantity: Zero or more

The Track Fragment (traf) box corresponds to a track in the F4V file. Each traf box contains zero or more track runs, which comprise a contiguous run for that track.

traf box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'traf' (0x74726166)
Boxes	BOX [ ]	Arbitrary number of boxes that define the track runs in the fragment

For more information, see section 8.8.6 of ISO/IEC 14496-12:2008.

2.12.2.1 Track Fragment Header box

Box type: 'tfhd'

Container: Track Fragment box ('traf')

Mandatory: Yes

Quantity: One

The Track Fragment Header (tfhd) box sets up information and defaults used for the runs of samples in a movie fragment. Each movie fragment can add zero or more fragments to each track, and a track fragment can add zero or more contiguous runs of samples.

tfhd box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'tfhd' (0x74666864)
Version	UI8	Expected to be 0

Flags	UI24	The following flags may be used in any combination: 0x000001 = base data offset present 0x000002 = sample description present 0x000008 = default sample duration present 0x000010 = default sample size present 0x000020 = default sample flags present 0x010000 = duration-is-empty: there are no samples for the duration provided by DefaultSampleDuration in either the tfhd box or the trex box. See note below table
TrackID	UI32	Identity of the associated track, as specified in the Track Header box
BaseDataOffset	IF Flags & 0x000001 == true UI64	Optional. The base offset to use when calculating data offsets in each track run. The default value is defined below the table.
SampleDescriptionIndex	IF Flags & 0x000002 == true UI32	Optional. SampleDescriptionIndex to be used in this fragment. This shall override the DefaultSampleDescriptionIndex in the trex box for this fragment.
DefaultSampleDuration	IF Flags & 0x000008 == true UI32	Optional. Default SampleDuration to be used in this fragment. This shall override the DefaultSampleDuration in the trex box for this fragment
DefaultSampleSize	IF Flags & 0x000010 == true UI32	Optional. Default SampleSize to be used in this fragment. This shall override the DefaultSampleSize in the trex box for this fragment
DefaultSampleFlags	IF Flags & 0x000020 == true SAMPLEFLAGS	Optional. Default SampleFlags to be used in this fragment. This shall override the DefaultSampleFlags in the trex box for this fragment

BaseDataOffset: If a value is not provided here, the default value for the first track in the movie fragment is the position of the first byte of the enclosing Movie Fragment box. For subsequent track fragments, the default is the end of the data defined by the preceding fragment. Fragments that are "inheriting" their offset in this way shall all use the same data-reference, that is, the data for these tracks shall be in the same file.

Note: 0x010000 duration-is-empty: If an F4V document has edit lists in the moov box and has empty duration fragments, it is considered malformed.

For more information, see section 8.8.7 of ISO/IEC 14496-12:2008.

2.12.2.2 Track Fragment Run box

Box type: 'trun'

Container: Track Fragment box ('traf')

Mandatory: No

Quantity: Zero or more

A Track Fragment Run (trun) box defines a contiguous set of samples for a track. If the duration-is-empty flag is set in the Track Fragment box (traf) box, there are no trun boxes.

trun box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'trun' (0x7472756E)
Version	UI8	Expected to be 0
Flags	UI24	The following flags may be used in any combination, except both 0x000004 and 0x000400: 0x000001 = data-offset-present 0x000004 = first-sample-flags-present 0x000100 = sample duration present 0x000200 = sample size present 0x000400 = sample flags present 0x000800 = sample composition time offsets present
SampleCount	UI32	The number of entries in SampleInformation
DataOffset	IF Flags & 0x000001 == true SI32	Optional. Value to be added to data offset defined in tfhd box. The default value is defined below the table.
FirstSampleFlags	IF Flags & 0x000004 == true SAMPLEFLAGS	Optional. Flag to be used only for the first sample of the set described in this trun box. See text below table.
SampleInformation	SampleInformationStructure [SampleCount]	All fields within the structure are optional

DataOffset: If the data-offset is not present, the data for this run starts at one of two locations. If this run is the first in a track fragment, it starts at the base-data-offset defined by the track fragment header. Otherwise, it starts immediately after the data of the previous run.

FirstSampleFlags: Override the default flags for the first sample only. This makes it possible to record a group of frames where the first is a key and the rest are difference frames, without supplying explicit flags for every sample. When this flag is set, sample-flags shall not be present.

Each SampleInformationStructure record has the following layout:

SampleInformationStructure

Field	Type	Comment
SampleDuration	IF Flags & 0x000100 == true UI32	Optional. The duration of each sample, in TimeScale units defined in the Media Header for this track. If not present, default is used.
SampleSize	IF Flags & 0x000200 == true UI32	Optional. The size of each sample. If not present, default is used.
SampleFlags	IF Flags & 0x000400 == true SAMPLEFLAGS	Optional. The SampleFlags for each sample. If not present, default is used.

SampleCompositionTimeOffset	IF Flags & 0x000800 == true UI32	Optional. The composition time offset for each sample. If not present, default is used.

For more information, see section 8.8.8 of ISO/IEC 14496-12:2008.

2.13 Media Data box

Box type: 'mdat'

Container: File

Mandatory: Yes

Quantity: One

A Media Data (mdat) box contains the media data payload for the F4V file. All video samples, audio samples, data samples, and hint tracks and samples are contained in the mdat box. See 1.8 Supported Media Types.

The mdat box occurs at the top level of an F4V file, along with the Media (moov) box.

The mdat box cannot be understood on its own, which is why a moov box must be present in the file as well.

mdat box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mdat' (0x6D646174)
Payload	UI8 [ ]	Bytes of media data, the structure of which is defined in the file's moov box

For more information, see section 8.2.2 of ISO/IEC 14496-12:2008.

2.13.1 Hint Track Samples for HTTP Streaming

The mdat box contains the hint track used for HTTP streaming with F4V fragments. The hint track contains AdobeMuxHintSamples. The Adobe Mux Hint Sample Entry box (rtmp) describes the hint track.

2.13.1.1 AdobeMuxHintSample

A collection of AdobeMuxHintSamples makes up a hint track that is in the Adobe Multiplexed Hint Track Format.

An AdobeMuxHintSample has the following layout:

AdobeMuxHintSample

Field	Type	Comment
PacketCount	IF PacketCountField == 1 UI8	Number of AdobeMuxPacket entries in this AdobeMuxHintSample. When PacketCountField == 0, AdobeMuxPackets are self-describing and the number can be implicitly determined.
Packets	AdobeMuxPacket [PacketCount]	Array of AdobeMuxPacket elements

2.13.1.2 AdobeMuxPacket

An AdobeMuxPacket has the following layout. The part up to and including EncryptionHeader is identical to the corresponding part in the FLVTAG defined in Section E.4.1.

AdobeMuxPacket

Field	Type	Comment
Reserved	UI2	Reserved for FMS, should be 0

Filter	UI1	Indicates if packets are filtered. 0 = No pre-processing required. 1 = Pre-processing (such as decryption) of the packet is required before it can be rendered. Shall be 0 in unencrypted files, and 1 for encrypted tags. See Annex F. FLV Encryption for the use of filters.
TagType	UI5	Type of this tag. The following types are defined: 8 = audio 9 = video All other values are reserved (18 = script data shall not be used)
DataSize	UI24	Length of the message. Number of bytes after StreamID to end of packet (Equal to packet length – 11)
Timestamp	UI24	Time in milliseconds at which the data in this tag applies. This value is relative to the first tag in the FLV file, which always has a timestamp of 0.
TimestampExtended	UI8	Extension of the Timestamp field to form a SI32 value. This field represents the upper 8 bits, while the previous Timestamp field represents the lower 24 bits of the time in milliseconds.
StreamID	UI24	Always 0.
AudioTagHeader	IF TagType == 8 AudioTagHeader	AudioTagHeader element as defined in Section E.4.2.1.
VideoTagHeader	IF TagType == 9 VideoTagHeader	VideoTagHeader element as defined in Section E.4.3.1.
EncryptionHeader	IF Filter == 1 EncryptionTagHeader	Encryption header shall be included for each protected sample, as defined in Section F.3.1.
ConstructorCount	IF ConstructorCountField == 1 UI8	Number of AdobeMuxHintConstructors. This field is particularly used when a single FLV Tag or an RTMP Message is constructed using multiple data blocks using different modes or different parts of the original sample. If ConstructorCountField == 0, then ConstructorCount = 1
DataEntry	AdobeMuxHintConstructor [ConstructorCount]	Array of AdobeMuxHintConstructors elements
TrailerLength	If TrailerLengthField == 1 UI8	Length of the trailer, in bytes. If TrailerLengthField == 0, then TrailerLength = TrailerDefaultSize
Trailer	UI8 [TrailerLength]	Additional data, for example, for compatibility. When in FLV compatibility mode, this field carries the PreviousTagSize.

