Marlin – File Formats Specification

Version 1.1.3
Final

Source	Marlin Developer Community
Date	October 22, 2010

Refer to Notices on page 2 for important legal information

Notice

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, CONCERNING THE COMPLETENESS, ACCURACY, OR APPLICABILITY OF ANY INFORMATION CONTAINED IN THIS DOCUMENT. THE MARLIN DEVELOPER COMMUNITY ("MDC") ON BEHALF OF ITSELF AND ITS PARTICIPANTS (COLLECTIVELY, THE "PARTIES") DISCLAIM ALL LIABILITY OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, ARISING OR RESULTING FROM THE RELIANCE OR USE BY ANY PARTY OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. THE PARTIES COLLECTIVELY AND INDIVIDUALLY MAKE NO REPRESENTATIONS CONCERNING THE APPLICABILITY OF ANY PATENT, COPYRIGHT (OTHER THAN THE COPYRIGHT TO THE DOCUMENT DESCRIBED BELOW) OR OTHER PROPRIETARY RIGHT OF THIS DOCUMENT OR ITS USE, AND THE RECEIPT OR ANY USE OF THIS DOCUMENT OR ITS CONTENTS DOES NOT IN ANY WAY CREATE BY IMPLICATION, ESTOPPEL OR OTHERWISE, ANY LICENSE OR RIGHT TO OR UNDER ANY PATENT, COPYRIGHT, TRADEMARK OR TRADE SECRET RIGHTS WHICH ARE OR MAY BE ASSOCIATED WITH THE IDEAS, TECHNIQUES, CONCEPTS OR EXPRESSIONS CONTAINED HEREIN.

Use of this document is subject to the agreement executed between you and the Parties, if any.

Any copyright notices shall not be removed, varied, or denigrated in any manner.

Intellectual Property

A commercial implementation of this specification requires a license from the Marlin Trust Management Organization.

Contact Information

Feedback on this specification should be addressed to: editor@marlin-community.com

Contact information for the Marlin Trust Management Organization can be found at: http://www.marlin-trust.com/

CONTENTS

1 Introduction

1.1 Document Organization

This document covers the media file formats utilized by implementations of the Marlin Core System Specification [MCS]. It is organized as follows:

- (this) introduction, including abbreviations, definitions and references
- Content File Formats for Broadband and Mobile delivery systems

1.2 Conformance Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

These capitalized key words are used to unambiguously specify requirements and behavior that affect the interoperability and security of implementations. When these key words are not capitalized they are meant in their natural-language sense.

All elements of this specification are considered Normative unless specifically marked Informative. All Normative Elements are Mandatory to implement, except where such an element is specifically marked OPTIONAL. Finally, where Normative elements are described as OPTIONAL, they MAY be omitted from an implementation, but when implemented, they MUST be implemented as described.

1.3 Names and Identifiers

This specification uses Uniform Resource Identifiers [RFC2396] to identify various entities including resources, algorithms, policies, attributes and other application specific objects. This specification relies upon the name and identification techniques defined in [MCS] § 1.3.

1.4 Abbreviations

AES	Advanced Encryption Standard
AES-CTR-128	AES Counter mode with 128-bit key
BKB	Broadcast Key Block
CBC	Cipher Block Chaining
CCI	Copy Control Information
CPRM	Content Protection for Recordable Media
CTR	Counter
DCF	DRM Content Format
DLNA	Digital Living Network Alliance
HBES	Hierarchical Hash-Chain Broadcast Encryption Scheme
ISO	International Organization for Standardization
MAC	Message Authentication Code
MDCF	Marlin DRM Content Format
OMA	Open Mobile Alliance
XML	Extensible Markup Language

1.5 Terms and Definitions

Marlin Content	Media packaged into a Marlin File Format Container
Marlin Content	The File Format and Media Codec Profile that Marlin

Format	Content is contained in.
Content Key	The symmetric key that encrypts the payload of the content
Starfish	The Marlin broadcast encryption scheme based on HBES (Hierarchical Hash-Chain Broadcast Encryption Scheme)

157

158 ## *1.6 References*

159 ### 1.6.1 Normative References

160

[AES]	NIST FIPS 197: Advanced Encryption Standard (AES). November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[AES-MODES]	Recommendation of Block Cipher Modes of Operation. NIST. NIST Special Publication 800-38A. http://csrc.nist.gov/CryptoToolkit/modes/800-38_Series_Publications/SP800-38A.pdf
[AVCFF]	"Information technology – Coding of audio-visual objects – Part 15: AVC File Format", ISO/IEC 14496-15.
[DLNA1]	Digital Living Network Alliance, "Home Networked Device Interoperability Guidelines", Version: 1.0, June 2, 2004.
[DLNAOMF1]	Digital Living Network Alliance, "Home Networked Device Interoperability Guidelines v.1.0 Addendum, Optional Media Format Guidelines", v1.0,
[DLNA1ERR]	Digital Living Network Alliance, "Home Networked Device Interoperability Guidelines Version 1.0 Errata", October 18, 2004.
[hmacwithsha1]	H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. IETF RFC 2104. February 1997. http://www.ietf.org/rfc/rfc2104.txt
[ISMACryp]	"ISMA Encryption and Authentication Specification", version 1.0, February 2004.
[ISO14496-12]	Information technology — Coding of audio-visual objects – Part 12: ISO Base Media File Format". ISO (International Organization for Standardization). ISO/IEC 14496-12, 2003
[ISOMFF]	"Information technology – Coding of audio-visual objects – Part 12: ISO base media file format", second edition, ISO/IEC 14496-12:2005(E), 2005-04-01.
[MCS]	Marlin - Core System Specification version 1.0. December 2006.
[MIME]	N. Freed & N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. IETF RFC 2045. November 1996. http://www.ietf.org/rfc/rfc2045.txt
[MP4S]	"Information technology – Coding of audio-visual objects – Part1: Systems", ISO/IEC 14496-1:2004.
[MP4FF]	"Information technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part.14: MP4 file format", ISO/IEC 14496-14:2003.
[MURIT]	"URI Templates for Marlin", Version 1.0, Sept 10 2007.
[OMADCF]	DRM Content Format V2.0, Candidate version 2.0. March 29, 2005. Open Mobile Alliance. OMA-DRM-DCF-V2_0-0-20050329-C
[RFC1738]	T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators (URL).* IETF RFC 1738. December 1994. http://www.ietf.org/rfc/rfc1738.txt

[RFC2104]	H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, IETF RFC 2104, February 1997. http://www.ietf.org/rfc/rfc2104.txt
[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.
[RFC2396]	T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396. August 1998. http://www.ietf.org/rfc/rfc2396.txt
[RFC2616]	R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, eds. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616 http://www.ietf.org/rfc/rfc2616.txt
[RFC2630]	Cryptographic Message Syntax. Network Working Group. R. Housley, Request for Comments: 2630. June 1999. http://www.ietf.org/rfc/rfc2630.txt
[RFC3986]	Uniform Resource Identifier (URI): Generic Syntax. Network Working Group. T. Berners-Lee, Request for Comments: 3986. January 2005 http://www.ietf.org/rfc/rfc3986.txt
[RFC3394]	J. Schaad, R. Housley. *Advanced Encryption Standard (AES) Key Wrap Algorithm*. IETF RFC 3394. September 2002. http://www.ietf.org/rfc/rfc3394.txt
[RSA-1_5]	B. Kaliski, J. Staddon. *PKCS #1: RSA Cryptography Specifications Version 2.0.* IETF RFC2437. October 1998. http://www.ietf.org/rfc/rfc2437.txt
[SCTE52]	ANSI/STCE 52: "Data Encryption Standard – Cipher Block Chaining Packet Encryption Specification". http://www.scte.org/documents/pdf/ANSISCTE522003DVS042.pdf
[SHA1]	FIPS PUB 180-1. *Secure Hash Standard.* U.S. Department of Commerce/National Institute of Standards and Technology. http://www.itl.nist.gov/fipspubs/fip180-1.htm
[SHA256]	FIPS PUB 180-2. *Secure Hash Standard.* U.S. Department of Commerce/National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[URN]	R. Moats.. *URN Syntax.* IETF RFC 2141. May 1997. http://www.ietf.org/rfc/rfc2141.txt L. Daigle, D. van Gulik, R. Iannella, P. Falstrom.. *URN Namespace Definition Mechanisms.* IETF RFC 2611. June 1999. http://www.ietf.org/rfc/rfc2611.txt
[X509Cor1]	*ITU-T Recommendation X.509 (2000) Corrigendum 1: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks Technical Corrigendum 1*

1.6.2 Informative References

[3GPP]	3rd Generation Partnership Project, "Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP) (Release6)", 3GPP TS 26.244 V6.2.0, 2004-12.
[PKIX]	R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509*

	Public Key Infrastructure Certificate and CRL Profile. IETF RFC 3280. April 2002. http://www.ietf.org/rfc/rfc3280.txt
[URL]	T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators (URL).* IETF RFC 1738. December 1994. http://www.ietf.org/rfc/rfc1738.txt
[X509]	*ITU-T Recommendation X.509* (1997 E): *Information Technology - Open Systems Interconnection - The Directory: Authentication Framework,* June 1997.

2 File Format for Marlin Content

In order to ensure at least a minimal set of interoperability between implementations of core system specifications, Marlin specifies a set of content file format profiles in terms of containers, media encryption mechanisms, codecs profiles, and metadata structures.

The Marlin Content Formats SHALL be supported by all devices implementing the core system specifications, providing that the device capability supports the media profile

2.1 File Format for Marlin Broadband Content

Marlin Broadband content SHALL use the MP4 file format defined in [MP4FF] for protected and non-protected media. In case of the AVC codec used, the [AVCFF] is also referred to. For protected content, the scheme signaling method defined in the [ISOMFF] and adopted by the [ISMACryp] is used.

The encryption of the content SHALL use AES Counter-Mode encryption, with 128-bit key size (AES-CTR-128) following the [ISMACryp] specification. For the encryption scheme, boxes defined in [ISMACryp] SHALL be used within the Protection Scheme Information Box ('sinf').

In addition, new boxes in the sinf are defined in order to carry Octopus related information such as Octopus content identifier.

