INTERNATIONAL STANDARD

ISO/IEC 23008-1

Third edition
2023-01

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 1: MPEG media transport (MMT)

Technologies de l'information — Codage à haute efficacité et livraison des medias dans des environnements hétérogènes —

Partie 1: Transport des médias MPEG

Reference number
ISO/IEC 23008-1:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC 23008-1:2017), which has been technically revised. It also incorporates the Amendment ISO/IEC 23008-1:2017/Amd 1:2017.

The main changes are as follows:

— addition of signalling message for layout configuration;

— addition of signaling messages related to delivery over mobile networks;

— addition of signalling messages to support multipath delivery;

— addition of procedcure for session setup and control;

— addition of procedcure for using WebSockets.

A list of all parts in the ISO/IEC 23008 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document specifies the MPEG media transport (MMT) technologies for the transport and delivery of coded media data for multimedia services over heterogeneous packet-switched networks including internet protocol (IP) networks and digital broadcasting networks. In this document, "coded media data" includes both timed audiovisual media data and non-timed data.

MMT is designed under the assumption that the coded media data will be delivered over a packet-switched delivery network. Several characteristics of such delivery environment, such as non-constant end-to-end delay of each packet from the sending entity to the receiving entity, have been taken into consideration.

For efficient and effective delivery and consumption of coded media data over packet-switched delivery networks, this document provides the following elements:

— the logical model to construct contents composed of components from various sources, for example, components of mash-up applications;

— the formats to convey information about the coded media data, to enable delivery layer processing, such as packetization;

— the packetization method and the structure of the packet to deliver media content over packet-switched networks supporting media and coding independent hybrid delivery over multiple channels;

— the format of the signalling messages to manage delivery and consumption of media content.

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 1: MPEG media transport (MMT)

1 Scope

This document specifies MPEG media transport (MMT) technologies, which include a single encapsulation format, delivery protocols and signalling messages for transport and delivery of multimedia data over heterogeneous packet-switched networks for multimedia services. Types of packet-switched networks supported by this document include bidirectional networks such as Internet Protocol (IP) networks and unidirectional networks such as digital broadcast networks (which may or may not use the IP).

The technologies specified by this document belong to one of three functional areas of MMT: media processing unit (MPU) format, signalling messages and delivery protocol.

The MPU format specifies the "`mpuf`" branded ISO-based media file format (ISOBMFF) encapsulating both timed and non-timed media contents. The MPU format is a self-contained ISOBMFF structure enabling independent consumption of media data, which hides codec-specific details from the delivery function.

The signalling functional area specifies the formats of signalling messages carrying information for managing media content delivery and consumption, e.g. specific media locations and delivery configuration of media contents.

The delivery functional area specifies the payload formats that are independent of media and codec types, which allow fragmentation and aggregation of contents encapsulated as specified by this document for delivery using packet-switched oriented transport protocols. The delivery functional area also provides an application layer transport protocol that allows for advanced delivery of media contents.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12:2022, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 23000-19, *Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax, January 2005*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electroped ia: available at https://www.electropedia.org/

3.1.1
access unit
AU
smallest media data entity to which timing information can be attributed

3.1.2
asset
any multimedia data entity that is associated with a unique identifier and that is used for building a multimedia presentation

3.1.3
dependent asset
asset (3.1.2) for which one or more other assets are necessary for decoding of the contained media content

3.1.4
encoding symbol
unit of data generated by the encoding process

3.1.5
encoding symbol block
set of *encoding symbols* (3.1.4)

3.1.6
FEC code
algorithm for encoding data such that the encoded data flow is resilient to data loss

3.1.7
FEC encoded flow
logical set of flows that consists of an *FEC source flow* (3.1.11) and one or more associated *FEC repair flows* (3.1.9)

3.1.8
FEC payload ID
identifier that identifies the contents of an *MMTP packet* (3.1.20) with respect to the *MMT FEC scheme* (3.1.16)

3.1.9
FEC repair flow
data flow carrying repair symbols to protect an *FEC source flow* (3.1.11)

3.1.10
FEC repair packet
MMTP packet (3.1.20) along with *repair FEC payload identifier* (3.1.27) to deliver one or more *repair symbols* (3.1.29) of a *repair symbol block* (3.1.30)

3.1.11
FEC source flow
flow of *MMTP packets* (3.1.20) protected by an *MMT FEC scheme* (3.1.16)

3.1.12
FEC source packet
MMTP packet (3.1.20) protected by an FEC encoding

3.1.13
media fragment unit
MFU
fragment of a *media processing unit* (3.1.14)

3.1.14
media processing unit
MPU
generic container for independently decodable *timed* (3.1.35) or *non-timed data* (3.1.25) that is media codec agnostic

3.1.15
MMT entity
software and/or hardware implementation that is compliant to a profile of MMT

3.1.16
MMT FEC scheme
forward error correction procedure that defines the additional protocol aspects required to use an FEC scheme in MMT

3.1.17
MMT protocol
MMTP
application layer transport protocol for delivering *MMTP payload* (3.1.22) over IP networks

3.1.18
MMT receiving entity
MMT entity (3.1.15) that receives and consumes media data

3.1.19
MMT sending entity
MMT entity (3.1.15) that sends media data to one or more *MMT receiving entities* (3.1.18)

3.1.20
MMTP packet
formatted unit of the media data to be delivered using the *MMT protocol* (3.1.17)

3.1.21
MMTP packet flow
sequence of *MMTP packets* (3.1.20) with same *MMT sending entity* (3.1.19) and *MMT receiving entity* (3.1.18)

3.1.22
MMTP payload
formatted unit of media data to carry *Packages* (3.1.26) and/or signalling messages using either the *MMT protocol* (3.1.17) or an Internet application layer transport protocols (e.g. RTP).