2.13.1.3 AdobeMuxHintConstructor

An AdobeMuxHintConstructor has the following layout:

AdobeMuxHintConstructor

Field	Type	Comment
Mode	If ModeField == 1 UI8	Hint track mode being used. When ModeField == 0, the mode can be determined from the Adobe Mux Hint Process (amhp) box.
HintInfo	If Mode == 2 AdobeMuxHintSampleConstructor ELSE AdobeMuxHintImmediateConstructor	As indicated by mode. Although there are three hinting modes defined, only two Constructors are specified since both the Immediate and the Immediate noDuplication modes use the AdobeMuxHintImmediateConstructor.

2.13.1.4 AdobeMuxHintImmediateConstructor

The AdobeMuxHintImmediateConstructor shall be used in the Immediate and Immediate NoDuplication modes. These modes are described in the Adobe Multiplexed Hint Track Format.

The AdobeMuxHintImmediateConstructor has the following layout:

AdobeMuxHintImmediateConstructor

Field	Type	Comment
Length	If LengthField == 1 UI24	Number of bytes to take from the data that follows. If LengthField == 0, this field is not present, and the length is computed from AdobeMuxPacket.DataSize.
Data	UI8 [Length]	Bytes of data to place into the payload portion

2.13.1.5 AdobeMuxHintSampleConstructor

The AdobeMuxHintSampleConstructor shall be used in the sample mode. The sample mode is described in the Adobe Multiplexed Hint Track Format.

An AdobeMuxHintSampleConstructor has the following layout:

AdobeMuxHintSampleConstructor

Field	Type	Comment
TrackRefIndex	SI8	Value that indicates which track the sample data will come from. A value of 0 means that exactly one media track is referenced. Values from 1 to 127 are indexes into the Hint track reference Atom entries. These values indicate which original media track the sample is to be read from. A value of -1 means the hint track itself. That is, get the sample from the same track as the hint sample you are currently parsing.
Length	UI24	Number of bytes in the sample to copy. LengthField shall be 1 for Mode == 2.
SampleNumber	UI32	Sample number of the track.
SampleOffset	UI32	Offset from the start of the sample to the point where to start copying

2.14 Meta box

Box type: 'meta'

Container: File, Movie box ('moov'), or Track box ('trak')

Mandatory: No

Quantity: Zero or one at each file level, movie level, or track level

The container for the Meta (meta) box is an F4V file, a Movie (moov) box, or a Track (trak) box.

The meta box can contain a variety of other boxes that carry metadata.

meta box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'meta' (0x6D657461)
Version	UI8	Reserved, set to 0
Flags	UI24	Reserved, set to 0
Boxes	BOX [ ]	Arbitrary number of boxes that define the file's metadata

For more information, see section 8.11.1 of ISO/IEC 14496-12:2008.

2.15 Free Space boxes

Box type: 'free' or 'skip'

Container: File or any box

Mandatory: No

Quantity: Any

The contents of the Free (free) and the Skip (skip) boxes are free file space and the player shall ignore their contents. The boxes may be used wherever boxes are permitted. The boxes can reserve space for future expansion of data in the container boxes.

free space box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'free' (0x66726565) or 'skip' (0x736b6970)
Void	UI8 [ ]	Arbitrary number of bytes to end of box

2.16 Movie Fragment Random Access box

Box type: 'mfra'

Container: File

Mandatory: No

Quantity: One

The Movie Fragment Random Access (mfra) box assists in finding random access points in a fragmented F4V file by providing Track Fragment Random Access (tfra) boxes for tracks (not necessarily for all the tracks). The information in this box is not definitive, and provides only a hint to random access points.

The mfra box should be last in the file. The last box within the mfra box provides a copy of the length field from the mfra box.

mfra box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mfra' (0x6D667261)
Boxes	BOX [ ]	Arbitrary number of boxes that define the random access points

For more information, see section 8.8.9 of ISO/IEC 14496-12:2008.

2.16.1 Track Fragment Random Access box

Box type: 'tfra'

Container: Movie Fragment Random Access box ('mfra')

Mandatory: No

Quantity: Zero or more

Each Track Fragment Random Access (tfra) box entry provides the location and the presentation time of a random accessible sample. The tfra box does not need to contain an entry for each random accessible sample in the track. The absence of this box does not mean that all the samples are sync samples.

tfra box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'tfra' (0x74667261)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved, set to 0
TrackID	UI32	Identifies the track
Reserved	UI26	Reserved. Set to 0
LengthSizeTrafNumMinus1	UI2	Length, in bytes, of the TrafNumber field in the RandomAccessStructure record, minus one
LengthSizeTrunNumMinus1	UI2	Length, in bytes, of the TrunNumber field in the RandomAccessStructure record, minus one
LengthSizeSampleNumMinus1	UI2	Length, in bytes, of the SampleNumber field in the RandomAccessStructure record, minus one
NumberEntry	UI32	Number of entries for this track. If 0, every sample is a random access point
RandomAccessSample	RandomAccessStructure [NumberEntry]	Position and presentation time of random access samples

Each RandomAccessStructure record has the following layout:

RandomAccessStructure

Field	Type	Comment
Time	IF Version == 0 UI32 IF Version == 1	Presentation time of the random access sample, in TimeScale units defined in the Media Header for this track

	UI64	
MoofOffset	IF Version == 0 UI32 IF Version == 1 UI64	The byte-offset of the corresponding Movie Fragment box, from the beginning of the file
TrafNumber	{ UI8, UI16, UI24, UI32 } [LengthSizeTrafNumMinus1]	Traf number containing the random accessible sample. The first traf in each moof is numbered 1. Type is one of UI8, UI16, UI24, UI32 indexed by LengthSizeTrafNumMinus1
TrunNumber	{ UI8, UI16, UI24, UI32 } [LengthSizeTrunNumMinus1]	Trun number containing the random accessible sample. The first trun in each traf is numbered 1.
SampleNumber	{ UI8, UI16, UI24, UI32 } [LengthSizeSampleNumMinus1]	Sample number containing the random accessible sample. The first sample in each trun is numbered 1.

For more information, see section 8.8.10 of ISO/IEC 14496-12:2008.

2.16.2 Movie Fragment Random Access Offset box

Box type: 'mfro'

Container: Movie Fragment Random Access box ('mfra')

Mandatory: Yes

Quantity: One

The Movie Fragment Random Access Offset (mfro) box provides a copy of the length field of the Movie Fragment Random Access (mfra) box and assists in finding the mfra box. The mfro box shall be placed last within the mfra box.

mfro box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'mfro' (0x6d66726f)
Version	UI8	Either 0 or 1
Flags	UI24	Reserved, set to 0
Size	UI32	Size of enclosing Movie Fragment Random Access box, in bytes

For more information, see section 8.8.11 of ISO/IEC 14496-12:2008.

3 F4V Metadata

This section describes the metadata supported by the F4V file format

3.1 Tag box

Box types: 'auth', 'titl', 'dscp' and 'cprt'

Container: Movie box ('moov')

Mandatory: No

Quantity: Zero or one of each type.

The F4V file format supports four optional tag boxes contained within a Movie (moov) box. An F4V file may contain up to 256 tags (including the tags in these boxes and the tags defined in an ilst box).

Tag box

Field	Type	Comment
Header	BOXHEADER	BoxType shall be one of the following: 'auth' (0x61757468) for Author 'titl' (0x7469746C) for Title 'dscp' (0x64736370) for Description 'cprt' (0x63707274) for Copyright
Version	UI8	Shall be 0
Flags	UI24	Reserved, set to 0
Pad	UI1	Padding, set to 0
Language	UI5 [3]	3-character code specifying language (see ISO 639-2/T). Each character is interpreted as 0x60 + (5 bit) code to yield an ASCII character.
TagString	UI8 [ ]	Tag string data, occupying the remainder of the box. The TagString length shall not exceed 65535 bytes

3.2 XMP Metadata box

Box type: 'uuid'

Container: File

Mandatory: No

Quantity: One

Beginning in version 10, Flash Player can load XMP data embedded in an F4V file. XMP is Adobe's Extensible Metadata Platform. For more information, see www.adobe.com/go/xmp.

The XMP Metadata box shall immediately follow the Movie (moov) box, with no intervening boxes. The size of the XMP Metadata box shall not exceed 64 megabytes.