While Octopus allows a license to be delivered separately from the content file, it can also be embedded in the content file. A box definition to store the license is specified in this document. Whenever possible, licenses SHOULD be stored in this box.

The Marlin Broadband content format adopts DLNA media profiles as baseline. The referenced DLNA media profiles adopt the ISO Base Media File Format. The remainder of this document specifies extensions to this profile.

This section addresses the following topics:

- File identification for the Marlin broadband content files.
- Specifications for protection of the Marlin broadband content files.
- Extensions to the ISO Base Media File Format for metadata in the Mar lin broadband content files.
- List of metadata items for the Marlin broadband content files.
- DLNA media profiles referred to by this specification.

2.1.1 File Identification

The brand '**mln1**' SHALL be used to indicate conformance with this specification in a file. Whenever possible, the brand '**mln1**' SHOULD be used as a major-brand in a Marlin Broadband content file.

2.1.2 Protection Scheme Information

The Marlin Broadband file format defines new boxes in the Protection Scheme Information Box ('sinf) which is identified by the scheme-type value 'iAEC' in the Scheme Type Box ('schm') as defined in the [ISMACryp].

These boxes SHALL be used to carry the Marlin Octopus specific information and encryption parameters defined in the [ISMACryp]. The encryption scheme for the protected Marlin Broadband content files SHALL be the default encryption scheme of the [ISMACryp], i.e. AES-CTR-128. The structure of the Marlin Broadband Profile 'sinf' box is based on the one specified in [ISMACryp] and is shown in Table 2-1.

Box						Defined by…	Description
sinf						MPEG-4	*Container for protection information*
	frma					MPEG-4	*The 4CC of the original un-protected sample description*
	schm					MPEG-4	*Protection scheme type*
	schi					MPEG-4	*Container of protection information interpreted by the scheme.*
		iKMS				ISMA	*ISMA KMS Box*
		iSFM				ISMA	*ISMA Sample Format Box*
		iSLT				*Marlin BP*	*Salting key needed for AES-CTR-128 defined in [ISMACryp].*
		8id				*Marlin BP*	*Octopus ID*
		8bdl				*Marlin BP*	*Octopus Bundle (optional)*

Table 2-1 Structure of Protection Scheme Information Box ('sinf')

The ISMA KMS Box enables interoperability only if participants use the same Key Management System. Therefore, the contents of the ISMA KMS Box SHOULD contain the following URI to indicate the Marlin KMS:

urn:marlin:ismacryp:kms-id:0

2.1.2.1 ISMA Salting Key Box

2.1.2.1.1 Definition

Box Type: 'iSLT'
Container: Scheme Information Box ('schi')
Mandatory: Yes
Quantity: Exactly one

2.1.2.1.2 Syntax

```
aligned(8) class ISMASaltingKeyBox extends Box ('iSLT') {
        bit (64)                saltValue;
}
```

2.1.2.1.3 Semantics

saltValue is a binary data that specifies salting key used in the default encryption scheme defined in the [ISMACryp], i.e. AES-CTR-128.

2.1.2.2 Octopus ID Box

2.1.2.2.1 Definition

Box Type: ‘8id ’
Container: Scheme Information Box (‘schi’)
Mandatory: Yes
Quantity: Exactly one

2.1.2.2.2 Syntax

```
aligned(8) class OctopusIdBox extends Box (‘8id ’) {
        string                  id;
}
```

2.1.2.2.3 Semantics

id is a null-terminated UTF-8 characters that indicates Octopus Content Object Identifier.

2.1.2.3 Octopus Bundle Box

2.1.2.3.1 Definition

Box Type: ‘8bdl’
Container: Scheme Information Box (‘schi’) and movie level user data box
Mandatory: No
Quantity: Zero or more

2.1.2.3.2 Syntax

```
aligned(8) class OctopusBundleBox extends Box (‘8bdl’) {
        unsigned int(32)                encoding;
        unsigned int(32)                encoding_version;
        bit(8)                  bundle_data[];
}
```

2.1.2.3.3 Semantics

encoding is four-character code that indicates encoding type of the bundle_data field. The values defined in this specification are indicated in Table 2-2.
encoding_version is an integer that indicates version of the encoding type identified by the encoding field in this box. The values defined in this specification are indicated in Table 2-2.
bundle_data is data that represents the Octopus objects.

encoding	encoding_version	Description
‘xml ‘	0x00000000	XML encoding used for this version of the Marlin Broadband file format

Table 2-2 Encoding and encoding version options

2.1.3 Additions to the ISO Base Media File Format

This section specifies the generic metadata format and its protection scheme.

The metadata structure is for the generic metadata including text, URL and compressed/uncompressed binary data. This metadata structure conforms to the meta box defined in the [ISOMFF] and specifies contained boxes within the meta box as allowed by the [ISOMFF]. The metadata structure specified is identified by a handler_type as defined in the [ISOMFF]. The handler_type is declared in this chapter.

In order to support signing the metadata, some additional boxes are specified.

2.1.3.1 Metadata structure

2.1.3.1.1 Handler type

The handler_type for the Handler Box in the Meta box SHALL be set to '**mtdb**'.

2.1.3.1.2 Metadata Item Directory Box

The metadata item directory box provides a directory of metadata in the containing file, movie or track. This box consists of metadata items which directly store metadata inside. Unless the primary item box occurs within the containing meta box, this box MUST exist..

The definition, syntax and semantics are specified below.

Box Type : 'idir'
Container : Meta Box ('meta')
Mandatory : No
Quantity : zero or exactly one

```
aligned(8) class MetaItemDirectoryBox () extends Box ('idir') {
        Box                                     metadata_items[];
}
```

2.1.3.1.3 Metadata Item Box

Each metadata item is stored in a metadata item box and listed in a metadata item directory box. The metadata item box consists of a collection of entries (or boxes). The Item Data Entry specified in a following clause is a mandatory entry of this box and stores metadata body. Optional entries MAY be located in this box to add information or functionalities for the metadata. The item tag value (or box type) of the metadata item box indicates a type of the metadata stored

The definition, syntax and semantics of the Metadata item box are specified below.

Box Type : item_tag
Container : Metadata Item Directory Box ('idir')
Mandatory : Yes
Quantity : One or more (zero or more per each item_tag)

```
aligned(8) class MetadataItemBox () extends Box (item_tag) {
        ItemReferenceEntry              ref_entry;      // optional
        ItemLanguageEntry               lang_entry;     // optional
        ItemDataEntry                   data_entry;
}
```

2.1.3.1.4 Item Data Entry

The item data entry stores actual metadata within a metadata item box. There SHALL be exactly one item data entry in a metadata item box. The type of the metadata and its structure are identified by the item tag (or box type) of the containing metadata item box.

The definition, syntax and semantics are specified below.

Box Type : 'idat'
Container : Metadata Item Box
Mandatory : Yes
Quantity : Exactly one

```
aligned(8) class ItemDataEntry () extends FullBox('idat', version = 0, flags) {
        unsigned int(16)                    item_ID;
        bit(8)                              data[];
}
```

item_ID is an arbitrary integer 'name' for this resource which can be used to refer to it. The value SHALL be unique within the container meta box (not the Metadata Item Directory box).
data is a byte stream storing metadata. The structure in this field depends on item tag (or box type) of the container Metadata Item Box.

2.1.3.1.5 Item Reference Entry

The item reference entry provides a reference from the containing metadata item box to another metadata item within a meta box. The item IDs declared in item reference entries SHALL be used as the references.
The definition, syntax and semantics are specified below.

Box Type : 'iref'
Container : Metadata Item Box
Mandatory : No
Quantity : Zero or exactly one

```
aligned(8) class ItemReferenceEntry () extends FullBox('iref', version = 0,
flags) {
        unsigned int(16)                    entry_count;
        for (i = 0; i < entry_count; i++) {
                unsigned int(16)            item_ID;
        }
}
```

entry_count is an integer that gives the number of the entries of item_ID.
item_ID is an integer that indicates the ID of the item which is referenced to by the containing item.

2.1.3.1.6 Item Language Entry

The item language entry provides the language code for textual metadata stored in the containing metadata item box. This entry is meaningful only for the textual metadata item.
The definition, syntax and semantics are specified below.

Box Type : ‘ilng’
Container : Metadata Item Box
Mandatory : No
Quantity : Zero or exactly one

```
aligned(8) class ItemLanguageEntry () extends Box(‘iref’) {
        const bit(1)                    pad = 0;
        unsigned int(5)[3]              language; // ISO-639-2/T
language code
}
```

language declares the language code for the text stored in the ItemDataEntry within a Metadata Item box where this entry is. See ISO 639-2/T for the set of three character codes. Each character is packed as the difference between its ASCII value and 0x60. The code is confined to being three lower-case letters, so these values are strictly positive.

2.1.3.2 Signed metadata

In order to support the feature of verifying the integrity of metadata, the signing scheme for metadata specified in this section SHALL be used. The boxes are stored in the protection scheme information box. As defined in the [ISOMFF], the item protection box works as a container of the protection scheme information box(es) and the item information box describes association between the protection scheme information and protected items.

2.1.3.2.1 Scheme Type

The scheme signaling that uses a scheme type box (‘schm’) and a scheme information box (‘schi’) is used in the protection scheme information box (‘sinf’). The value of the scheme_type in the scheme type box SHALL be ‘**smtd**’ for this signing scheme. The scheme type SHALL occur only in the protection scheme information box within a meta box.

2.1.3.2.2 Scheme Information

The scheme information box (‘schi’) identified by the scheme_type declared in the section 2.1.3.2.1 stores the following boxes.

2.1.3.2.3 Metadata Signature Box

The metadata signature box stores signature of metadata items and indicates the signing scheme used for the signature. The signature targets are metadata items which item IDs are associated with the container protection scheme information box (‘sinf’) by the item information box (‘iinf’). The signed metadata item boxes are concatenated in numeric order of their item IDs and signed together.
The definition, syntax and semantics are specified below.