3.1.23
MMTP session
single *MMTP transport flow* (3.1.24) that is used for certain period of time

3.1.24
MMTP transport flow
series of *MMTP packet flow* (3.1.21) delivered to the same destination

3.1.25
non-timed data
media data that do not have inherent timeline for the decoding and/or presenting of its media content

3.1.26
package
logical collection of media data, delivered using MMT

3.1.27
repair FEC payload ID
FEC payload ID (3.1.8) for repair packets

3.1.28
repair packet block
segmented set of *FEC repair flow* (3.1.9) which can be used to recover lost source packets

3.1.29
repair symbol
encoding symbol that contains redundancy information for error correction

3.1.30
repair symbol block
set of *repair symbols* (3.1.29) which can be used to recover lost *source symbols* (3.1.33)

3.1.31
source FEC payload ID
FEC payload ID (3.1.8) for source packets

3.1.32
source packet block
segmented set of *FEC source flow* (3.1.11) that is to be protected as a single block

3.1.33
source symbol
unit of data to be encoded by an FEC encoding process

3.1.34
source symbol block
set of *source symbols* (3.1.33) generated from a single *source packet block* (3.1.32)

3.1.35
timed data
data that has inherent timeline information for the decoding and/or presentation of its media contents

3.1.36
asset delivery characteristics
ADC
description about required quality of service (QoS) for delivery of *assets* (3.1.2)

Note 1 to entry: ADC is represented by the parameters agnostic to a specific delivery environment.

3.1.37
network abstraction for media
parameter that is used for an interface between media application layer and underlying network layer

3.2 Abbreviated terms

ADC	asset delivery characteristics
AL-FEC	application layer forward error correction
ARQ	automatic repeat request
AU	access unit

AVC	advanced video coding
CLI	cross layer interface
CRI	clock relation information
DCI	device capability information
EVC	essential video coding
GFD	generic file delivery
HRBM	hypothetical receiver buffer model
HTTP	hypertext transfer protocol
ISOBMFF	ISO base media file format
LA-FEC	layer aware forward error correction
LR	license revocation
LS	license signalling
MPI	media presentation information
MC	measurement configuration
MFU	media fragment unit
MMT	MPEG media transport
MMTP	MMT protocol
MP	MMT package
MPU	media processing unit
MTU	maximum transmission unit
MVC	multi-view video coding
NAM	network abstraction for media
NTP	network time protocol
PA	package access
PID	packet identifier
PTP	precision time protocol
RAP	random access point
RTP	real-time protocol
SDP	session description protocol
SI	security information
SSWR	security software request

SVC	scalable video coding
TCP	transmission control protocol
TS	transport stream
UDP	user datagram protocol
URI	uniform resource identifier
URL	uniform resource locator
URN	uniform resource name
UUID	universally unique identifier
UTC	coordinated universal time
XML	extensible mark-up language

4 Conventions

The following convention applies in this document.

— The Big Endian number representation scheme is used.

5 Overview

This document defines a set of tools to enable advanced media transport and delivery services. The tools spread over three different functional areas: media processing unit (MPU) format, delivery and signalling. Even though the tools are designed to be efficiently used together, they may also be used independently regardless of the use of tools from the other functional areas.

The media processing unit (MPU) functional area defines the logical structure of media content, the Package and the format of the data units to be processed by an MMT entity and their instantiation with the ISO base media file format as specified in ISO/IEC 14496-12. The Package specifies the components comprising the media content and the relationship among them to provide necessary information for advanced delivery. The format of data units in this document is defined to encapsulate the encoded media data for either storage or delivery and to allow for easy conversion between data to be stored and data to be delivered (see Clause 7).

The delivery functional area defines an application layer transport protocol and a payload format. The application layer transport protocol defined in this document provides enhanced features for delivery of multimedia data when compared with conventional application layer transport protocols, e.g. multiplexing and support of mixed use of streaming and download delivery in a single packet flow (see 9.2). The payload format is defined to enable the carriage of encoded media data which is agnostic to media types and encoding methods (see 9.3).

The signalling functional area defines formats of signalling messages to manage delivery and consumption of media data. Signalling messages for consumption management are used to signal the structure of the Package (see 10.3) and signalling messages for delivery management are used to signal the structure of the payload format and protocol configuration (see 10.4).

A multimedia service may use any subset of the tools defined in this document according to its specific needs. Furthermore, interfaces between protocols and standards defined by this specification and those defined in other specifications can also be defined and used. Figure 1 illustrates the different functions and their relationships to existing protocols and standards.