With the XMP Metadata box, the file can communicate XMP metadata to a SWF movie via ActionScript. The XMPMetadata is exposed to ActionScript via a STRING property named `data`.

XMP Metadata box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'uuid' (0x75756964)
UUID	UI8 [16]	16-byte (128-bit) universally unique identifier (UUID). The UUID shall be the hexadecimal bytes BE 7A CF CB 97 A9 42 E8 9C 71 99 94 91 E3 AF AC.

XMPMetadata	UI8 [ ]	XMP metadata, formatted according to the XMP metadata standard

3.3 ilst box

Box type: 'ilst'

Container: Meta box ('meta')

Mandatory: No

Quantity: One

An ilst box occurs inside a Meta (meta) box and contains an arbitrary number of metadata tags. An F4V file may contain up to 256 tags (including the tags in this box and in the 'auth', 'titl', 'dscp', and 'cprt' boxes).

ilst box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'ilst' (0x696C7374)
TagCount	UI32	The number of tags enumerated in the ilst box
Tags	TAGRECORD [TagCount]	A number of TAGRECORD entries

Each TAGRECORD has the following layout:

TAGRECORD

Field	Type	Comment
TagLength	UI32	The total length of the TAGRECORD, including this length field
TagName	UI8 [4]	4 bytes indicating the name of the tag. These bytes usually come from the human-readable ASCII set, but not always
DataLength	UI32	The total length of the data portion of the TAGRECORD
DataTag	UI8 [4]	The 4 bytes 'd', 'a', 't', and 'a' to indicate the data portion of the TAGRECORD
DataType	UI32	Specifies the type of data in the data payload of the TAGRECORD
Reserved	UI32	Reserved, set to 0
Payload	UI8 [ ]	An arbitrary number of bytes occupying the remainder of the TAGRECORD. The precise payload format is dependent on the DataType

The supported values for the DataType are:

- 0: custom data. In the case of 'trkn' and 'disk' tag types, the data payload is interpreted as a single UI32
- 1: text data
- 13, 14: binary data
- 21: generic data

3.4 Text Track Metadata

Box type: See below

Container: Text samples ('text' or 'tx3g') in Media Data box ('mdat')

Mandatory: No

Quantity: Any

Text samples ('text' or 'tx3g') can contain the following metadata boxes. Their contents are exposed to a running ActionScript program through the `onTextData` property.

3.4.1 Style box

A Style (styl) box carries text style specifications. This information is exposed to ActionScript via the `style` property.

styl box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'styl' (0x7374796C)
Count	UI16	The number of entries in the Styles array
Styles	STYLERECORD [Count]	An array of STYLERECORD structures, each exposed as an ActionScript object

An individual STYLERECORD has the following layout:

STYLERECORD

Field	Type	Comment
StartChar	UI16	The first character to which this STYLERECORD applies, exposed to ActionScript via a DOUBLE property named `startchar`
EndChar	UI16	The last character to which this STYLERECORD applies, exposed to ActionScript via a DOUBLE property named `endchar`
FontID	UI16	The font ID to use for this style, exposed to ActionScript via a DOUBLE property named `fontid`
FaceStyleFlags	UI8	Exposed to ActionScript via a DOUBLE property named `facestyleflags`
FontSize	UI8	The size to use for the font, exposed to ActionScript via the property `fontsize`
TextColor	UI32	The RGBA color for the text, exposed to ActionScript via the property `textcolor`

3.4.2 Highlight box

A highlight (hlit) box specifies a range of text to be highlighting. This information is exposed to ActionScript via the `highlight` property.

hlit box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'hlit' (0x686C6974)
StartChar	UI16	The first character to highlight, exposed to ActionScript via a DOUBLE property named `startchar`
EndChar	UI16	The final character to highlight, exposed to ActionScript via a DOUBLE property named `endchar`

3.4.3 Highlight Color box

A Highlight Color (hclr) box specifies the highlight color for text. This information is exposed to ActionScript via the `highlightcolor` property.

hclr box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'hclr' (0x68636C72)
HighlightColor	UI16 [3]	A three-element array that holds values for red, green, and blue components in that order, exposed to ActionScript via a DOUBLE property named `highlightcolor`

3.4.4 Karaoke box

A Karaoke (krok) box specifies karaoke metadata. This information is exposed to ActionScript via the `karaoke` property. Times are expressed in TimeScale units as defined for the track.

krok box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'krok' (0x6B726F6B)
StartTime	UI32	Exposed to ActionScript via a DOUBLE property named `starttime`
Count	UI16	The number of entries in the KaraokeRecords array
KaraokeRecords	KARAOKEREC [Count]	An array of KARAOKEREC structures, each exposed to ActionScript as an object

An individual KARAOKEREC has the following structure:

KARAOKEREC

Field	Type	Comment
EndTime	UI32	Exposed to ActionScript via a DOUBLE property named `endtime`
StartChar	UI16	Exposed to ActionScript via a DOUBLE property named `startchar`
EndChar	UI16	Exposed to ActionScript via a DOUBLE property named `endchar`

3.4.5 Scroll Delay box

A Scroll Delay (dlay) box specifies a scroll delay. This information is exposed to ActionScript via the `scrolldelay` property, expressed in TimeScale units in relation to the track.

dlay box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'dlay' (0x646C6179)
ScrollDelay	UI32	Exposed to ActionScript via a DOUBLE property named `scrolldelay`

3.4.6 Drop Shadow Offset box

A Drop Shadow (drpo) box specifies drop shadow offset coordinates for text.

drpo box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'drpo' (0x6472706F)
DropShadowOffsetX	UI16	Exposed to ActionScript via a DOUBLE property named `dropshadowoffsetx`
DropShadowOffsetY	UI16	Exposed to ActionScript via a DOUBLE property named `dropshadowoffsety`

3.4.7 Drop Shadow Alpha box

A Drop Shadow Alpha (drpt) box specifies drop shadow alpha value.

drpt box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'drpt' (0x64727074)
DropShadowAlpha	UI16	A 16-bit alpha value, exposed to ActionScript via a DOUBLE property named `dropshadowalpha`

3.4.8 Hypertext box

A Hypertext box (href) specifies a hypertext link with ALT text over a text range. This information is exposed to ActionScript via the `hypertext` property.

href box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'href' (0x68726566)
StartChar	UI16	The beginning character of the text range, exposed to ActionScript via a DOUBLE property named `startchar`
EndChar	UI16	The last character of the text range, exposed to ActionScript via a DOUBLE property named `endchar`
URLSize	UI8	The length of the URL string
URL	UI8 [URLSize]	The URL string, exposed to ActionScript via a STRING property named `url`
ALTSize	UI8	The length of the ALT string
ALT	UI8 [ALTSize]	The ALT string which is displayed when the user's mouse hovers over the link, exposed to ActionScript via a STRING property named `alt`

3.4.9 Text Box box

A Text Box (tbox) box defines the coordinates for a text box. This information is exposed to ActionScript via the `textbox` property.

tbox box

Field	Type	Comment
Header	BOXHEADER	BoxType = 'tbox' (0x74626F78)

Top	UI16	The top pixel coordinate, exposed to ActionScript via a DOUBLE property named `top`
Left	UI16	The left pixel coordinate, exposed to ActionScript via a DOUBLE property named `left`
Bottom	UI16	The bottom pixel coordinate, exposed to ActionScript via a DOUBLE property named `bottom`
Right	UI16	The right pixel coordinate, exposed to ActionScript via a DOUBLE property named `right`

3.4.10 Blinking box

A Blinking (blnk) box specifies a range of text to set blinking. This information is exposed to ActionScript via the `blink` property.

blnk box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'blnk' (0x626C6E6B)
StartChar	UI16	The first character in the blinking range, exposed to ActionScript via a DOUBLE property named `startchar`
EndChar	UI16	The ending character in the blinking range, exposed to ActionScript via a DOUBLE property named `endchar`

3.4.11 Text Wrap box

A Text Wrap (twrp) box sets the wrap flag for text.

twrp box		
Field	**Type**	**Comment**
Header	BOXHEADER	BoxType = 'twrp' (0x74777270)
WrapFlag	UI8	Boolean that is nonzero if the text should wrap, exposed to ActionScript via a DOUBLE property named `wrapflag`

Annex A. Embedding Cue Points

A.1 Overview

The ability to mix media playback and rich interactive data is a key benefit of using Flash to build media applications. F4V applications can support the embedding of temporal data (cue points) in the form of AMF samples. These samples are transmitted to the Flash runtime along with audio and video samples, where they are dispatched to the application script.