Box Type : ‘msig’
Container : Scheme Information Box (‘schi’)
Mandatory : Yes
Quantity : Exactly one

```
aligned(8) class MetadataSignatureBox () extends FullBox(‘msig’,
version = 0, flags) {
        unsigned int(32)                    signing_scheme;
        bit(8)                              signature[];
}
```

signing_scheme is an integer that indicates the scheme of signing metadata. The following values are defined in this document.
- ‘RSA1’ : RSA-SHA1[RSA-1_5] for public key signatures.

signature is a signature value generated according to the scheme.

Certificate Path Box

The certificate path box contains an ordered list of X.509 certificates. The certificates are used to verify signature in the metadata signature box. Therefore, this box occurs only when the signing scheme specified in the metadata signature box requires the certificates. For example, RSA signature requires certificates. On the other hand, HMAC signature does not.

Box Type : ‘crtp’
Container : Scheme Information Box (‘schi’)
Mandatory : No
Quantity : Zero or exactly one

```
aligned(8) class CertificatePathBox () extends FullBox(‘crtp’, version =
0, flags) {
        bit(8)                              x509certpath[];
}
```

x509certpath is an ordered list of X.509 certificates packaged in a PKIPath for the signature in the metadata signature box.

Metadata item for signed metadata

To prevent replacement of signature and metadata, Octopus content ID(s) for track(s) associated with the signed metadata MUST be included in the metadata items to be signed. A metadata item to indicate which Octopus content ID(s) is/are included in signed metadata is specified in this clause. The metadata item contains a list of track IDs. This metadata item box is replaced with Octopus ID boxes in tracks identified by the track IDs and signed together with other metadata items to be signed. The order of replaced Octopus ID boxes shall be the same as the order of track IDs in the list.

Item	Tag	Data type	Description
Octopus content ID references	‘OIDR‘	Binary	This item contains a list of track IDs. This item is only for the signed metadata scheme specified in this document. Octopus ID boxes stored in tracks indicated by the list are signed together with other metadata items.

The syntax and semantics of the item data entry for this metadata item is as follows.

```
aligned(8) class OctopusContentIDReferencesItemDataEntry ()
extends ItemDataEntry('idat', version = 0, flags, item_ID) {
        unsigned int(32)                        entry_count;
        for (i = 0; i < entry_count; i++) {
                unsigned int(32)                track_ID;
        }
}
```

entry_count is an integer that gives the number of the entries of track_IDs. When the value of this field is set to 0, the entry refers to the containing track when this entry is in the track-level meta box or all tracks in the containing movie or file when this entry is in the movie-level or file-level meta box.
track_ID is an integer that indicates the referenced track by this entry.

2.1.4 Metadata Items

The list below is examples of metadata items, which could be used by an audio distribution service.
Multiple instances of metadata item with the same item tag are allowed for some items. Otherwise, there SHALL be only one instance of the metadata item in a metadata item directory box. One item and other items referring to the item by the item reference box may form a group. In the following list, still image metadata and its associated metadata typically form a group. See the list below for which item may have multiple instances or be a member of a group.
When an item stores textual metadata or a string, it is encoded with UTF-8 or UTF-16 characters. It's not a null-terminated string. If UTF-16 is used, the string shall start with the BYTE ORDER MARK (0xFEFF), to distinguish it from a UTF-8 string. This mark does not form part of the final string.
When an item stores URL metadata, it is a string encoded with UTF-8 characters. It's not a null-terminated string.
The JPEG metadata items in the following table conform to codec related guidelines of the media profile, JPEG_SM, of DLNA guidelines [DLNA1].

Item	Tag	Data type	Description
Album Product ID	'APID'	Text	This item contains the product ID of an album. This ID may arbitrarily be assigned and is managed by a content holder.
Album Title	'TALB'	Text	This item contains an album title.
Album Main Artist	'AART'	Text	This item contains the main artist of an album.
Track Number	'TRKN'	Text	This item contains a track number. A track number is a sequence number that indicates a track's position in an album. To express a track number, a numerical string that consists of one or more digits "0" to "9" shall be used Examples are: "1234", "0000"

Song Title	‘TITL‘	Text	This item contains a track title.
Song Title (for data sorting)	‘TITS‘	Text	This item contains a track title for data sorting.
Song Subtitle	‘STIT‘	Text	This item contains a track subtitle.
Song Subtitle (for data sorting)	‘STIS‘	Text	This frame contains a track subtitle for data sorting.
Song Time	‘TLEN‘	Text	This item contains the playing duration of a track in milliseconds. To express the track duration, it is necessary to use a nonnegative numerical string that consists of one or more digits “0” to “9”, where “0” may not be used as the first digit. Examples are: Correct : “1234” Wrong : “01234”, “-123”, “1.5”
Phonogram Rights	‘PGMR‘	Text	This item contains the phonogram rights (or rights of the master recording) of a track.
Label/Publisher	‘TPUB‘	Text	This item contains the publisher or label name of a track.
URL to Label/Publisher	‘WPUB‘	URL	This item contains a link to the official webpage of a track’s record label or publisher.
Record Company	‘RCDC‘	Text	This item contains the recording company of a track. The recording company that has release or published this track is contained.
URL to Record Company	‘WRCC‘	URL	This item contains a link to the official webpage of the recording company of a track.
ISRC Code	‘ISRC‘	Text	This item contains the International Standard Recording Code (ISRC) (12-character code) of a track.
Main Artist	‘MART‘	Text	This item contains the main artist of a track.
Main Artist (for data sorting)	‘MATS‘	Text	This item contains the main artist of a track for data sorting.
URL to Main Artist	‘WOAR‘	URL	This item contains a link to the official webpage of a track’s artist.
Lyricist	‘TEXT‘	Text	This item contains the lyricist of a track.
Lyricist (for data sorting)	‘TXTS‘	Text	This item contains the lyricist of a track for data sorting.
Song Writer	‘COMP‘	Text	This item contains the composer of a track.
Song Writer (for data sorting)	‘COMS‘	Text	This item contains the composer of a track for data sorting.
Arranger	‘ARGM‘	Text	This item contains the arranger of a track.
Arranger (for data sorting)	‘ARGS‘	Text	This item contains the arranger of a track for data sorting.
Producer	‘PRDC‘	Text	This item contains the producer of a track.
Producer (for data	‘PRDS‘	Text	This item contains the producer of a

sorting)			track for data sorting.
Thumbnail (JPEG)	‘THJP‘	JPEG	This item contains thumbnail data of a track. Multiple setting is possible. Associable with other items. Other members of this group are “Copyright of Thumbnail”.
Copyright of Thumbnail	‘CRTH‘	Text	This item contains the copyrights of a thumbnail of a track. If thumbnail data is not present, this item shall not exist. Multiple setting is possible. Associable with other items. Other members of this group are “Thumbnail (JPEG)”.
Cover Art (JPEG)	‘CAJP‘	JPEG	This item contains a cover art of a track. Multiple setting is possible. Associable with other items. Other members of this group are “Copyright of Cover Art (JPEG)”.
Copyright of Cover Art	‘CRCA‘	Text	This item contains the copyrights of a cover art of a track. If cover art data is not present, this item shall not exist. Multiple setting is possible. Associable with other items. Other members of this group are “Cover Art (JPEG)”.
Lyrics (JPEG)	‘LYJP‘	JPEG	This item contains the lyrics data of a track. Multiple setting is possible. Associable with other items. Other members
Copyright of Lyrics	‘CRLY‘	Text	This item contains the copyrights of the lyrics data of a track. If lyrics data is not present, this item shall not exist. Multiple setting is possible. Associable with other items. Other members of this group are “Lyrics (JPEG)”.
URL to Shop	‘WCOM‘	URL	This item contains a link to where the user can buy this track. Multiple setting is possible.
Octopus ID	‘8ID ‘	Text	Octopus ID which is originally stored in the Octopus ID Box per track. This item in a movie-level meta box shall exist only when all protected tracks in the movie have identical Octopus IDs.

512

513 ### 2.1.5 Media Format Profiles

514 The Marlin Broadband content format refers to the [DLNA1] and [DLNAOMF1] for
515 profiles of codecs and basis of formats. Since the Marlin Broadband content format is
516 based on the [ISOMFF], the formats referred to are also derived from the [ISOMFF]
517 and this document defines the incremental specifications to these formats
518 The referred media format profiles are as follows.
519

Name	Description	Media Class	Media Format	MIMEType	Reference
AAC_ISO	Mandatory Profile for audio media class	Audio	AAC	audio/mp4	[DLNAOMF1]

	content				
AAC_LTP_ISO	Profile for audio media class content	Audio	AAC	audio/mp4	[DLNAOMF1]
AAC_LTP_MULT5_ISO	Profile for audio media class content with up to 5.1 channels	Audio	AAC	audio/mp4	[DLNAOMF1]
AAC_LTP_MULT7_ISO	Profile for audio media class content with up to 7.1 channels	Audio	AAC	audio/mp4	[DLNAOMF1]
AAC_MULT5_ISO	Profile for audio media class content with up to 5.1 channels	Audio	AAC	audio/mp4	[DLNAOMF1]
HEAAC_L2_ISO	Profile for audio media class content	Audio	AAC	audio/mp4	[DLNAOMF1]
HEAAC_L3_ISO	Profile for audio media class content	Audio	AAC	audio/mp4	[DLNAOMF1]
HEAAC_MULT5_ISO	Profile for audio media class content with up to 5.1 channels	Audio	AAC	audio/mp4	[DLNAOMF1]
AVC_MP4_MP_SD_AAC_MULTI5	Mandatory profile for AV class media. AVC wrapped in MP4 main profile standard def with AAC audio.	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_MP_SD_AAC_LTP_MULT5	AVC wrapped in MP4 main profile standard def with AAC LTP audio.	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_MP_SD_AAC_LTP_MULT7	AVC wrapped in MP4 main profile standard def with AAC LTP audio.	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_MP_SD_HEAAC_L2	AVC wrapped in MP4 main profile standard def with HEAAC L2 audio	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_BL_CIF30_AAC_MULT5	AVC wrapped in MP4 baseline profile CIF30 with AAC audio	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_BL_CIF30_HEAAC_L2	AVC wrapped in MP4 baseline profile CIF30 with HEAAC audio	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_BL_CIF30_AAC_LTP	AVC wrapped in MP4 baseline profile CIF30 with AAC LTP audio	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_BL_CIF30_AAC_LT	AVC wrapped in MP4 baseline profile	AV	MPEG-4	video/mp4	[DLNAOMF1]

P_MULT5	CIF30 with AAC LTP audio				
AVC_MP4_BL_CIF15_AAC	AVC wrapped in MP4 baseline profile CIF15 with AAC audio	AV	MPEG-4	video/mp4	[DLNAOMF1]
AVC_MP4_BL_CIF15_AAC_LTP	AVC wrapped in MP4 baseline profile CIF15 with AAC LTP audio	AV	MPEG-4	video/mp4	[DLNAOMF1]

2.2 Marlin DRM Content Format (MDCF)

The Marlin DCF file format is based on the OMA DCF format. This format is required for the Marlin OMA Delivery specification.