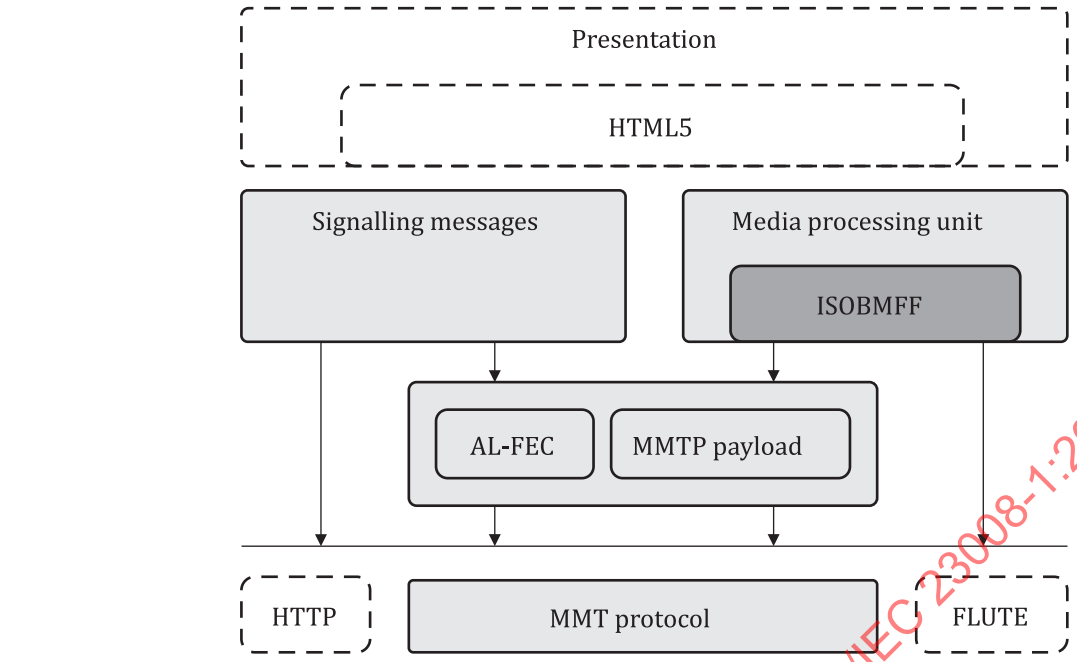


Figure 1 — MMT functional areas, tools and interfaces

Figure 2 depicts the end-to-end architecture for this document. The MMT sending entity is responsible for sending the Packages to the MMT receiving entity as MMTP packet flows. The sending entity may be required to gather contents from content providers based on the presentation information of the Package that is provided by a Package provider.

A Package provider and content providers may be co-located. Media content is provided as an Asset that is segmented into a series of encapsulated MMT Processing Units that forms a MMTP packet flow.

The MMTP packet flow of such content is generated by using the associated transport characteristics information. Signalling messages may be used to manage the delivery and the consumption of Packages.

This document defines the interfaces between the MMT sending entity and the MMT receiving entity, as well as their operations. The MMT sending entity shall conform to the sender operations as defined in Clause 9.

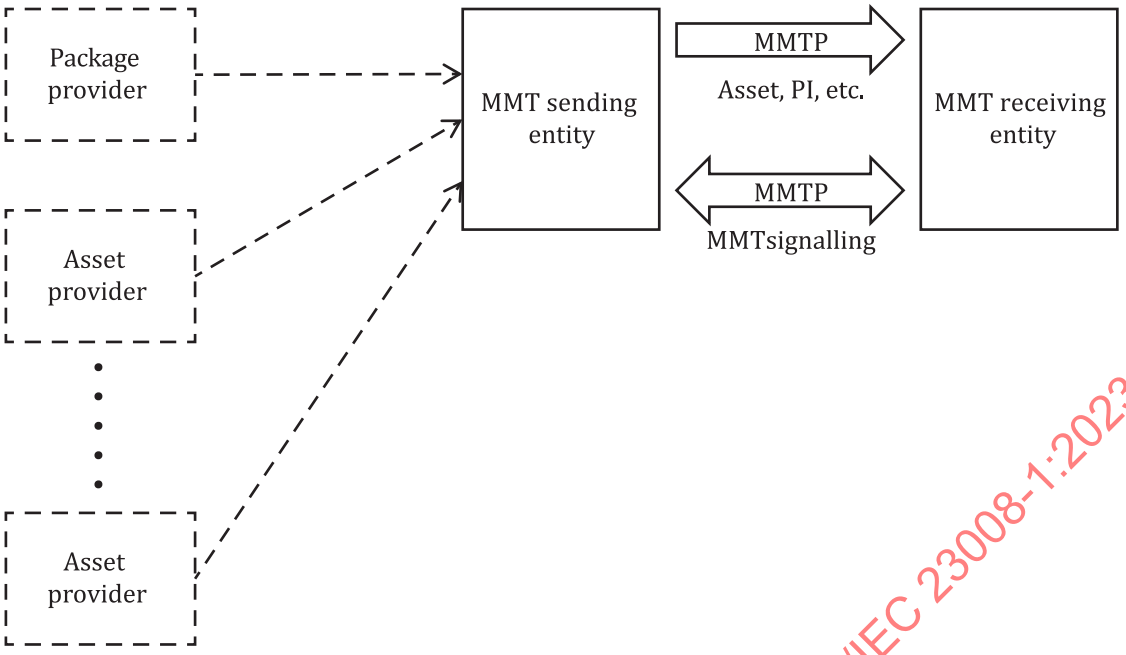


Figure 2 — End-to-end architecture of MMT

An MMT receiving entity operates at one or more MMT functional areas. An exemplary MMT receiving entity architecture is shown in Figure 3.