A.2 The AMF Sample Format

An AMF sample is an AMF object containing a list of typed AMF values.

The AMF object shall be either an AMF0 object, or an AMF3 object, according to the type specified for the data track. The specifications for AMF0 and AMF3 can be found at:

http://opensource.adobe.com/wiki/display/blazeds/Developer+Documentation

The first value shall be a string that represents the name of the AMF sample. The AMF values will be dispatched to a method with this name. For example, if the first field is a string called "onFoo", then the method "onFoo" is called when the AMF sample is played.

Table 3 lists names that are reserved for the runtime, and that are not dispatched to the script:

Table 3. Reserved names

attachAudio	attachVideo	call	close	getBufferInfo
onStatus	pause	play	play2	publish
receiveAudio	receiveVideo	seek	send	setBufferTime

A.3 The AMF Data Track Structure

AMF samples can be stored in a data track. The data track shall be configured as follows:

The HandlerType in the Handler Reference (hdlr) box shall be 'data'

The Media Header box type shall be 'nmhd'

The Sample Description (stsd) box shall contain one description record describing AMF samples.

The description entry format shall be the SampleEntry type.

The description entry's box type shall be 'amf0' or 'amf3' corresponding to the samples' AMF format.

The following boxes of the data track shall contain an entry for each AMF sample:

- Decoding Time To Sample (stts) box, for the decoding time of the AMF sample.
- Sample Size (stsz) box, for the size of the AMF sample.
- Chunk Offset (stco or co64) box, for the offset of the AMF sample.
- Composition Time to Sample (ctts) box, for the time of passing the AMF sample to the ActionScript program.

Within the Media Data (mdat) box, the samples in the data track should be interleaved with the audio and video samples.

A.3.1 Decoding The Data Track

While playing an F4V file, the AMF samples inside the mdat box are passed to the AMF decoder.

At the time stated in the stts box, the AMF decoder decodes the sample.

At the time stated in the ctts box, the AMF decoder passes the decoded AMF sample to the ActionScript program.

A.4 Progressive Download

With proper interleaving, this method is suitable for progressive download.

The AMF content should be interleaved at the right time along with the audio and video content, to ensure that the data is available at the right time as the file is being downloaded.

The AMF data should not be stored at the end of the file, as, in such case, the entire file would have to be downloaded before the first AMF sample could be dispatched, even if that sample were to occur temporally very early in the content.

A.5 Multiple Data Tracks

There shall be only one data track.

Annex B. Flash Player Metadata

B.1 Stream Properties

When Flash Player loads an F4V file, various stream properties are made available to a running ActionScript program via the `NetStream.onMetaData` property. The available properties differ depending on the software used to create the file. Typical properties are:

- `audiocodecid`: a STRING with four characters that define the audio codec used, if audio is present and is encoded with a codec that Flash Player can decode
- `avclevel`: a DOUBLE indicating the AVC level that the video conforms to, if video is present and is encoded with AVC/H.264
- `avcprofile`: a DOUBLE indicating the AVC profile that the video conforms to, if video is present and is encoded with AVC/H.264
- `duration`: a DOUBLE indicating the total length of the movie in seconds
- `height`: a DOUBLE indicating the height of the video, if video is present and is encoded with a codec that Flash Player can decode
- `moovposition`: a DOUBLE indicating the absolute offset of the moov box within the F4V file. This property is useful for determining if the file will load progressively
- `videocodecid`: a STRING with four characters that define the video codec used, if video is present and is encoded with a codec that Flash Player can decode
- `videoframerate`: a DOUBLE indicating the average video frame rate of the video, if video is present and is encoded with a codec that Flash Player can decode
- `width`: a DOUBLE indicating the width of the video, if video is present and is encoded with a codec that Flash Player can decode

B.2 Image Metadata

If an F4V sample is an image type (GIF, PNG, or JPEG), the data is made available to a running ActionScript program through the `onImageData` property. The following properties are present:

- `data`: a BYTEARRAY containing the compressed image data (that is, the original JPEG, PNG or GIF file data)
- `trackid`: a DOUBLE indicating the track that this sample belongs to

Annex C. HTTP Streaming: File Structure

C.1 Overview

Flash Player supports HTTP streaming with F4V fragments. An HTTP streaming presentation is composed of an HTTP streaming manifest file (F4M) and HTTP streaming segments (Fragmented F4V files, or F4Fs).

The presentation is divided along the time line into HTTP streaming segments, further divided into HTTP streaming fragments. The presentation may be available in parallel at multiple quality levels. The presentation can be cached and delivered by fragment (fully or partially) and quality level.

The presentation's Bootstrap Info box specifies the data structure of the presentation and access to it.

The manifest file (F4M) is further described in
http://opensource.adobe.com/wiki/display/osmf/Flash+Media+Manifest+File+Format+Specification

C.2 HTTP Streaming Segment

An HTTP streaming segment is a complete F4V file containing fragments. The segment may belong to only one quality level. The segment shall comprise of a set of boxes followed by a set of HTTP streaming fragments.

The set of boxes shall include the following boxes, preferably in this order (optional boxes are indicated by brackets []):

- ftyp
- [afra]
- [abst]
- moov
- rtmp
- [mdat]

The set of boxes may include afra and mdat boxes. If included, the afra box shall be located before the mdat box and the abst box.

A moof box outside the HTTP streaming fragments in an HTTP streaming segment shall not be used for HTTP streaming.

C.3 HTTP Streaming Fragment

An HTTP streaming fragment shall include one of each of the following boxes, preferably in this order:

- afra
- abst
- moof
- mdat

The afra box shall be located before all other boxes.

The fragment is not a complete F4V file. The boxes ftyp, pdin, and moov are not allowed in an HTTP streaming fragment.

C.4 URL Construction

Each HTTP streaming segment is a separate URL resource (file). A URL scheme can address each HTTP streaming fragment uniquely. The URL for an HTTP streaming fragment shall be constructed as follows:

```
http://<ServerBaseUrl>/<MovieIdentifier><QualitySegmentUrlModifier>Seg<SegmentNumber>-
Frag<FragmentNumber>
```

where the F4V specification defines the fields in brackets and numbers have no leading zeroes.

If ServerEntryCount == 0, <ServerBaseUrl> and the trailing slash shall be omitted.

If QualityEntryCount == 0, <QualitySegmentUrlModifier> shall be omitted.

EXAMPLE: `http://adobe.com/MyMovie/highSeg1-Frag210`

C.5 Adobe Multiplexed Hint Track Format

F4V files that are intended for HTTP streaming need to include a hint track. The hint track provides information that enables the streaming server to create transmission packets. See the ISO spec, section 7, for information on streaming.

Adobe supports the Adobe Multiplexed Hint Track format. The Adobe Multiplexed Hint Track format is flexible enough to support RTMP packets as well as a format like FLV, wherein the samples as a whole are interleaved in time order. FLV compatibility mode is defined for efficient mapping between the hint samples and the FLV format. The format can be configured to ensure that the mdat consisting of a series of hint samples would be identical to a portion of an FLV corresponding to those samples.

The Adobe Mux Hint Sample Entry box (rtmp) describes the hint track. The hint track contains a collection of AdobeMuxHintSamples and is in the Media Data box.

Three hinting modes are defined:

- The immediate Mode (Mode = 0)

  In this mode, the payload of the multiplexed track is available directly in the hint sample itself for efficiency. However, this mode has to be carefully used, as it can lead to some duplication in data.

- The immediate noDuplication Mode (Mode = 1)

  The immediate noDuplication mode is defined to avoid the duplication of data in the immediate mode. In this configuration, the offsets in the sample tables of the original (audio/video) tracks are adjusted to physically point to the (immediate mode) hint samples that contain the media data (locally in the mdat location). Because the chunk offsets are changed to point to the hint samples (after the header and where the sample starts) for every sample, all chunks contain only one sample.

  Therefore, the media box "mdat" will just contain the hint samples (no audio or video samples). The hint samples will have the media data embedded in its immediate data field. This is possible only when full samples are used in the Adobe Multiplexed packets (not in the chunked RTMP mode). In addition, this has the side effect of growing the chunk offset table, but it is minimal compared to the efficiency gained.

- The sample Mode (Mode = 2)

  In this mode, the Packet headers and trailers are defined to be part of the hint sample. The payload of the multiplexed track is "pointed to" from the hint sample into the media tracks by referring to a particular sample in the media track with a length and an offset.