2.2.1 Approach

The DCF format described in this document is designed to meet the following requirements:

- The format SHALL be content format agnostic
- It SHALL be possible to create a Marlin DCF from an OMA DCF format and vice versa.

The conversion described in this document has to follow the following requirements:

It SHOULD be possible to transform OMAv2 DCF's into Marlin DCF's without re-encryption of the content.
Note: It might be possible to transform OMAv2 content to Marlin content without re-encryption by using re-multiplexing (extract and repackage payload)

- The conversion from OMA DCF to Marlin SHOULD be reversible.
- The conversion SHOULD be performed using a stream process (without the need to store the complete file).

The approach taken in this document is to start with the OMA DCF format [OMADCF] and to replace all OMA specific elements with Marlin elements. Features not required for Marlin (e.g. hash) are removed.
Furthermore, an DNLA MediaProfile is included as an optional indication in the common headers box and we extended the user data to allow for Marlin defined user data.

2.2.2 Marlin DRM Content Format (MDCF) for Discrete Media

This section defines the DRM Content Format for Discrete Media. The format is an object-structured file as defined in section 4 of the ISO Base Media File Format specification [ISO14496-12], but it does not include all the media-related structures due to its simplified, media agnostic design. The actual data structures and conformance to the profile is defined in this specification. If a MDCF includes data structures or functionalities not conforming to this specification, a compliant file parser MAY ignore these. Furthermore, the Marlin DCF has been made to be transformable with the OMA DCF format [OMADCF] and has been adopted to hold Marlin specific data.

2.2.3 MDCF MIME Type

The MIME type for objects conforming to the format defined in this section MUST be

`application/vnd.marlin.drm.mdcf`

and the corresponding file extension MUST be "`.mdc`".

2.2.4 MDCF File Format

The structure of the Marlin Discrete Media profile of DRM Content Format (MDCF) MUST be according to the structure definitions below.

A MDCF file MUST include at least one `MarlinDRMContainer` box. The `MarlinDRMContainer` box is a container for a single Content Object and its associated headers. It MUST appear on the top level, i.e. to conform to this specification, it MUST NOT be nested inside another data type. There MAY exist multiple `MarlinDRMContainer` boxes in a file, but one MUST immediately follow the file header, and they all MUST be located on the top level in the nesting structure.

The *version* indicator field in each box MUST be 0 for files conforming to this specification. All numeric fields in the format MUST be stored in network byte order.

2.2.4.1 Constraints on ISO Format

In files conforming to this specification, box *size* MUST be greater than 1 unless otherwise specified. Some of the mandatory boxes MUST support the 64 bit length field and for those boxes, *size* field MUST be set to 1.
None of the boxes defined in this document SHALL use the *extended_type*.

The FullBox *version* is typically started from zero (0), incremented by each revision. The *flags* field MAY be used to include additional information, but SHOULD normally be set to 0, unless otherwise specified. This specification names each supported box to indicate that a box has a defined structure and a purpose in the MDCF.

There are also placeholders for extensions, with only a generic box reference. These extensions may be defined later, and thus a conforming file parser SHOULD skip any extension boxes it does not understand. In addition, all of the top level boxes (except ftype) are derived from the FullBox type, which supports version information. Later specifications MAY increment the version number if changes are made to any common data structures. Later versions of the boxes defined in this specification should remain backwards compatible with the help of this version indicator. A parser conforming to this specification MAY attempt to parse a box which has a greater version number than this specification, but the conformance is limited to the current version (0) of this specification. A conforming parser MUST check the version number field.

2.2.4.2 File Branding

The ISO base media file format defines a File Type box for identifying the major brand of the media file along with compatible brands. Files conforming to this specification MUST include a File Type box with the MDCF brand as the major brand identifier and compatible brand to make the File Type box fixed length.

The MDCF major brand is 32 bits (4 octets) wide with the hexadecimal value `0x6D646366` ('`mdcf`'). This MUST be followed by a four-octet minor version indicator and the MDCF brand as the single compatible brand. The minor version field is in network byte order. For files conforming to this version of the MDCF specification the version value MUST be 1 (`0x00000001`). A conforming file parser MUST support the minor version number to check whether is support this version of the Marlin DCF. It should be noted that future minor versions of the MDCF file format might use more

compatible brands in the File Type box. If the file parser encounters MDCF holding minor brand indication it does not support, it SHOULD ignore these files.
The Figure 2-1 MDCF file header and body shows the relationship of the File Type, brand, version and rest of the file content.

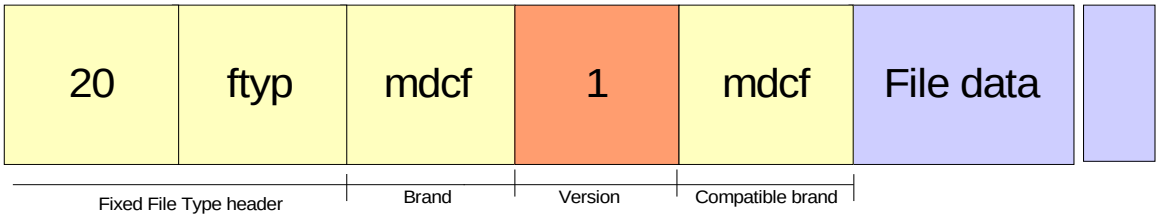

Figure 2-1 MDCF file header and body

2.2.5 Overall structure

The table below outlines the mandatory boxes and their order. Additional boxes MAY be added after the mandatory boxes have first appeared. Table 2-3. Logical MDCF box structure diagram shows the nesting order of the mandatory boxes, on the left is the parent and on the right, the child. The first column indicates which fields and boxes MUST be present in MDCF (marked as 'M') and which boxes MAY appear in the MDCF (marked as 'O'). Note that in the table, the second Marlin DRM Container box MUST include all the mandatory nested boxes as well.

The Marlin DRM Container box MUST include a MDCF headers box and a Content Object box. The first Marlin DRM Container box MUST be the first box after the file header (ftype) and the Marlin Discrete Media headers box MUST be the first box in the Marlin DRM Container.

Present in MDCF	Data type/value			Nesting level	Field purpose
M	Box('ftyp')			0	File header (fixed File Type box, 20 bytes)
M	Box('mdrm')			0	Marlin DRM Container box
M		Box('mdhe')		1	Marlin Discrete Media headers box
M			Box('mhdr')	2	MARLIN DRM Common Headers box
O			Box('udta')	2	ISO User Data box
O			Box('idir')	2	Marlin Metadata Box (optional)
M		Box('mdda')		1	Content Object box
O		Box('modc')		1	Marlin OMA DRM Container box
O		Box('mdlb')		1	Marlin DRM License Box

O	Box('mdrm')			0	If multipart MDCF, additional Marlin DRM Container box
O	Box('mdri')			0	Marlin DRM information box
O		Box('odri')		1	OMA DRM info box
O		Box('mdlb')		1	Marlin DRM License Box
O		Box('skip')		1	Additional free space

Table 2-3. Logical MDCF box structure diagram

2.2.5.1 Marlin DRM Container Box

```
aligned(8) class MarlinDRMContainer extends FullBox('mdrm', version, 0) {
    MarlinDRMDiscreteHeaders ContentHeaders;   // Headers for Discrete Media MDCF
    MarlinDRMContentObject   DRMContent;       // Actual encrypted content
    Box                      Extensions[];     // Extensions, to the end of the box
}
```

The `MarlinDRMContainer` box MUST include a single `MarlinDRMDiscreteHeaders` box and a single `MarlinDRMContent` box, followed by optional extensions. The `Extensions` inside the `MarlinDRMContainer` box are defined by Marlin. The Marlin DRM Container box MUST support 64 bit length attributes, i.e. the *size* attribute MUST be set to 1, and *largesize* MUST be used for determining the box size.

2.2.5.2 Marlin Discrete Media Headers Box

```
aligned(8) class MarlinDRMDiscreteHeaders extends FullBox('mdhe', version, flags) {
    unsigned int(8)        ContentTypeLength;              // Content Type Length
    char                   ContentType[ContentTypeLength]; // Content Type String
    unsigned int(8)        MediaProfileLength;             // Media Profile Length
    char                   MediaProfile[MediaProfileLength];// Media Profile String
    MarlinDRMCommonHeaders CommonHeaders;                  // Common headers
    if(flags & 0x000001) {
        UserDataBox        UserData;                       // ISO User Data Box
    }
    if(flags & 0x000002) {
        MetadataItemDirectoryBox     Metadata;             // Marlin Metadata
    }
}
```

The Discrete Media headers box includes fields specific to the MDCF format and the Common Headers box, followed by an optional user-data box. There MUST be exactly one `MarlinDRMDiscreteHeaders` box in a single Marlin DRM Container box, as the first box in the container.
The *ContentType* field indicates the actual media type contained in the Marlin DRM container. There MUST be exactly one `MarlinDRMCommonHeaders` (see section 2.2.5.2.3 for details) box per a single `MarlinDRMDiscreteHeaders` box.