The MMT protocol (MMTP) is used to receive and de-multiplex the streamed media based on the `packet_id` and the payload type. The de-capsulation procedure depends on the type of payload that is carried and is processed separately and thus, is not shown here.

The presentation engine layer is responsible for setting up the multimedia scene and referencing the content that is received using the MMT protocol.

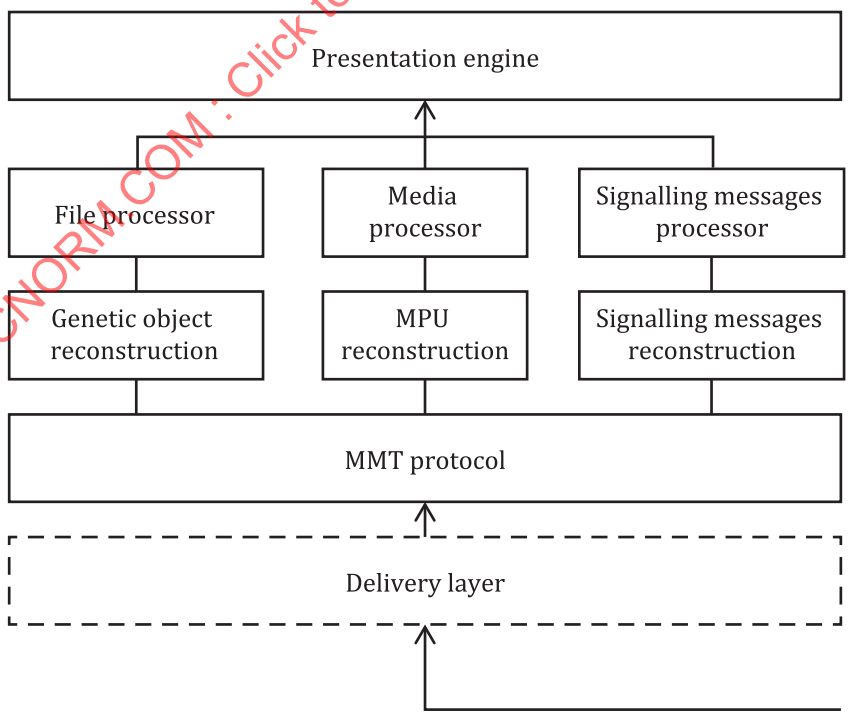


Figure 3 — Example of MMT receiving entity

6 MMT data model

6.1 General

This clause introduces the logical data model assumed for the operation of the MMT protocol. The MMT protocol provides both streaming delivery and download delivery of coded media data. For streaming delivery, MMT protocol assumes the specific data model including MPUs, Assets and Package. The MMT protocol preserves the data model during the delivery by indicating the structural relationships among MPU, Asset and Package using signalling messages.

The collection of the encoded media data and its related metadata builds a Package. The Package may be delivered from one or more MMT sending entities to the MMT receiving entities. Each piece of encoded media data of a Package, such as a piece of audio or video content, constitutes an Asset.

An Asset is associated with an identifier which may be agnostic to its actual physical location or service provider that is offering it, so that an Asset can be globally uniquely identified. Assets with different identifiers shall not be interchangeable. For example, two different Assets may carry two different encodings of the same content but they are not interchangeable.

MMT does not specify a particular identification mechanism but allows the usage of URIs or UUIDs for this purpose. Each Asset has its own timeline which may be of different duration than that of the whole presentation created by the Package.

Each MPU constitutes a non-overlapping piece of an Asset i.e. two consecutive MPUs of the same Asset shall not contain the same media samples. Each MPU may be consumed independently by the presentation engine of the MMT receiving entity.

6.2 Package

As shown in Figure 4, a Package is a logical entity. A Package shall contain one or more presentation information documents such as one specified in ISO/IEC 23008-11, one or more Assets and for each Asset, an associated asset delivery characteristics (ADC). In other words, as processing of a Package is applied per MPU basis and an Asset is a collection of one or more MPUs that share the same Asset ID. It can be also considered that one Package is composed of one presentation information, one or more MPUs and associated ADC for each Asset.

An Asset is a collection of one or more media processing units (MPUs) that share the same Asset ID. An Asset contains encoded media data such as audio or video or a web page. Media data can be either timed or non-timed.