Annex D. F4V Encryption

D.1 Overview

This section provides an overview of how Adobe's DRM key/rights management system is used to protect media within F4V. The protection applies only to the audio/video tracks of F4V. The encryption format for other track types (e.g. AMF tracks) is out of scope for this specification.

This section should be read in conjunction with ISO 14496-12:2008 Section 8.12 (Support for Protected Streams). It is critical that the reader understand the above file format before reading the next sections.

D.2 The Encryption Process

The encryption process changes the sample formats from plaintext to cipher text. An Adobe DRM Access Unit Header is inserted before each sample data. Because of converting plain text to cipher text, the underlying media cannot be accessed by consuming application without the appropriate keys.

The sample description entry in the sample description table that describes the encrypted samples is transformed. The transformed structure follows ISO 14496-12:2008 Section 8.12. The purpose of the transformation of the sample description entry is twofold: the sample description entry prevents accidental treatment of encrypted data as if it was unencrypted and documents the transforms applied.

The following sample description transformations are carried out:

- The 4CC of the sample description entry is replaced with a 4CC indicating the encryption:
    - 'encv' for an encrypted video stream (instead of e.g. 'mp4v', 'avc1'),
    - 'enca' for an encrypted audio stream (instead of e.g. 'mp4a', 'samr'),
    - 'encr' for an encrypted data stream.
- A Protection Scheme Info (sinf) box is appended to the sample description entry, leaving all other boxes unmodified. The sinf box contains all the information required both to understand the encryption transform applied and its parameters, and to find other information such as the kind and location of the key management system. It also documents the original (unencrypted) format of the media.
- The original format 4CC of the sample description entry is stored in the Original Format (frma) box , which is a sub-box of the sinf box.
- The Scheme Type (schm) box is also a sub-box of the sinf box and specifies the encryption scheme as 4CC and its version. In F4V file, this SchemeType 4CC shall be 'adkm', Adobe's DRM Key Management.
- In the sinf box, there is space for a "black box" (Scheme Information (schi) box) describing the parameters to the key management governing access to the encrypted media content. The schi box is a container box that is interpreted only by the scheme being used. In F4V file, this box shall be the Adobe DRM Key Management System box.

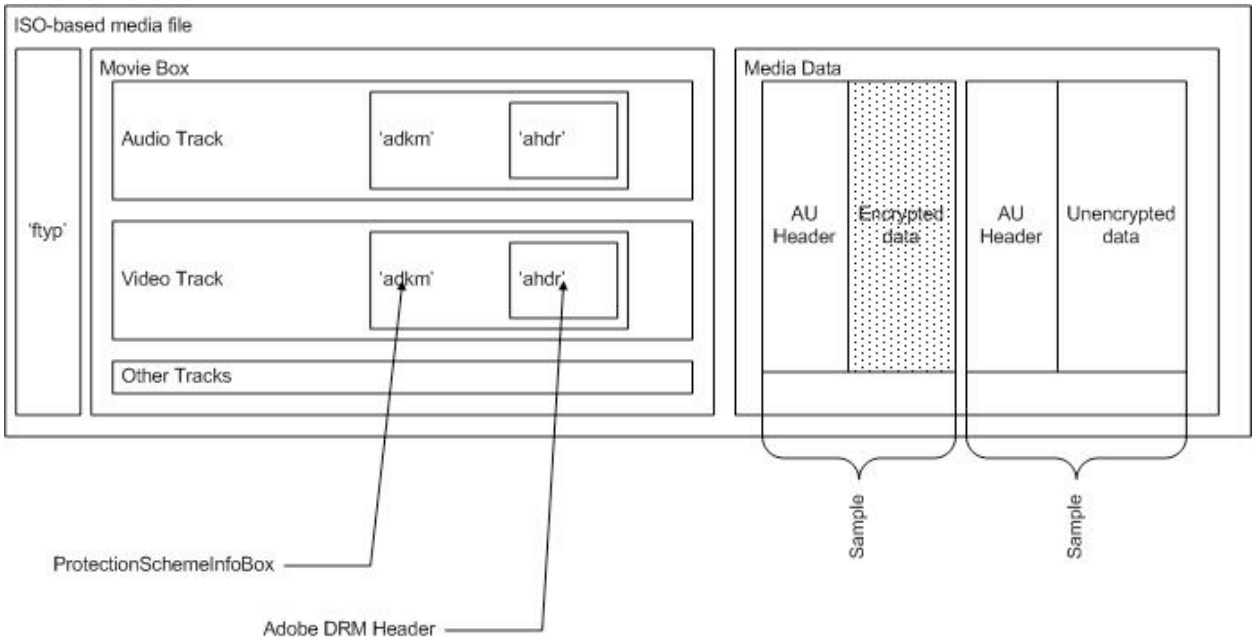

Figure 1 – Example on storing the protection information in F4V

Figure 1 illustrates how the protection information is stored in F4V. In the example, placing a Protection Scheme Info (sinf) box into each track's sample description entry, and specifying Adobe's DRM identifier as the key/rights management system, protects the audio and video tracks.

The sinf box is per sample entry in a sample description box. While it is possible to have more than one sample entry within a sample description box (there can be only one sample description box per track), this is not very common. Hence, the above diagram only shows one sinf box per track. However, as it is possible to have more than one per track, both the DRM packager and DRM decoder should be able to handle the case.

D.3 Encryption of Samples

This section describes how each sample is transformed when applying DRM to an audio or video track within a F4V.

D.3.1 Access Unit Header

The Access Unit Header, defined in Table 4, specifies the format for each sample unit protected by Adobe's DRM. A media file format specifies the layout of the media data as samples, but the encryption/decryption process requires additional information carried in each sample. The additional information is dependent on the DRM key management used. Adobe's DRM specifies its own access unit header, which shall precede each codec-specific sample data. The F4V Access Unit Header is identical to the FLV Selective Encryption Filter Params.

Table 4. Access Unit Header

Access Unit Header		
Field	**Type**	**Comment**
EncryptedAU	UI1	Selective Encryption indicator shows if the packet is encrypted. 0 = sample is not encrypted 1 = sample is encrypted.
Reserved	UI7	Shall be 0
IV	IF EncryptedAU == 1 UI8 [IVLength]	Only present if the sample is encrypted. Contains 16 bytes of IV data for AES-CBC

The above specified access unit header shall be added to every sample whose sample description entry has DRM turned on (i.e. has a Protection Scheme Info (sinf) box present), even when the particular sample is not encrypted. The header is the only way the decoder knows whether a particular sample is encrypted or not (in case SelectiveEncryption is 1). Selective encryption of samples can improve performance, when only critical Keyframes are encrypted.

D.3.2 Padding Of Encrypted Samples

All encrypted samples shall be padded to a multiple of the block cipher's block length.

The padding scheme shall be as described in RFC 2630, which is reproduced here:

Block ciphers expect the input data to be multiple of k octets (in case of AES 128, require it to be a multiple of 16 octets), where k is greater than 1. For such algorithms, the input shall be padded at the trailing end with k-(length mod k) octets all having the value k-(length mod k), where length is the length of the input.

Thus, the input is padded at the trailing end with one of the following k byte sequences, as shown in Table 5.

Table 5. Padding the cipher block

Condition	Bytes added to the end of the block						
IF length mod k = k-1							01
IF length mod k = k-2						02	02
…							
IF length mod k = n					…	k-n	k-n
…							
IF length mod k = 0	k	k	…	k	k	k	k

The size of the padding can be determined unambiguously from a padded block since all the input is padded, including input values that are already a multiple of the block size, and no padding sequence is a suffix of another. The last octet indicates how many octets to trim.

Annex E. The FLV File Format

E.1 Overview

Each tag type in an FLV file constitutes a single stream. There shall be no more than one audio and one video stream, synchronized together, in an FLV file. An FLV file shall not define multiple independent streams of a single type.

The simple data types used in FLV are defined in the SWF format specification. FLV files use an additional type that is not defined for SWF files: UI24 representing an unsigned 24-bit integer.

Unlike SWF files, FLV files shall store multi-byte numbers in big-endian byte order. For example, as a UI16 in SWF file format, the byte sequence that represents the number 300 (0x12C) is 0x2C 0x01; as a UI16 in FLV file format, the byte sequence that represents the number 300 is 0x01 0x2C.