Field name	Type	Purpose
ContentTypeLength	Unsigned int(8)	Length of the ContentType

		field
ContentType	ContentTypeLength octets	The MIME media type of the plaintext data encoded as US-ASCII
MediaProfileLength	Unsigned int(8)	Length of the MediaProfile field
MediaProfile	MediaProfileLength octets	The media profile of the plaintext data encoded as US-ASCII
CommonHeaders	MarlinDRMCommonHeaders	Marlin DRM Common Headers box as in 2.2.5.2.3
UserData	UserDataBox	User Data as defined in 2.2.5.2.4 (OPTIONAL)
Metadata	MetadataItemDirectoryBox	Marlin Metadata defined in Section 2.2.5.2.5 (OPTIONAL)

Table 2-4. MARLIN DRM Discrete Media header fields

2.2.5.2.1 ContentType

The *ContentType* field MUST indicate the original MIME media type of the Content Object i.e. what content type the result of a successful extraction of the `MarlinDRMContentObject` box represents. The *ContentType* field is encoded using US-ASCII encoding and MUST NOT include a NULL character.
It is RECOMMENDED that the value represented in the ContentType field follows the values indicated in table 8 of the DNLA interoperability Guidelines [DLNA1][DLNAOMFA1].

2.2.5.2.2 MediaProfile

The *MediaProfile* field SHOULD indicate the DLNA media format profile [DLNA1][DLNAOMFA1] of the Content Object i.e. what content type the result of a successful extraction of the `MarlinDRMContentObject` box represents. The *MediaProfile* field is encoded using US-ASCII encoding and MUST NOT include a NULL character. In the case that no *MediaProfile* field is included, the *MediaProfileLength* field SHALL be set to 0.

2.2.5.2.3 CommonHeaders

```
aligned(8) class MarlinDRMCommonHeaders extends FullBox('mhdr', version, 0) {
    unsigned int(8)  EncryptionMethod;      // Encryption method
    unsigned int(8)  EncryptionPadding;     // Padding type
    unsigned int(64)                 PlaintextLength; // Plaintext
content length in bytes
    unsigned int(16)                 ContentIDLength; // Length of
ContentID field in bytes
    char          ContentID[ContentIDLength];  // Content ID string
    Box           ExtendedHeaders[]; // Extended headers boxes
}
```

The Common Headers box defines a structure for the required headers. Their semantics are defined in the sections below. This box MUST appear in a MDCF.

A Device MUST NOT modify any of the fields in the Common Headers box.

2.2.5.2.3.1 Common Headers Version

The *version* field of the `FullBox` defines which version of DRM Content Format specification was used by the author of the Content Object. The value for *version* MUST be 0 for objects conforming to this specification.

2.2.5.2.3.2 EncryptionMethod Field

The *EncryptionMethod* field defines how the encrypted content can be decrypted. Values for the field are defined in the table below.

Algorithm-id	Value	Semantics
NULL	0x00	No encryption for this object. NULL encrypted Content Objects may be used without acquiring a Rights Object. Value of the *PaddingScheme* field MUST be 0.
AES_128_CBC	0x01	AES symmetric encryption as defined by NIST [AES-MODES]. 128 bit keys. Cipher block chaining mode (CBC). 128 bit initialization vector prefixing the ciphertext. Padding according to [RFC2630].
AES_128_CTR	0x02	AES symmetric encryption as defined by NIST [AES-MODES]. 128 bit keys. Counter mode (CTR). 128 bit initial counter block is constructed using a unique counter that prefixes the ciphertext. No padding.

Table 2-5. Algorithm-id values

Content packagers and import functions should take care in using NULL *EncryptionMethod* because, given a null-encrypted Media Object within a MDCF, the following statements hold true:

- Null-encrypted Media Objects do not have any Confidentiality protection.
- Null-encrypted Media Objects can always be used without an associated Rights Object.

2.2.5.2.3.3 PaddingScheme Field

The *PaddingScheme* parameter defines how the last block of ciphertext is padded. Values of the *PaddingScheme* field are defined in the table below:

Padding-Scheme	Value	Semantics
None	0x00	No padding (e.g. when using NULL or CTR algorithm).
RFC_2630	0x01	Padding according to [RFC2630].

Table 2-6. PaddingScheme values

2.2.5.2.3.4 PlaintextLength Field

The *PlaintextLength* field defines the length of the original plaintext. If the content is encrypted, it MUST have a PlaintextLength value set. If the extracted content length does not match the PlaintextLength field value, it is an error and the Content Object

MUST be discarded. In a progressive download scenario, the DRM Client can verify the PlaintextLength only after the complete Content Object has been received and possibly after content use has started.

2.2.5.2.3.5 ContentIDLength Field

The *ContentIDLength* field defines the number of bytes occupied by the *ContentID* field. The value MUST be greater than zero. A Device MUST support Octopus Content Object Identifiers of at least 256 bytes. For best interoperability, content author should not use an Octopus Content Object Identifiers larger than 256 bytes.

2.2.5.2.3.6 ContentID Field

The *ContentID* field MUST contain a globally unique identifier for this Content Object. Note that even if two or more Content Objects contain the same Media Object, the Content Objects will each have a different (and globally unique) Octopus Content Object Identifier. The value MUST be encoded using UTF-8 encoding.

2.2.5.2.3.7 Extended Headers

The *ExtendedHeaders* field MAY include zero or more nested boxes that add functionalities to the common headers. The ExtendedHeaders field continues until the end of the parent box is reached.

2.2.5.2.3.8 Group ID

The *ExtendedHeaders* field MAY include one instance of the *MarlinDRMGroupID* Box:

```
aligned (8) class MarlinDRMGroupID extends FullBox('grpi', version, 0) {
    unsigned int(16)  GroupIDLength;           // length of the Group ID URI
    unsigned int(8)   GKEncryptionMethod;      // Group Key encryption algorithm
    unsigned int(16)  GKLength;                // length of the encrypted Group Key
    char              GroupID[GroupIDLength];  // Group ID URI
    byte              GroupKey[GKLength];      // Encrypted Group Key
}
```

The *GroupID* value identifies this MDCF as part of a group of MDCF's whose Rights can be defined in a common group License instead of (or in addition to) in separate content-specific Licenses. The value of *GroupID* MUST be a URI according to [RFC2396] and MUST contain a globally unique identifier. The value MUST be encoded using UTF-8 encoding.

Generally each content item in a group will be encrypted with a different content item encryption key. A single additional key (used for the whole group) is used to encrypt each content item encryption key for storage in the GroupKey field. This single key is the value of the content encryption key in an associated group license. Note that since the Group ID box is part of the Marlin DRM container box, it is possible for different content items in a multipart MDCF to belong to different groups. The GKEncryptionMethod field defines the algorithm used to encrypt the content item keys, as defined in Section 2.2.5.2.3.2.

Field name	Type	Purpose
GroupIDLength	unsigned int(16)	Length of the Group ID URI field
GKEncryptionMethod	unsigned int(8)	Group Key encryption algorithm
GKLength	unsigned int(16)	Length of the GroupKey field
GroupID	char[]	Group ID URI
GroupKey	byte[]	Encrypted Group Key

Table 2-7. Group ID box fields

2.2.5.2.4 User-Data

A user-data box ('udta'), as defined in [ISO14496-12], MAY be present in the discrete headers box. When a MDCF includes the `UserData`Box, it MUST be added immediately after the `MarlinDRMCommonHeaders` box. The presence of the user-data box MUST be indicated with the flag 0x000001 in the containing box header. The user-data box is a container box for informative user data. This user information is formatted as a set of sub-boxes with specific box types that more precisely define their usage. Each of the sub-boxes MAY be included only once unless otherwise noted.

Some of these sub-boxes contain text information, which is metadata, as defined in [TS26.244]. Devices SHALL at least support the subset of the sub-boxes that is defined in [OMADCF] (except for InfoURL).

2.2.5.2.5 MetadataItemDirectoryBox

This box is defined in Section 2.1.3.1.2, and MAY be present in a Marlin DCF.

2.2.5.3 Content Object Box

```
aligned(8) class MarlinDRMContentObject extends FullBox('mdda', version, 0) {
      unsigned int(64) MarlinDRMDataLength;        // Length of the encrypted content
      byte    MarlinDRMData[];                     // Encrypted content
}
```

The Content Object box MUST include only the data length field and data bytes for a single Content Object. Later revisions of this box may include additional fields, so conforming implementations MUST use the *MarlinDRMDataLength* field to indicate/determine the amount of actual data bytes. The data length includes the Initialization Vector in the beginning of the encrypted data, as depicted in Figure 2-2.

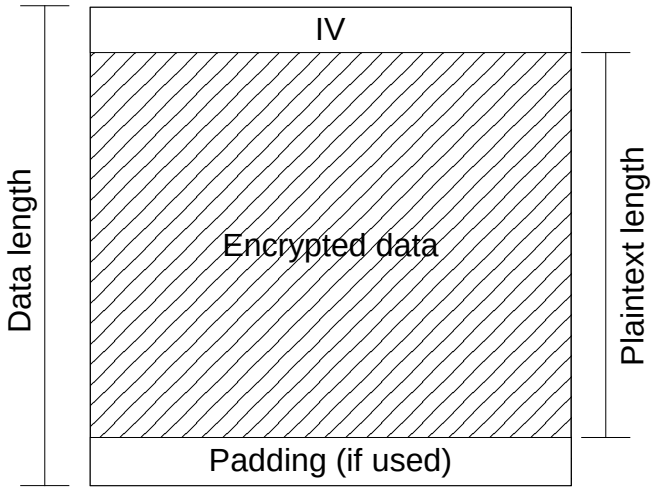

Figure 2-2: Data Length and IV

The Content Object box MUST support the 64 bit size field and thus *size* MUST be set to 1 and *largesize* MUST be used for determining actual box size. The *MarlinDRMDataLength* field MAY indicate a length of zero, and the Device MAY try to acquire the actual Content Object by using e.g. the *ContentURL*, if provided.

Field name	Type	Purpose
MarlinDRMDataLength	Unsigned int(64)	Length of the MarlinDRMData field, in octets
MarlinDRMData	byte []	Content bytes, as specified by the MarlinDRMDiscreteHeaders box

Table 2-8: Content Object box

2.2.5.4 Extensions

Any additional boxes contained in a single Marlin DRM container box have not been defined in this specification. A Content Issuer MAY place additional boxes into the *Extensions* but Devices MAY ignore these.