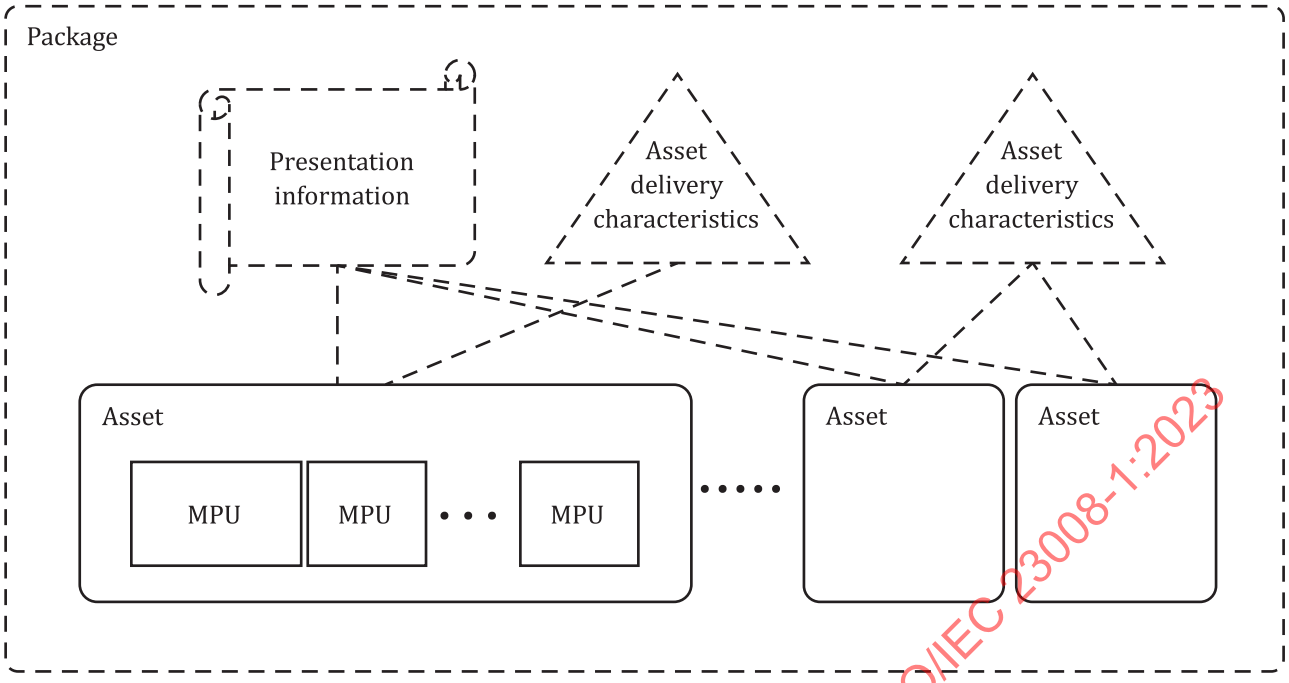


Figure 4 — Overview of Package

Presentation information (PI) documents specify the spatial and temporal relationship among the Assets for consumption. The combination of HTML5 and composition information (CI) documents specified in ISO/IEC 23008-11 is an example of PI documents. In addition, media presentation description (MPD) specified in ISO/IEC 23009-1 can also be used as PI document. A PI document may also be used to determine the delivery order of Assets in a Package. A PI document shall be delivered either as one or more signalling messages defined in this document (see 10.3.3) or as a complete document by some means that is not specified in this document. In the case of broadcast delivery, service providers may decide to carousel PI documents and determine the frequency at which carouseling is to be performed.

Asset delivery characteristics (ADC) shall provide the required QoS information for transmission of Assets. Multiple Assets can be associated with a single ADC. However, a single Asset shall not be associated with multiple ADCs. This information can be used by the entity packetizing the Package to configure the fields of the MMTP payload header and MMTP packet header for efficient delivery of the Assets.

ADC may provide information about an Asset that is relevant for the transport of that Asset.

NOTE Annex D contains a QoS management model for MMT.

6.3 Asset

An Asset is any multimedia data to be used for building a multimedia presentation. An Asset is a logical grouping of MPUs that share the same Asset ID for carrying encoded media data. Encoded media data of an Asset can be either timed data or non-timed data. Timed data are encoded media data that have an inherent timeline and may require synchronized decoding and presentation of the data units at a designated time. Non-timed data are any other type of data that do not have an inherent timeline for decoding and presenting of its media content. The decoding time and the presentation time of each item of non-timed data are not necessarily related to that of other items of the same non-timed data. For example, it can be determined by user interaction or presentation information.

Two MPUs of the same Asset carrying timed media data shall have no overlap in their presentation time.

Any type of data which is referenced by the presentation information is an Asset. Examples of media data types which can be considered as an individual Asset are audio, video, or a web page.

6.4 Media processing unit (MPU)

A media processing unit (MPU) is a media data item that may be processed by an MMT entity and consumed by the presentation engine independently from other MPUs.

Processing of an MPU by an MMT entity includes encapsulation/de-capsulation and packetization/de-packetization. An MPU may include the MMT hint track indicating the boundaries of MFUs for media-aware packetization.

Consumption of an MPU includes media processing (e.g. encoding/decoding) and presentation.

For packetization purposes, an MPU may be fragmented into data units that may be smaller than an Access Unit (AU). The syntax and semantics of MPU are not dependent on the type of media data carried in the MPU.

MPUs of a single Asset shall have either timed or non-timed media.

An MPU may contain a portion of data formatted according to other standards, e.g. MPEG-4 AVC or MPEG-2 TS.