See also the SWF File Format Specification at http://www.adobe.com/go/swf_file_format

E.2 The FLV header

An FLV file shall begin with the FLV header:

FLV header

Field	Type	Comment
Signature	UI8	Signature byte always 'F' (0x46)
Signature	UI8	Signature byte always 'L' (0x4C)
Signature	UI8	Signature byte always 'V' (0x56)
Version	UI8	File version (for example, 0x01 for FLV version 1)
TypeFlagsReserved	UB [5]	Shall be 0
TypeFlagsAudio	UB [1]	1 = Audio tags are present
TypeFlagsReserved	UB [1]	Shall be 0
TypeFlagsVideo	UB [1]	1 = Video tags are present
DataOffset	UI32	The length of this header in bytes

The DataOffset field usually has a value of 9 for FLV version 1. This field is present to accommodate larger headers in future versions.

E.3 The FLV File Body

After the FLV header, the remainder of an FLV file shall consist of alternating back-pointers and tags. They interleave as shown in the following table:

FLV File Body

Field	Type	Comment
PreviousTagSize0	UI32	Always 0
Tag1	FLVTAG	First tag
PreviousTagSize1	UI32	Size of previous tag, including its header, in bytes. For FLV version 1, this value is 11 plus the DataSize of the previous tag.
Tag2	FLVTAG	Second tag
...		
PreviousTagSizeN-1	UI32	Size of second-to-last tag, including its header, in bytes.

TagN	FLVTAG	Last tag
PreviousTagSizeN	UI32	Size of last tag, including its header, in bytes.

E.4 FLV Tag Definition

E.4.1 FLV Tag

The FLV tag contains metadata for audio, video, or scripts, optional encryption metadata, and the payload.

FLVTAG

Field	Type	Comment
Reserved	UB [2]	Reserved for FMS, should be 0
Filter	UB [1]	Indicates if packets are filtered. 0 = No pre-processing required. 1 = Pre-processing (such as decryption) of the packet is required before it can be rendered. Shall be 0 in unencrypted files, and 1 for encrypted tags. See Annex F. FLV Encryption for the use of filters.
TagType	UB [5]	Type of contents in this tag. The following types are defined: 8 = audio 9 = video 18 = script data
DataSize	UI24	Length of the message. Number of bytes after StreamID to end of tag (Equal to length of the tag – 11)
Timestamp	UI24	Time in milliseconds at which the data in this tag applies. This value is relative to the first tag in the FLV file, which always has a timestamp of 0.
TimestampExtended	UI8	Extension of the Timestamp field to form a SI32 value. This field represents the upper 8 bits, while the previous Timestamp field represents the lower 24 bits of the time in milliseconds.
StreamID	UI24	Always 0.
AudioTagHeader	IF TagType == 8 AudioTagHeader	AudioTagHeader element as defined in Section E.4.2.1.
VideoTagHeader	IF TagType == 9 VideoTagHeader	VideoTagHeader element as defined in Section E.4.3.1.
EncryptionHeader	IF Filter == 1 EncryptionTagHeader	Encryption header shall be included for each protected sample, as defined in Section F.3.1.
FilterParams	IF Filter == 1 FilterParams	FilterParams shall be included for each protected sample, as defined in Section F.3.2.

Data	IF TagType == 8 AUDIODATA IF TagType == 9 VIDEODATA IF TagType == 18 SCRIPTDATA	Data specific for each media type.

In playback, the time sequencing of FLV tags depends on the FLV timestamps only. Any timing mechanisms built into the payload data format shall be ignored.

E.4.2 Audio Tags

Audio tags are similar to the DefineSound tag in the SWF file format. For formats also supported in SWF, the payload data is identical in FLV and SWF.

E.4.2.1 AUDIODATA

The AudioTagHeader contains audio-specific metadata.

AudioTagHeader

Field	Type	Comment
SoundFormat (See notes following table, for special encodings)	UB [4]	Format of SoundData. The following values are defined: 0 = Linear PCM, platform endian 1 = ADPCM 2 = MP3 3 = Linear PCM, little endian 4 = Nellymoser 16 kHz mono 5 = Nellymoser 8 kHz mono 6 = Nellymoser 7 = G.711 A-law logarithmic PCM 8 = G.711 mu-law logarithmic PCM 9 = reserved 10 = AAC 11 = Speex 14 = MP3 8 kHz 15 = Device-specific sound Formats 7, 8, 14, and 15 are reserved. AAC is supported in Flash Player 9,0,115,0 and higher. Speex is supported in Flash Player 10 and higher.
SoundRate	UB [2]	Sampling rate. The following values are defined: 0 = 5.5 kHz 1 = 11 kHz 2 = 22 kHz 3 = 44 kHz
SoundSize	UB [1]	Size of each audio sample. This parameter only pertains to uncompressed formats. Compressed formats always decode to 16 bits internally. 0 = 8-bit samples 1 = 16-bit samples

SoundType	UB [1]	Mono or stereo sound 0 = Mono sound 1 = Stereo sound
AACPacketType	IF SoundFormat == 10 UI8	The following values are defined: 0 = AAC sequence header 1 = AAC raw

Format 3, linear PCM, stores raw PCM samples. If the data is 8-bit, the samples are unsigned bytes. If the data is 16-bit, the samples are stored as little endian, signed numbers. If the data is stereo, left and right samples are stored interleaved: left - right - left - right - and so on.

Format 0 PCM is the same as format 3 PCM, except that format 0 stores 16-bit PCM samples in the endian order of the platform on which the file was created. For this reason, format 0 should not be used.

Nellymoser 8 kHz and 16 kHz are special cases, as the SoundRate field cannot represent 8 or 16 kHz sampling rates. When Nellymoser 8 kHz or Nellymoser 16 kHz is specified in SoundFormat, the Flash Player ignores the SoundRate and SoundType fields. For other Nellymoser sampling rates, specify the normal Nellymoser SoundFormat and use the SoundRate and SoundType fields as usual.

If the SoundFormat indicates AAC, the SoundType should be 1 (stereo) and the SoundRate should be 3 (44 kHz). However, this does not mean that AAC audio in FLV is always stereo, 44 kHz data. Instead, the Flash Player ignores these values and extracts the channel and sample rate data is encoded in the AAC bit stream.

If the SoundFormat indicates Speex, the audio is compressed mono sampled at 16 kHz, the SoundRate shall be 0, the SoundSize shall be 1, and the SoundType shall be 0. For information regarding Speex capabilities and limitations when stored in a SWF file, see the SWF File Format Specification at http://www.adobe.com/go/swf_file_format.

The AUDIODATA segment contains the audio payload.

AUDIODATA

Field	Type	Comment
IF Encrypted		See Annex F. FLV Encryption for details.
Body	EncryptedBody	AudioTagBody encrypted as specified in Section F.3.3.
ELSE		
Body	AudioTagBody	

The AudioTagBody holds the audio payload.

AudioTagBody

Field	Type	Comment
SoundData	IF SoundFormat == 10 AACAUDIODATA ELSE Varies by format	

E.4.2.2 AACAUDIODATA

The AAC format is supported in Flash Player 9,0,115,0 and higher.

AACAUDIODATA

Field	Type	Comment
Data	IF AACPacketType == 0	The AudioSpecificConfig is defined in ISO

	AudioSpecificConfig ELSE IF AACPacketType == 1 Raw AAC frame data in UI8 [ ]	14496-3. Note that this is not the same as the contents of the esds box from an MP4/F4V file.

E.4.3 Video Tags

Video tags are similar to the VideoFrame tag in the SWF file format, and their payload data is identical. See also the SWF File Format Specification at http://www.adobe.com/go/swf_file_format

E.4.3.1 VIDEODATA

The VideoTagHeader contains video-specific metadata.

VideoTagHeader

Field	Type	Comment
Frame Type	UB [4]	Type of video frame. The following values are defined: 1 = key frame (for AVC, a seekable frame) 2 = inter frame (for AVC, a non-seekable frame) 3 = disposable inter frame (H.263 only) 4 = generated key frame (reserved for server use only) 5 = video info/command frame
CodecID	UB [4]	Codec Identifier. The following values are defined: 2 = Sorenson H.263 3 = Screen video 4 = On2 VP6 5 = On2 VP6 with alpha channel 6 = Screen video version 2 7 = AVC
AVCPacketType	IF CodecID == 7 UI8	The following values are defined: 0 = AVC sequence header 1 = AVC NALU 2 = AVC end of sequence (lower level NALU sequence ender is not required or supported)
CompositionTime	IF CodecID == 7 SI24	IF AVCPacketType == 1 Composition time offset ELSE 0 See ISO 14496-12, 8.15.3 for an explanation of composition times. The offset in an FLV file is always in milliseconds.

The VIDEODATA segment contains video metadata, optional encryption metadata, and the video payload.