2.2.5.4.1 Marlin OMA DRM Container box

This box allows storage of information related to an OMA DCF file that has been the source for creation of this MDCF.

```
aligned(8) class MarlinOMAContainer extends FullBox('modc', version, 0) {
    OMADRMDiscreteHeaders  discreteHeaders;
    Box                    OMAextensions[];
}
```

Table 2-9: Marlin OMA Container box fields

2.2.5.4.1.1 discreteHeaders

THE OMADRMDISCRETEHEADERS BOX AS IS PRESENT IN THE ORIGINAL [OMADCF].

2.2.5.4.1.2 OMAextensions

The Extensions boxes present in the original [OMADCF].

2.2.5.4.2 Marlin DRM License Box

The Octopus License box MAY be used to insert a license, into a MDCF.
An Extended Box MAY include one or more Octopus License boxes.

```
aligned(8) class MarlinDRMLicense extends FullBox('mdlb', version, 0) {
    unsigned int(32)     encoding;
    unsigned int(32)     encoding_version;
    bit(8)               license_data[];        // binary Rights Object
}
```

Table 2-10: Marlin Octopus License box fields

2.2.5.4.2.1 encoding

Encoding is four-character code that indicates encoding type of the *License_data* field. The values defined in this specification are indicated in Table 2-11.

2.2.5.4.2.2 encoding_version

Encoding_version is an integer that indicates version of the encoding type identified by the encoding field in this box. The values defined in this specification are indicated in Table 2-11.

2.2.5.4.2.3 license _data

License _data is data that represents the Octopus objects.

encoding	encoding_version	Description
‘xml ‘	0x00000000	XML encoding used for this version of the Marlin Broadband file format

Table 2-11, Encoding and encoding version options

2.2.5.5 Marlin DRM Information Box

The `MarlinDRMInformation` box MUST be located at the top level of the box hierarchy and there MUST NOT be more than one instance of the box per MDCF. The `MarlinDRMInformation` box MAY include free space boxes as defined in ISO base media file format [ISO14496-12] to pre-allocate space for editing. A `MarlinDRMInformation` box MUST NOT appear in the beginning of the file, but MAY appear after the last `MarlinDRMContainer` (see 2.2.5.1) in MDCF. Having the `MarlinDRMInformation` box as the last box in the file is RECOMMENDED. A Device MAY modify, extend, truncate, delete or add the `MarlinDRMInformation` box. The contents of the box MUST be interpreted as an array of Boxes, continuing until the end of the parent box.

```
 aligned(8) MarlinDRMInformation extends FullBox(‘mdri’, version, 0) {
      Box    data[];              // array of any boxes and free space
}
```

2.2.5.5.1 Marlin DRM License Box

A `MarlinDRMInformation` box MAY include zero or more Licenses. The License is treated as binary data and a Device MAY add or delete Octopus License boxes in the `MarlinDRMInformation` box.
See Section 2.2.5.4.2.

2.2.5.5.2 OMA DRM Info box

```
aligned(8) OMAHash extends Box('odri') {
      FileTypeBox  ftype;            // ftype of original OMA DCF file
      Byte         OMAHashValue[];   // hash value of the original DCF file
}
```

This box contains the hash value of the original OMA file. This allows a process that attempts to reconstruct the original OMA file to check whether this has been achieved.

2.2.5.5.2.1 ftype

This is the *ftyp* box of the original OMA file.

2.2.5.5.2.2 OMAHashValue

The hash value of the original DCF file as defined in [OMADCF].

2.2.6 Multiple Marlin DRM Containers

A MDCF MAY include more than one Marlin DRM Container. Each of these containers MUST conform to the definition of the Marlin DRM Container, and MUST be placed sequentially on the top level (i.e. nesting them is not allowed). The media type of the first Marlin DRM Container is considered to be the default media type of the MDCF's content.

Each Marlin DRM Container MUST have a unique *ContentID* in its headers. This kind of a MDCF with multiple Content containers is called a Multipart MDCF.

Note that a Multipart MDCF is different from a MDCF including a composite object. A Composite Object (such as MIME multipart, ZIP and so on) is included in a single Marlin DRM Container and has only one set of Marlin DRM headers associated with it, whereas Multipart MDCFs contain multiple Marlin DRM Containers each including separate headers associated with the contained content. Multipart MDCFs support the association of different rights with individual Media Objects.

2.2.7 Additional Extensions

Additional extension boxes MAY be added after the Marlin DRM Container. A conforming file parser, which does not recognize the additional boxes, MUST ignore them. However, any extensions MUST be designed in a way that the mandatory parts of this specification are always included and the file remains interoperable with conforming implementations.

2.3 File Format using IPMP for Marlin Broadband Content

2.3.1 Overall Designs

Marlin Broadband content SHALL use the ISO Base Media File Format [ISOMFF]. In case of the MPEG-4 Audio/Video codecs and/or AVC codec applied, the MP4 file format [MP4FF] and/or AVC file format [AVCFF] are also referred to respectively.

A file that conforms to this specification uses the Object Descriptor framework and the IPMP framework of MPEG-4 Systems [MP4S] to maintain the security of the content.
Figure 2-3 shows an example of a simple interchange file. The figure also outlines how to access to streams through the Object Descriptor framework and hook up the IPMP system to streams.
Players shall be capable of parsing any files that are recorded in accordance with this specification. Playback of such files may be done depending on their capabilities. Players, when reading the files, shall ignore boxes or descriptors that are unrecognized or prohibited for use in this section. Malfunctions or hang-ups shall not occur, and parsing should be continued by ignoring such unknown boxes or descriptors while aborting playback is allowed. Recorders shall set the version and flags of all boxes to 0, unless this specification states otherwise.

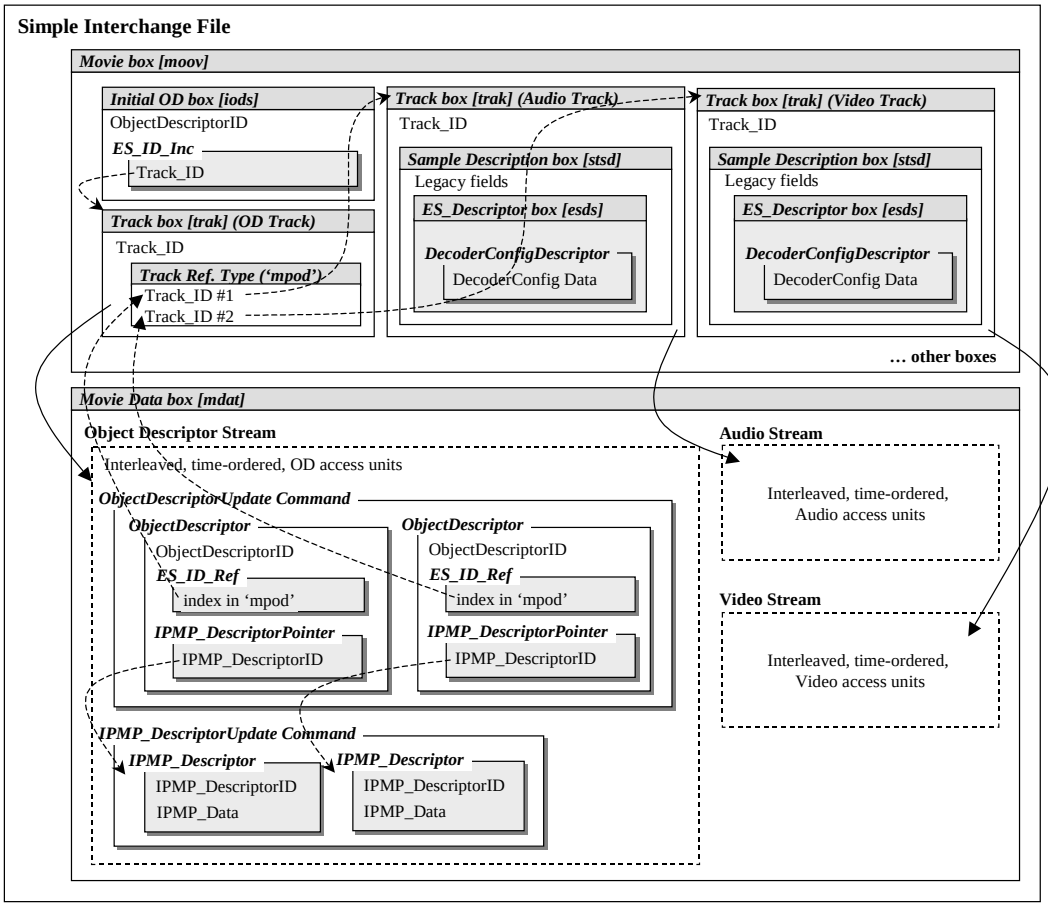

Figure 2-3 An example of a simple interchange file

IPMP framework allows for DRM or content protection system to define specific IPMP_data to the DRM or content protection system. The IPMP_data for Marlin is defined in the following sections to carry Octopus related information such as Octopus content identifier.

While Octopus allows a license to be delivered separately from the content file, it can also be embedded in the content file. A box definition to store the license is specified in this document. Whenever possible, licenses SHOULD be stored in this box.

For this file format, the following brands SHALL appear as compatible brands in File Type Box:

- ‘isom’
- Appropriate brands according to enclosed codec. e.g. ‘mp42’ for AAC-LC.

In addition, ‘iso2’ or later SHALL NOT appear as major brand nor compatible brand in File Type Box.

This section addresses the following topics:

- Specifications for protection of the Marlin broadband content files.
- Extensions to the ISO Base Media File Format using the Private Extension Box.
- Stream encryption.

2.3.2 Protected Stream Support

In this specification, the method for signaling the nature of the protection shall conform to MPEG-4 Systems [MP4S], that is, the IPMP framework shall be used to signal that the streams are protected.
This section describes specific IPMP Data in the IPMP Descriptor. See MPEG-4 Systems [MP4S] for more details of IPMP framework.

2.3.2.1 IPMP System Type

An IPMP_data is security information and specified by an IPMP system type. The IPMPS_Type field in the IPMP Descriptor SHALL contain the value 42321 (A551h). The IPMP_data specified by the IPMPS_Type of the value 42321 (A551h) is defined in section 2.3.2.2.