For any Asset with `asset_id X` that depends on Asset with `asset_id Y`, the `m-th` MPU of the Asset with `asset_id X` and the `n-th` MPU of the Asset with `asset_id Y` shall be non-overlapping whenever `m` is not equal to `n`, i.e. no sample in the `m-th` MPU of Asset with `asset_id X` is inside the time interval defined by the sample boundaries of the `n-th` MPU of Asset with `asset_id Y`. Additionally, if the "`sidx`" box is present, the media intervals defined by the "`sidx`" box shall be non-overlapping, i.e. no media sample in the `k-th` media interval (defined by the "`sidx`" box) in an MPU is inside the time interval defined by the sample boundaries of the `j-th` media time interval (defined by the "`sidx`" box) for `j` different from `k`. In the absence of an "`sidx`" box, the concatenation of the `j-th` MPU of Asset with `asset_id Y` with the `j-th` MPU of the Asset with `asset_id X` without its MPU metadata results in a valid MPU. When an "`sidx`" box is present, the concatenation of the `k-th` media interval (defined by the "`sidx`" box) of the `j-th` MPU of Asset with `asset_id Y` with the `k-th` media interval (defined by the "`sidx`" box) of the `j-th` MPU of the Asset with `asset_id X` following the metadata of the MPU with `asset_id Y` results in a valid MPU.

A single MPU shall contain an integral number of AUs or non-timed data. In other words, for timed data, a single AU shall not be fragmented into multiple MPUs. For non-timed data, a single MPU contains one or more non-timed data items to be consumed by presentation engine.

An MPU shall be identified by an associated Asset identification (`asset_id`) and a sequence number.

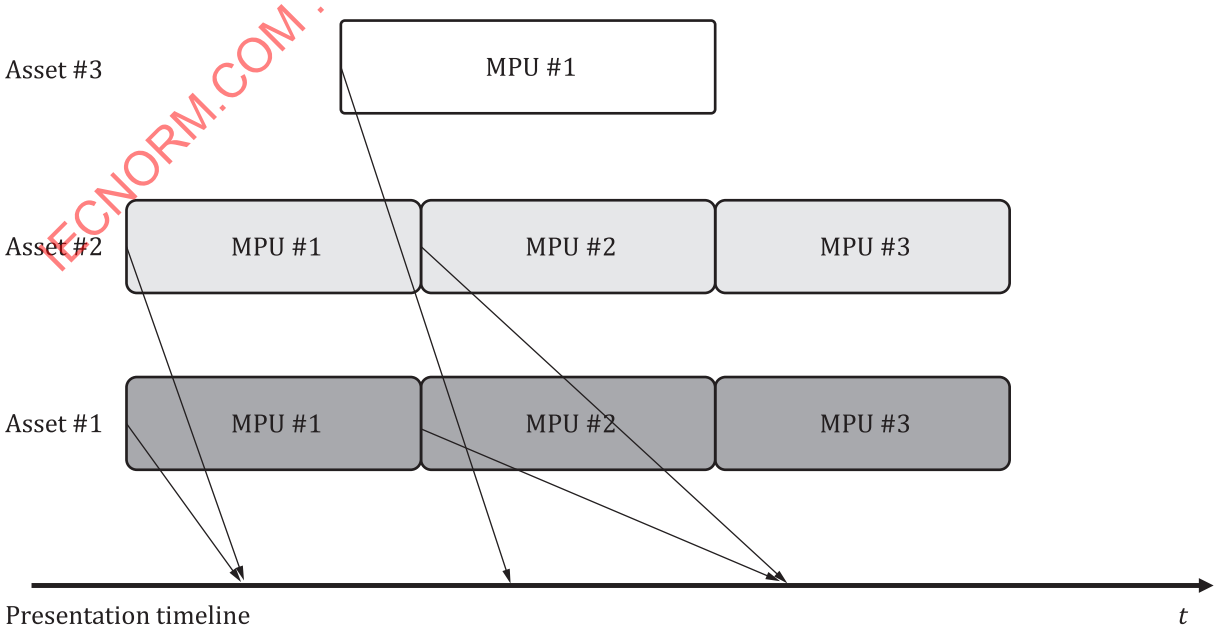


Figure 5 — Example of mapping MPUs to the presentation timeline

An MPU that contains timed media shall have at least one Stream Access Point (SAP) as defined in ISO/IEC 14496-12:2022, Annex I. The first access unit of an MPU shall be an SAP (of SAP type 1, 2, or 3) for processing by an MMT entity. For timed media, this implies that the first AU in the MPU payload is always the first in decoding order. For the MPU containing the data formatted according to other standards, the MPU payload starts with the information necessary for the processing of such a format. For example, if an MPU contains video data, the MPU payload contains one or more groups of pictures and the decoder configuration information is required to process them. For timed media data, the presentation duration and the decoding order and the presentation order of each AU are signalled as part of the fragment metadata. The MPU does not have its initial presentation time. The presentation time of the first AU in an MPU is described by the PI document. The PI document specifies the initial presentation time of each MPU. Figure 5 depicts an example of the timing of the presentation of MPUs from different Assets that are provided by the PI document. In this example, the PI document specifies that the MMT receiving entity shall present MPU #1 of Asset #1 and of Asset #2 simultaneously. At a later point, MPU #1 from Asset #3 is scheduled to be presented. Finally, MPU #2 of Asset #1 and Asset #2 are to be presented in synchronization. The specified presentation time for an MPU defines the presentation time of the first AU in the presentation order of that MPU. If any "`elst`" box is available, the indicated offset shall be applied to the composition time of the first sample in the presentation order of the MPU in addition to the presentation time provided by any presentation information. The presentation time of every MPU shall be provided as part of the presentation information.

An MPU that contains non-timed media data may designate one data item as the primary data item as defined in ISO/IEC 14496-12:2022, 8.11.4.