VIDEODATA		
Field	**Type**	**Comment**
IF Encrypted		See Annex F. FLV Encryption for details.
Body	EncryptedBody	VideoTagBody encrypted as specified in Section F.3.3.
ELSE		
Body	VideoTagBody	

The VideoTagBody contains the video frame payload.

VideoTagBody		
Field	**Type**	**Comment**
VideoTagBody	IF FrameType == 5 UI8 ELSE (IF CodecID == 2 H263VIDEOPACKET IF CodecID == 3 SCREENVIDEOPACKET IF CodecID == 4 VP6FLVVIDEOPACKET IF CodecID == 5 VP6FLVALPHAVIDEOPACKET IF CodecID == 6 SCREENV2VIDEOPACKET IF CodecID == 7 AVCVIDEOPACKET)	Video frame payload or frame info If FrameType == 5, instead of a video payload, the Video Data Body contains a UI8 with the following meaning: 0 = Start of client-side seeking video frame sequence 1 = End of client-side seeking video frame sequence For all but AVCVIDEOPACKET, see the SWF File Format Specification for details

E.4.3.2 AVCVIDEOPACKET

An AVCVIDEOPACKET carries a payload of AVC video data.

AVCVIDEOPACKET		
Field	**Type**	**Comment**
Data	IF AVCPacketType == 0 AVCDecoderConfigurationRecord IF AVCPacketType == 1 One or more NALUs (Full frames are required)	

See ISO 14496-15, 5.2.4.1 for the description of AVCDecoderConfigurationRecord. This contains the same information that would be stored in an avcC box in an MP4/FLV file.

E.4.4 Data Tags

Data tags encapsulate single-method invocations, which usually are called on a NetStream object in Flash Player. Data tags comprise of a method name and a set of arguments.

E.4.4.1 SCRIPTDATA

The SCRIPTDATA segment contains optional encryption metadata, and the script payload.

SCRIPTDATA

Field	Type	Comment
IF Encrypted		See Annex F. FLV Encryption for details.
Body	EncryptedBody	ScriptTagBody encrypted as specified in Section F.3.3.
ELSE		
Body	ScriptTagBody	

The ScriptTagBody contains SCRIPTDATA encoded in the Action Message Format (AMF), which is a compact binary format used to serialize ActionScript object graphs. The specification for AMF0 is available at: http://opensource.adobe.com/wiki/display/blazeds/Developer+Documentation

ScriptTagBody

Field	Type	Comment
Name	SCRIPTDATAVALUE	Method or object name. SCRIPTDATAVALUE.Type = 2 (String)
Value	SCRIPTDATAVALUE	AMF arguments or object properties. SCRIPTDATAVALUE.Type = 8 (ECMA array)

E.4.4.2 SCRIPTDATAVALUE

A SCRIPTDATAVALUE record contains a typed ActionScript value.

SCRIPTDATAVALUE

Field	Type	Comment
Type	UI8	Type of the ScriptDataValue. The following types are defined: 0 = Number 1 = Boolean 2 = String 3 = Object 4 = MovieClip (reserved, not supported) 5 = Null 6 = Undefined 7 = Reference 8 = ECMA array 9 = Object end marker 10 = Strict array 11 = Date 12 = Long string

ScriptDataValue	IF Type == 0 DOUBLE IF Type == 1 UI8 IF Type == 2 SCRIPTDATASTRING IF Type == 3 SCRIPTDATAOBJECT IF Type == 7 UI16 IF Type == 8 SCRIPTDATAECMAARRAY IF Type == 10 SCRIPTDATASTRICTARRAY IF Type == 11 SCRIPTDATADATE IF Type == 12 SCRIPTDATALONGSTRING	Script data value. The Boolean value is (ScriptDataValue ≠ 0).

E.4.4.3 SCRIPTDATADATE

A SCRIPTDATADATE record stores date and time.

SCRIPTDATADATE

Field	Type	Comment
DateTime	DOUBLE	Number of milliseconds since Jan 1, 1970 UTC.
LocalDateTimeOffset	SI16	Local time offset in minutes from UTC. For time zones located west of Greenwich, UK, this value is a negative number. Time zones east of Greenwich, UK, are positive.

E.4.4.4 SCRIPTDATAECMAARRAY

A SCRIPTDATAECMAARRAY record stores an ECMA array. An ECMA Array is an associative array, and shall be used when an ActionScript Array contains non-ordinal indices. All indices, ordinal or otherwise, are strings instead of integers. For the purposes of serialization, this type is very similar to an anonymous ActionScript Object. The list contains approximately ECMAArrayLength number of items. A SCRIPTDATAOBJECTEND record follows the list of items.

SCRIPTDATAECMAARRAY

Field	Type	Comment
ECMAArrayLength	UI32	Approximate number of items in ECMA array
Variables	SCRIPTDATAOBJECTPROPERTY [ ]	List of variable names and values
List Terminator	SCRIPTDATAOBJECTEND	List terminator

E.4.4.5 SCRIPTDATALONGSTRING

SCRIPTDATASTRING and SCRIPTDATALONGSTRING records store strings.

SCRIPTDATALONGSTRING

Field	Type	Comment
StringLength	UI32	StringData length in bytes
StringData	STRING	String data, with no terminating NUL

E.4.4.6 SCRIPTDATAOBJECT

A SCRIPTDATAOBJECT record encodes the properties of an anonymous ActionScript object. A SCRIPTDATAOBJECTEND record follows the list of properties.

SCRIPTDATAOBJECT

Field	Type	Comment
ObjectProperties	SCRIPTDATAOBJECTPROPERTY [ ]	List of object properties
List Terminator	SCRIPTDATAOBJECTEND	List terminator

E.4.4.7 SCRIPTDATAOBJECTEND

The SCRIPTDATAOBJECTEND record terminates a list of SCRIPTDATAOBJECTPROPERTY records. The SCRIPTDATAOBJECTEND record is a SCRIPTDATAOBJECTPROPERTY record with a zero-length string and an Object end marker.

SCRIPTDATAOBJECTEND

Field	Type	Comment
ObjectEndMarker	UI8 [3]	Shall be 0, 0, 9

E.4.4.8 SCRIPTDATAOBJECTPROPERTY

A SCRIPTDATAOBJECTPROPERTY record defines an object property of an ActionScript object or a variable of associated array.

SCRIPTDATAOBJECTPROPERTY

Field	Type	Comment
PropertyName	SCRIPTDATASTRING	Name of the object property or variable
PropertyData	SCRIPTDATAVALUE	Value and type of the object property or variable

E.4.4.9 SCRIPTDATASTRICTARRAY

A SCRIPTDATASTRICTARRAY record stores a strict array. A strict array contains only ordinal indices, which are implied, not stored in the record. The indices can be dense or sparse. Undefined entries in the sparse regions between indices shall be serialized as undefined. The list shall contain StrictArrayLength number of values. No terminating record follows the list.

SCRIPTDATASTRICTARRAY

Field	Type	Comment
StrictArrayLength	UI32	Number of items in the array
StrictArrayValue	SCRIPTDATAVALUE [StrictArrayLength]	List of typed values

E.4.4.10 SCRIPTDATASTRING

SCRIPTDATASTRING and SCRIPTDATALONGSTRING records store strings.
The SCRIPTDATASTRING record may be used for strings no longer than 65535 characters.

SCRIPTDATASTRING

Field	Type	Comment
StringLength	UI16	StringData length in bytes.
StringData	STRING	String data, up to 65535 bytes, with no terminating NUL

E.5 onMetaData

The FLV metadata object shall be carried in a SCRIPTDATA tag named `onMetadata`. Various properties are available to a running ActionScript program via the `NetStream.onMetaData` property. The available properties differ depending on the software creating the FLV file. Typical properties include:

onMetadata properties

Property Name	Type	Comment
audiocodecid	Number	Audio codec ID used in the file (see E.4.2.1 for available SoundFormat values)
audiodatarate	Number	Audio bit rate in kilobits per second
audiodelay	Number	Delay introduced by the audio codec in seconds
audiosamplerate	Number	Frequency at which the audio stream is replayed
audiosamplesize	Number	Resolution of a single audio sample
canSeekToEnd	Boolean	Indicating the last video frame is a key frame
creationdate	String	Creation date and time
duration	Number	Total duration of the file in seconds
filesize	Number	Total size of the file in bytes
framerate	Number	Number of frames per second
height	Number	Height of the video in pixels
stereo	Boolean	Indicating stereo audio
videocodecid	Number	Video codec ID used in the file (see E.4.3.1 for available CodecID values)
videodatarate	Number	Video bit rate in kilobits per second
width	Number	Width of the video in pixels

E.6 XMP Metadata in FLV

The XMP metadata object shall be carried in a SCRIPTDATA tag named `onXMPData`. The tag shall be placed at time 0. The tag should be after all time 0 `onMetaData` tags, and before all time 0 audio or video tags, but readers should not require this ordering.