2.3.2.2 Protection Information in IPMP_data

When a data stream is protected, some configuration information is necessary to decode the protected stream, such as the identifier of the format of the protected stream, the identifier of the cryptographic key, encryption parameters, etc.
This specification defines the ‘sinf’ structure to contain such configuration information. Additionally, there are several types of formats to describe such protection information, so this specification also defines the structure and semantics to distinguish those types. This enables multiple ‘sinf’ boxes for each format type to be placed in a single container.

The container of this ‘sinf’ boxes is IPMP_data defined by the IPMP framework, the association between the ‘sinf’ box and the stream is represented by the IPMP framework, and the protection information for Marlin Broadband is defined in the following sections.
The syntax of the IPMP_data identified by the IPMPS_Type specified in section 2.3.2.1 is as follows:

```
aligned(8) class IPMP_data()
{
    SecurityInformationDescriptorBox sinf[];
}
```

See section 2.3.2.3 for details of the Security Information Descriptor Box.

2.3.2.3 Security Information Descriptor Box

The protection information in IPMP_data leverages the box structure specified in ISO Base Media File Format [ISOMFF]. As specified in [ISOMFF], Boxes with an unrecognized type or an unrecognized version in case those boxes are extended from FullBox SHALL be ignored and skipped. Note that boxes defined in this section are distinct and independent from those defined in [ISOMFF], although the base data structures are common. For example, values of the box types are assigned and managed independently from those of [ISOMFF].

An overall view of the Security Information Descriptor Box is provided in section 2.3.2.3.
The table shows top-level boxes in the left-hand column. Boxes contained in another box are indented and listed below the container box. Not all the defined boxes need to be used in all 'sinf' boxes; the mandatory boxes are marked with an asterisk (*). See the description of the individual boxes for a discussion of the defaults if the optional boxes are not present.
In order to improve interoperability and utility of the security information, Scheme Type Box SHALL precede Security Scheme Information Box.

Table 2.1 Structure of Security Information Descriptor Box

Box						Section	Description
sinf						2.3.2.3	*Container for protection information*
	schm				*	2.3.2.4	*Type descriptor of the protection information*
	schi				*	2.3.2.5	*Specific type of protection information*

2.3.2.3.1 Definition

Box Type: 'sinf'
Container: IPMP data
Mandatory: Yes
Quantity: One or more (exactly one for the same scheme type)
This box associates a Scheme Type Box and a Security Scheme Information Box.

2.3.2.3.2 Syntax

```
aligned(8) class SecuritInformationDescriptorBox extends
Box('sinf'){
}
```

2.3.2.4 Scheme Type Box

2.3.2.4.1 Definition

Box Type: 'schm'
Container: Security Information Descriptor Box ('sinf')
Mandatory: Yes
Quantity: Exactly one
This box specifies the format type and the version number. Optionally, this box can contain a URL for a security component server with which software components can be updated. The URL, which is in `component_location` field, is optional. The `flags` field indicates the existence of the `component location` field.

If an unknown scheme type in this box is found, the containing Security Information Descriptor Box SHALL be skipped and ignored.

2.3.2.4.2 Syntax

```
aligned(8) class SchemeTypeBox extends FullBox('schm',
version = 0, flags){
    unsigned int(32) scheme_type;
    unsigned int(16) scheme_version;
    if (flags & 0x000001){
         string component_location;
    }
}
```

2.3.2.4.3 Semantics

flags This field is a 24-bit integer containing flags; the following flag is defined:

Table 2.2 Flags

Values	Description
000001h.	Component_location `exists` flag: indicates that the component_location field exists.
others	reserved

In the event that any flags defined here are not used, the `flags` value should be 000000h.

scheme_type This field contains a four-character code that specifies the type of the protection information stored in the Security Scheme Information Box which is associated with this box.

scheme_version This field contains a two-byte value that specifies the version of the format. It provides compatibility of the 'schi' box. The value may be increased when the syntax of the contents of 'schi' for the specified scheme_type has some syntax extension and any parser of an old version for the scheme_type can parse it safely.

component_location This field contains a null-terminated string that specifies a URL of a security component server. The URL shall conform to [RFC1738] using ASCII characters only. The octet encoding of the string shall be UTF-8.

2.3.2.5 Security Scheme Information Box

2.3.2.5.1 Definition

Box Type: 'schi'
Container: Security Information Descriptor Box ('sinf')
Mandatory: Yes
Quantity: Exactly one

This box contains boxes to describe the protection information specified by the Scheme Type Box.

This box has the following structure:

2.3.2.5.2 Syntax

```
aligned(8) class SecuritySchemeInformationBox extends
Box('schi'){
}
```

The contents of this box is dependent on the scheme type of the Scheme Type Box.

The Security Information Descriptor box for Marlin Broadband with the stream encryption defined in section 2.3.3.1 is identified by the following scheme type and version number in Scheme Type Box defined in section 2.3.2.4:

Scheme Type: 'ACBC' (41434243h) or 'ACGK' (4143474bh)

Scheme Version: 1.0 (0100h)

The box structure is shown in Table 2-12 and Table 2-13. When the scheme_type is set to 'ACGK' (4143474bh), the Marlin Group Key Box ('gkey') defined in section 2.3.2.14 SHALL appear in this box in addition to all the boxes defined for 'ACBC' (41434243h) scheme_type. Note that boxes other than those described below may be defined in the future versions of this specification. In such a case, they shall be treated as boxes of unknown types and ignored.

Table 2-12 The Structure of the Security Information Descriptor Box for Marlin Broadband (scheme_type is 'ACBC' (41434243h))

Box						Section	Description
sinf						2.3.2.3	*Container for protection information*
	schm				*	2.3.2.4	*Type descriptor of the protection information*
	schi				*	2.3.2.5	*Specific type of protection information*
		8id			*	2.3.2.6	*Octopus ID Box*
		satr			*	2.3.2.7	*Marlin Security Attributes*
			styp		*	2.3.2.8	*Stream Type*
			sgna			2.3.2.9	*Signed Attributes (optional)*
				rurl		2.3.2.10	*Rights URL (optional)*
			asig			2.3.2.11	*Attribute Signature (optional)*
			cert			2.3.2.12	*Certificate (optional)*
		hmac			*	2.3.2.12	*HMAC value*
		8bdl				2.3.2.13	*Octopus Bundle Box (optional)*

Note that the mandatory boxes are marked with an asterisk (*).

Table 2-13 The Structure of the Security Information Descriptor Box for Marlin Broadband (scheme_type is 'ACGK' (4143474bh))

Box						Section	Description
sinf						2.3.2.3	*Container for protection information*
	schm				*	2.3.2.4	*Type descriptor of the protection information*
	schi				*	2.3.2.5	*Specific type of protection information*
		8id			*	2.3.2.6	*Octopus ID Box*
		gkey			*	2.3.2.14	*Group Key Box (optional)*
		satr			*	2.3.2.7	*Marlin Security Attributes*
			styp		*	2.3.2.8	*Stream Type*
			sgna			2.3.2.9	*Signed Attributes (optional)*
				rurl		2.3.2.10	*Rights URL (optional)*
			asig			2.3.2.11	*Attribute Signature (optional)*
			cert			2.3.2.12	*Certificate (optional)*
		hmac			*	2.3.2.12	*HMAC value*
		8bdl				2.3.2.13	*Octopus Bundle Box (optional)*

Note that the mandatory boxes are marked with an asterisk (*).

2.3.2.6 Octopus ID Box

2.3.2.6.1 Definition

Box Type: ‘8id ’
Container: Scheme Information Box (‘schi’)
Mandatory: Yes
Quantity: Exactly one

2.3.2.6.2 Syntax

```
aligned(8) class OctopusIdBox extends Box(‘8id ’){
    string     id;
}
```

2.3.2.6.3 Semantics

id This field contains a null-terminated UTF-8 characters that indicates Octopus Content Object Identifier. For details on Octopus Content Object Identifiers, refer to Octopus Object.

2.3.2.7 Marlin Security Attributes Box

2.3.2.7.1 Definition

Box Type: ‘satr’
Container: Scheme Information Box (‘schi’)
Mandatory: Yes
Quantity: Exactly one
This box stores security attributes for Marlin Broadband. Note that boxes other than those described below may be defined in the future versions of this specification.

2.3.2.7.2 Syntax

```
aligned(8) class SecurityAttributesBox extends Box(‘satr’){
}
```

2.3.2.8 Marlin Stream Type Box

2.3.2.8.1 Definition

Box Type: ‘styp’
Container: Marlin Security Attributes Box (‘satr’)
Mandatory: Yes
Quantity: Exactly one

2.3.2.8.2 Syntax

```
aligned(8) class StreamTypeBox extends Box(‘styp’){
    string     type;      // URN that indicates the type of
                          // the stream (video or audio)
}
```

2.3.2.8.3 Semantics

type This field contains a null-terminated character string that indicates the type of the track (or stream) that it corresponds to. The following values are defined in this specification. For video and audio, Video steam and Audio steam in the talbe SHALL be used instead of the General stream. For streams of other media types, handler_type value in Handler Reference Box in the Track Box which is associated with IPMP_Descriptor including this Marlin Stream Type Box SHALL be used as <handler_type> in the Table 2-14. When handler_type value includes characters which are not allowed in URI, the percent-encoding mechanism [RFC3986] SHALL be used for such characters. The handler_type is defined in the section 8.9 of [ISOMFF].

Table 2-14 type

Type of Stream	Value
Video stream(*)	urn:marlin:organization:sne:content-type:video
Audio stream	urn:marlin:organization:sne:content-type:audio
General stream	urn:marlin:organization:sne:content-type:<handler_type>
others	reserved

* A Subtitle Graphics stream is treated the same as a video stream.

2.3.2.9 Marlin Signed Attributes Box

2.3.2.9.1 Definition

Box Type: 'sgna'
Container: Marlin Security Attributes Box ('satr')
Mandatory: No
Quantity: Zero or exactly one

This box stores signed attributes for Marlin Broadband. The child boxes of this box SHALL not be accessed unless verification of signature in the Marlin Attribute Signature Box ('asig') succeeds. Note that boxes other than those described below may be defined in future versions of this specification.