6.5 Asset delivery characteristics

6.5.1 General

The asset delivery characteristic (ADC) describes the QoS requirements and statistics of Assets for delivery. Each Asset in a Package shall be associated with an ADC. The ADC for each Asset is used by an MMT sending entity to derive the appropriate QoS parameters and the transmission parameters to which a resource reservation and a delivery policy may apply. The ADC is represented in a protocol agnostic format to be generally used by QoS control service entity defined by other standard development organizations, such as IETF, 3GPP, IEEE, etc. It consists of a `QoS_descriptor` element and a `bitstream_descriptor` element. ADC is an XML file that conforms to the schema in 6.5.3. The MIME type of ADC is defined in B.2.

6.5.2 ADC descriptors

6.5.2.1 QoS descriptor

The `QoS_descriptor` element defines required QoS levels on delay and loss for Asset delivery. It consists of `loss_tolerance` attribute, `jitter_sensitivity` attribute, `class_of_service` attribute and `bidirection_indicator` attribute.

6.5.2.2 Bitstream descriptor

The `bitstream_descriptor` element provides the statistics of an Asset. It provides the parameters to implement token bucket traffic shaping such as sustainable rate and buffer size. In addition, peak rate and maximum MFU size represent burstiness of an Asset as shown in Figure 6, where burstiness is defined as a ratio between a `peak_rate` and `sustainable_rate`.

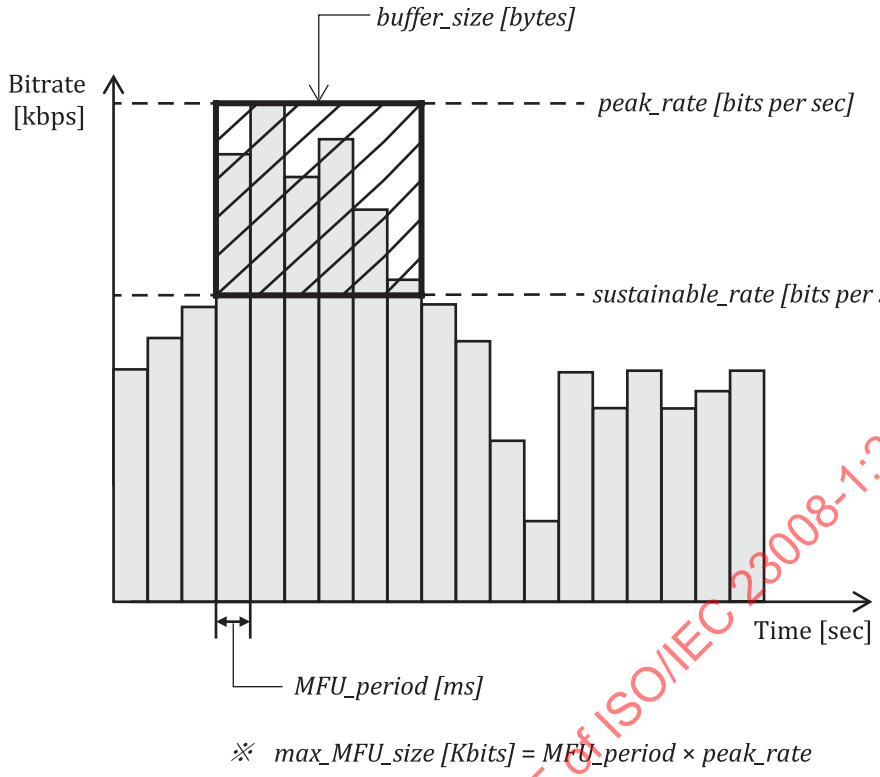


Figure 6 — bitstream_descriptor depicted for a variable bit-rate of Asset

6.5.3 Syntax

```
<complexType name="AssetDeliveryCharacteristic">
    <sequence>
        <element name="QoS_descriptor" type="mmt:QoS_descriptorType" />
        <element name="Bitstream_descriptor" type="mmt:Bitstream_descriptorType"/>
    </sequence>
</complexType>

<complexType name="QoS_descriptorType">
    <attribute name="  loss_tolerance " type="integer"/>
    <attribute name="jitter_sensitivity" type="integer"/>
    <attribute name="class_of_service" type="boolean"/>
    <attribute name="bidirection_indicator" type="boolean"/>
</complexType>

<complexType name="Bitstream_descriptorType">
    <choice>
        <complexType name="Bitstream_descriptorVBRType">
            <attribute name="sustainable_rate" type="float"/>
            <attribute name="buffer_size" type="float"/>
            <attribute name="peak_rate" type="float"/>
            <attribute name="max_MFU_size" type="integer"/>
            <attribute name="mfu_period" type="integer"/>
        </complexType>

        <complexType name="Bitstream_descriptorCBRType">
            <attribute name="peak_rate" type="float"/>
            <attribute name="max_MFU_size" type="integer"/>
            <attribute name="mfu_period" type="integer"/>
        </complexType>
    </choice>
</complexType>
```

6.5.4 Semantics

`loss_tolerance` – indicates the required loss tolerance of the Asset for the delivery. The value of the `loss_tolerance` attribute is listed in Table 1.

Table 1 — Value of loss_tolerance attribute

Value	Description
0	This Asset requires lossless delivery.
1	This Asset allows lossy delivery.