XMPMetadata object

Property Name	Type	Comment
liveXML	String or Long string	XMP metadata, formatted according to the XMP metadata specification

For further details, see www.adobe.com/devnet/xmp/pdfs/XMPSpecificationPart3.pdf

Annex F. FLV Encryption

F.1 Overview

FLV files are encrypted as follows:

1. An encryption header, containing the encryption metadata needed to decrypt the FLV such as encryption algorithm, key length, and content encryption key retrieval protocol identifier, is stored as ScriptData immediately after the FLV Header, before any encrypted content.
2. Content-carrying tags are encrypted.
    a. For efficiency, there is an option to only encrypt subset of the tags, such as I-frames.
    b. If the tag is encrypted, the filter flag is turned on in the packet. The filter flag indicates that the packet needs to be pre-processed before decoding. The encryption filter is specified in the packet. Non-compliant players will ignore tags with the filter flag set, as they effectively have a new tag type.
    c. Most metadata (e.g. whether it's an audio or video frame, key frame or I-frame, codec type) are kept in clear so that servers and client-side players can process metadata without the need to decrypt the content.
    d. Encryption is applied to the contents as dictated by the encryption algorithm and the encryption key. The encrypted data is stored in the packet.

This specification defines the header metadata and the format of the encrypted packets.

F.2 Header Information

F.2.1 AdditionalHeader object

In encrypted FLV files, the AdditionalHeader object shall be present, and shall include the Encryption Header object.

The AdditionalHeader object shall be carried in a SCRIPTDATA tag named `|AdditionalHeader`. (Note the vertical bar ('|') in the name.) The object should be present at the beginning of the FLV, with timestamp 0, immediately after the `onMetaData` ScriptData tag. This gives the FLV decoder access to the encryption metadata before it encounters any encrypted tags.

AdditionalHeader object

Property Name	Type	Comment
Encryption	Encryption Header object	Encryption Header

F.2.2 Encryption Header object

The Encryption Header object contains the encryption metadata needed to decrypt the FLV.

Encryption Header object

Property Name	Type	Comment
Version	Number	Version of Encryption Header. Shall be 1 or 2, indicating the version of the encryption format. 1 = FMRMS v1.x products. 2 = Flash Access 2.0 products. Contents protected using either version are in existence, so applications shall be able to consume both versions of the content.

Method	String	Encryption method. Shall be 'Standard'
Flags	Number	Encryption flags. Shall be 0.
Params	Standard Encoding Parameters object	Parameters for encryption method 'Standard'
IF Version == 1		
SigFormat	String	No information is provided on the SigFormat in this document.
Signature	Long string	No information is provided on the Signature in this document.

F.2.3 Standard Encoding Parameters object

This structure contains parameters specific to the 'Standard' encryption method.

Standard Encoding Parameters object

Property Name	Type	Comment
Version	Number	Version. Shall be 1.
EncryptionAlgorithm	String	The encryption algorithm. Shall be 'AES-CBC', which specifies that the encryption used is 'AES-CBC' with padding as per RFC 2630.
EncryptionParams	AES-CBC Encryption Parameters object	Parameters for encryption algorithm 'AES-CBC'.
KeyInfo	Key Information object	Information to get to the decryption key

F.2.4 AES-CBC Encryption Parameters object

This structure contains parameters specific to the encryption algorithm, in this case AES-CBC_128.

AES-CBC Encryption Parameters object

Property Name	Type	Comment
KeyLength	Number	Key length for the encryption algorithm in bytes. Shall be 16 (i.e. 128 bits)

F.2.5 Key Information object

The key information box contains information for retrieving the key for decryption of samples. The details of the entries contained in these boxes, and the mechanism used by the DRM client to retrieve the keys are outside the scope of this specification

Key Information object

Property Name	Type	Comment
SubType	String	IF EncryptionHeader.Version == 1 'APS' = (Adobe Policy Server) An online key agreement negotiation protocol ELSE 'FlashAccessv2' = An online key retrieval protocol

Data	IF SubType == 'APS' Adobe Policy Server object IF SubType == 'FlashAccessv2' FlashAccessv2 object	No further information is provided for SubType 'APS' as it is no longer produced by conforming applications.

F.2.6 FlashAccessv2 object

A Flash Access server provides the decryption key using an online key retrieval protocol.

The FlashAccessv2 object contains the following high level elements (the details of these elements are outside the scope of the document) needed by the FlashAccessv2 module to carry out the online key retrieval.

FlashAccessv2 object

Property Name	Type	Comment
Metadata	Long string	Base 64 encoded metadata used by the DRM client to retrieve the decryption key.

F.3 Encryption of Contents

This section describes how FLV tags are encrypted.

In an encrypted FLV file, each FLV tag can indicate its state of encryption:

- The Filter flag may indicate that pre-processing of the packet is required before it can be rendered.
- In Version 2, when the Filter flag is set, the Selective Encryption indicator may further indicate whether a packet is encrypted.

Whether the file is fully or partially encrypted, in Version 2 (EncryptionHeader.Version == 2) every audio and video packet should have the FLVTAG.Filter bit set. For script data that are not encrypted, the filter bit shall not be set, enabling the player to locate the onMetadata info.

A small set of specified bytes are kept in clear, to enable intelligent client-side processing without decrypting the rest of the content.

F.3.1 Encryption Tag Header

If the Filter flag is set in the FLV tag, the packet contents shall be pre-processed before rendering. The Encryption Tag Header specifies the filters to apply. Filters specify the type of encryption and indicate if encryption is applied.

EncryptionTagHeader

Field	Type	Comments
NumFilters	UI8	Number of filters applied to the packet. Shall be 1.
FilterName	String	Name of the filter. IF EncryptionHeader.Version == 1 'Encryption' ELSE 'SE' SE stands for Selective Encryption.
Length	UI24	Length of FilterParams in bytes

F.3.2 Filter Parameters

FilterParams contains parameters specific for the decryption method.

FilterParams		
Field	**Type**	**Comments**
FilterParams	IF FilterName = 'Encryption' EncryptionFilterParams IF FilterName = 'SE' SelectiveEncryptionFilterParams	Parameters specific to the filter.

The filter parameters for (non-selective) encryption are defined in EncryptionFilterParams. All packets with this field shall be encrypted.

EncryptionFilterParams		
Field	**Type**	**Comment**
IV	UI8 [16]	Contains 16 bytes of IV data for AES-CBC.

The filter parameters for selective encryption are defined in SelectiveEncryptionFilterParams.

SelectiveEncryptionFilterParams		
Field	**Type**	**Comment**
EncryptedAU	UB [1]	Selective Encryption indicator shows if the packet is encrypted. 0 = packet is not encrypted 1 = packet is encrypted.
Reserved	UB [7]	Shall be 0
IV	IF EncryptedAU == 1 UI8 [16]	Only present if the packet is encrypted. Contains 16 bytes of IV data for AES-CBC

F.3.3 Encrypted Body

If the packet is encrypted, then the body shall contain the Encrypted Body described in this section, else the body shall contain the plaintext data.

EncryptedBody		
Field	**Type**	**Comment**
Content	UI8 [Plaintext Length]	Cipher text
Padding	UI8 [Padding Length]	Encrypted padding.

F.3.3.1 Padding

All encrypted samples shall be padded to a multiple of the block cipher's block length.

The padding scheme shall be as described in RFC 2630, which is reproduced here:

Block ciphers expect the input data to be a multiple of k octets (in case of AES 128, a multiple of 16 octets), where k is greater than 1. For such algorithms, the input shall be padded at the trailing end with k - (length mod k) octets all having the value k - (length mod k), where length is the length of the input.

The padding brings the block size to the next integral multiple of the block cipher's block length. The padding is present even when the plaintext is evenly divisible by the block length.

EXAMPLE: If k is 16 bytes and length is 32 bytes, the padding is 16 bytes long containing 0x10, and the block size is 48 bytes.

F.4 Encryption and Metadata

The onMetaData script data shall always be kept in clear when the FLV is encrypted.

This is needed by various FLV parsers to successfully stream the FLV and by media players to provide some contextual information to the use.