2.3.2.9.2 Syntax

```
aligned(8) class SignedAttributesBox extends Box('sgna'){
}
```

2.3.2.10 Marlin Rights URL Box

2.3.2.10.1 Definition

Box Type: 'rurl'
Container: Marlin Signed Attributes Box ('sgna')
Mandatory: No
Quantity: Zero or more

URLs for Marlin License of this Marlin Content are described in this box. Type of the URL is indicated by the `type` field. There SHALL be only one Marlin Rights URL Box ('rurl') with a certain value of the `type` field in the container, Marlin Signed Attributes Box ('sgna'). There MAY be two or more Marlin Rights URL Boxes ('rurl') in the container, if each of them has different value of the `type` field.
Marlin Rights URL Box ('rurl') for Rights URL, which `type` field is set to 01h, SHALL appear in the container after all other types of Marlin Rights URL Boxes ('rurl'). In

case that two or more Marlin Rights URL Boxes (‘rurl’) appear in a Marlin Signed Attributes Box (‘sgna’), the box order indicates the priorities of the URLs. The URLs SHALL be used according to the appearing order.
The URL in this box is used for acquiring a Marlin Action Token, a Marlin License, or an HTML document by sending HTTP GET [RFC2616] to the URL. The MIME-type in the response tells whether a Marlin Action Token or a Marlin License is returned.

In case that the `type` field indicates Rights URL, explicit user consent is REQUIRED before sending HTTP GET. When requesting to the Rights URL, either of a Marlin Action Token, a Marlin License or an HTML document which is defined by a Marlin-adopting system SHALL be returned and appropriately handled.
In case that the `type` field indicates Slient Rights URL or Preview Rights URL, an attempt to acquire a Marlin Action Token or a Marlin License MAY be made automatically, i.e. without user interaction. When requesting to the Silent Rights URL or Preview Rights URL, either of a Marlin Action Token or a Marlin License SHALL be returned, then appropriately handled.

2.3.2.10.2 Syntax

```
aligned(8) class RightsURLBox extends FullBox(‘rurl’, version
= 0, flags = 0){
    unsigned int(8)      type;
    unsigned int(8)      method;
    string               uri;
}
```

2.3.2.10.3 Semantics

type This field contains an interger that indicates the type of the URL. The values defined in this version of the specification are described in Table 2-15.

method This field contains an interger that indicates the method about using the URL. The values and applicable type of URL defined in this version of the specification are described in Table 2-16.

url This field contains a null-terminated string that specifies a URL for Marlin License acquisition. The URL MAY contain a URI template as specified in [MURIT]. The DRM Client MUST support [MURIT]. Note that the minimal mandatory processing required by [MURIT] is to remove the template placeholders (i.e., delimited by a “{“ and “}” character) from the URL or replace it with a “~”.

Table 2-15 Type in Marlin Rights URL Box

Value	Type of URL
00h	Reserved
01h	Rights URL
02h	Slient Rights URL
03h	Preview Rights URL
others	Reserved

Table 2-16 Method in Marlin Rights URL Box

Value	Method	Applicable value of type of URL
00h	no method indicated	01h, 03h

01h	License should be acquired silently, on demand when the content is chosen for playback.	02h
02h	License should be acquired in advance, at the earliest opportunity.	02h
others	Reserved	N/A

2.3.2.11 Marlin Attribute Signature Box

2.3.2.11.1 Definition

Box Type: 'asig'
Container: Marlin Security Attributes Box ('satr')
Mandatory: No
Quantity: Zero or exactly one
In case that Marlin Signed Attributes Box ('sgna') defined in section 2.3.2.9 exists in the same container, this box SHALL exist. The target of the signature contained in this box SHALL be the entire Marlin Signed Attributes Box ('sgna') including size and type fields.

2.3.2.11.2 Syntax

```
aligned(8) class MarlinAttributeSignatureBox extends
FullBox('asig', version = 0, flags = 0){
    unsigned int(32)     algorithm_ID;
    bit(8)               signature[]; // to end of box
}
```

2.3.2.11.3 Semantics

algorithm_ID This field is an interger to indicates the algorithm of the signature. Only 00000001h is defined in this version of the specification and indicates RSA-SHA1 with 2048-bit key [RSA-1_5]. All the other values are reserved.

signature This field contains a signature value generated according to the algorithm indicated by algorithm_ID. The signature SHALL be made by an entity which certificate is issued by a subordinate Content Metadata Certification Service CA as specified in section 9.4.6.3 of [MCS].

2.3.2.12 Marlin Certificate Box

2.3.2.12.1 Definition

Box Type: 'cert'
Container: Marlin Security Attributes Box ('satr')
Mandatory: No
Quantity: Zero or exactly one
This box contains a certificate chain used to verify the signature in the Marlin Attribute Signature Box ('asig').

2.3.2.12.2 Syntax

```
aligned(8) class MarlinCertificateBox extends FullBox('cert',
version = 0, flags = 0){
    bit(8)          certificates[]; //to end of box
}
```

2.3.2.12.3 Semantics

certs This field contains an ordered list of X.509 certificates packaged in a PKIPath [X509Cor1] for the signature in the Marlin Attribute Signature Box ('asig') which resides in the same container.

2.3.2.13 Marlin HMAC Box

2.3.2.13.1 Definition

Box Type: 'hmac'
Container: Scheme Information Box ('schi')
Mandatory: Yes
Quantity: Exactly one

This box stores an HMAC signature on the Marlin Security Attributes Box ('satr'). The target of hmac signature is the entire box ('satr') including size and type fields.
In case the scheme_type of the Scheme Type Box ('schm') is set to 'ACBC' (41434243h), the encryption key used for the corresponding track SHALL be used to calculate the HMAC value. Then encryption keys for those tracks SHALL NOT be identical, if the contents of the Marlin Security Attributes Box in one track are not identical to the ones in another track.
In case the scheme_type of the Scheme Type Box ('schm') is set to 'ACGK' (4143474bh), the key stored in the Marlin Group Key Box ('gkey') in encrypted form SHALL be used to calculate the HMAC value for each track. Then keys for the HMAC of those tracks SHALL NOT be identical, if the contents of the Marlin Security Attributes Box in one track are not identical to the ones in another track.

2.3.2.13.2 Syntax

```
aligned(8) class HMACBox extends Box('hmac'){
    bit(256)   hmac;
}
```

2.3.2.13.3 Semantics

hmac This field contains the hash value of the entire contents of Marlin Security Attributes Box including the size field. It is calculated by HMAC-SHA256 [RFC2104][SHA256]. The encryption key used for the corresponding track shall be used for calculation of this value.

2.3.2.14 Octopus Bundle Box

2.3.2.14.1 Definition

Box Type: '8bdl'
Container: Scheme Information Box ('schi') or License Information Box ('uuid' with the type value of 'LICI')
Mandatory: No

Quantity: Zero or more

2.3.2.14.2 Syntax

```
aligned(8) class OctopusBundleBox extends Box('8bdl'){
    unsigned int(32)     encoding;
    unsigned int(32)     encoding_version;
    bit(8)               bundle_data[];
}
```

2.3.2.14.3 Semantics

encoding This field contains a four-character code that indicates encoding type of the bundle_data field. The following value is defined in this specification.

Table 2-17 Encoding

Values	Description
'xml ' (456D6C20h)	XML encoding used for Marlin Broadband
others	reserved

encoding_version This field contains a four-byte value that indicates the version of the encoding type identified by the encoding field in this box. It shall be se to 00000000h in this version.

bundle_data This field contains data that represents the Octopus objects. For details, refer to Marlin Core System [MCS].

2.3.2.15 Marlin Group Key Box

2.3.2.15.1 Definition

Box Type: 'gkey'
Container: Scheme Information Box ('schi')
Mandatory: No
Quantity: Zero or exactly one

This box contains an encrypted form of the content key directly encrypting the media samples. The content key is encrypted with the key of the Content Key object of the Marlin License bundle which is associated with the Content ID described in the Octopus ID Box ('8id ') in the same Security Scheme Information Box ('schi'). The encryption algorithm is AES Key Wrap Algorithm [RFC3394]. This box MAY appear only when scheme_type of Scheme Type Box ('schm') is set to 'ACGK' (4143474bh).

2.3.2.15.2 Syntax

```
aligned(8) class GroupKeyBox extends FullBox('gkey', 0, 0){
    bit(192)             encrypted_ckey;
}
```

2.3.2.15.3 Semantics

encrypted_ckey This field contains a content key directly encrypting associated media samples. The content key is encrypted with the key in the Marlin License bundle associated with Content ID in Octopus ID Box ('8id ') in the same container. The encryption algorithm of the key is AES Key Wrap Algorithm [RFC3394].

2.3.2.16 License Information Box

2.3.2.16.1 Definition

Box Type: ‘uuid’
Type Value: ‘LICI’
Container: File
Mandatory: No
Quantity: Zero or one

This box is a container for all boxes containing license information. This box is always required when license information is contained in a Marlin broadband content file.

The contents of this box is usually prepared by a content distribution service provider and transferred via such a service. It is prohibited for compliant products to create or modify the contents of this box except when appending this box or the boxes defined in the following sections as it is at the end of this box.

Note that boxes other than those defined in the following sections may be present in this box. In such a case, they shall be treated as boxes of unknown types and ignored.

2.3.2.16.1.1 Syntax

```
aligned(8) class LicenseInformationBox extends Box(‘uuid’,
‘LICI’){
}
```

2.3.3 Stream Encryption

2.3.3.1 AES with 128-bit key in CBC mode

The encryption algorithm is AES [AES] with 128-bit key. The encryption mode is CBC defined in [AES-MODES]. The encryption chaining of CBC mode is exercised per media sample of the ISO Base Media File Format after being padded according to the [RFC2630]. The encrypted data is preceded by initialization vector (IV) defined in [AES-MODES] for CBC mode and forms a media sample consisting of a media stream as shown below.

```
aligned(8) class AESCBCEncryptedSample {
    unsigned int(128) IV;
    unsigned int(8) encrypted_data[];
}
```