`jitter_sensitivity` – indicates the required jitter level of the underlying delivery network for the Asset delivery between end-to-end. The value of the `jitter_sensitivity` attribute is listed in Table 2.

Table 2 — Value of jitter_sensitivity attribute

Value	Description
0	This Asset requires the preserve time variation between MMTP packets.
1	This Asset does not require the preserve time variation between MMTP packets.

`class_of_service` – classifies the services in different classes and manages each type of bitstream in a particular way. For example, media aware network element (MANE) can manage each type of bitstream in a particular way. This field indicates the type of bitstream attribute as listed in Table 3.

Table 3 — Value of class_of_service attribute

Value	Description
0	The constant bit rate (CBR) service class shall guarantee peak bit rate at any time to be dedicated for transmission of the Asset. This class is appropriate for real-time services which require fixed bit rate such as VoIP without silence suppression.
1	The variable bit rate (VBR) service class shall guarantee sustainable bit rate and allow peak bit rate for the Asset with delay constraints over shared channel. This class is appropriate for most real-time services such as video telephony, video conferencing, streaming service, etc.

`Bidirection_indicator` – If set to "1", bidirectional delivery is required. If set to "0", bidirectional delivery is not required.

`Bitstream_descriptorVBRType` – when `class_of_service` is "1", "`Bitstream_descriptorVBRType`" shall be used for "`Bitstream_descriptorType`".

`Bitstream_descriptorCBRType` – when `class_of_service` is "0" "`Bitstream_descriptorCBRType`" shall be used for "`Bitstream_descriptorType`".

`sustainable_rate` – defines the minimum bit rate that shall be guaranteed for continuous delivery of the Asset. The `sustainable_rate` corresponds to drain rate in token bucket model. The `sustainable_rate` is expressed in bits per second.

`buffer_size` – defines the maximum buffer size for delivery of the Asset. The buffer absorbs excess instantaneous bit rate higher than the `sustainable_rate` and the `buffer_size` shall be large enough to avoid overflow. The `buffer_size` corresponds to bucket depth in the token bucket model. The `buffer_size` of a constant bit rate (CBR) Asset shall be zero. The `buffer_size` is expressed in bytes.

`peak_rate` – defines the peak bit rate during continuous delivery of the Asset. The `peak_rate` is the highest bit rate during every `MFU_period`. The `peak_rate` is expressed in bits per second.

`MFU_period` – defines the period of MFUs during continuous delivery of the Asset. The `MFU_period` is measured as the time interval of sending time between the first byte of two consecutive MFUs. The `MFU_period` is expressed in millisecond.

max_MFU_size – indicates the maximum size of MFU, which is MFU_period*peak_rate. The max_MFU_size is expressed in bytes.

6.6 Bundle delivery characteristics

6.6.1 General

The bundle delivery characteristics (BDC) describe the QoS requirements and statistics of the Bundle for delivery. Each Bundle in a Package shall be associated with a BDC. The BDC for each Bundle is used by an MMT sending entity to derive the appropriate QoS parameters and the transmission parameters to which a resource reservation and a delivery policy may apply. The BDC is represented in a protocol agnostic format to be generally used by QoS control service entity defined by other standard development organizations, such as IETF, 3GPP, IEEE, etc. It consists of a QoS_descriptor element and a bitstream_descriptor element as defined in ADC.

6.6.2 BDC descriptors

6.6.2.1 Qos descriptor

The QoS_descriptor element defines required QoS levels on delay and loss for Bundle delivery. It consists of the loss_tolerance attribute, jitter_sensitivity attribute, class_of_service attribute and bidirection_indicator attribute.

6.6.2.2 Bitstream descriptor

The bitstream_descriptor element provides the statistics of the Bundle. It provides the parameters to implement token bucket traffic shaping such as sustainable rate and buffer size. In addition, peak rate and maximum MFU size represent the burstiness of the Bundle where burstiness is defined as a ratio between a peak_rate and sustainable_rate.

6.6.3 Syntax

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"targetNamespace="mmt">
    <xs:element name="BundleDeliveryCharacteristic"
type="mmt:BundleDeliveryCharacteristicType">
        <xs:attribute name="MMT_package_id" type="xs:string"/>
    </xs:element>

    <xs:complexType name="mmt:BundleDeliveryCharacteristicType">
        <xs:sequence>
            <xs:element name="Bundle" type="mmt:BundleType"
minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="mmt:BundleType">
        <xs:sequence>
            <xs:element name="Element_Asset_id" type="asset_id_T" minOccurs="1">
                <xs:attribute name="Intra_Bundle_Priority" type="xs:integer"/>
            </xs:element>
        </xs:sequence>
        <xs:element name="Bundle_QoS_descriptor" type="mmt:QoS_descriptorType"/>
        <xs:element name="Bundle_Bitstream_descriptor" type="mmt:Bitstream_
descriptorType"/>
        <xs:attribute name="Bundle_id" type="xs:integer"/>
        <xs:attribute name="Inter_Bundle_Priority" type="xs:integer"/>
    </xs:complexType>
    <xs:complexType name="mmt:QoS_descriptorType">
        <xs:attribute name="loss_tolerance" type="xs:integer"/>
        <xs:attribute name="jitter_sensitivity" type="xs:integer"/>
        <xs:attribute name="class_of_service" type="xs:boolean"/>
```

```
        <xs:attribute name="distortion_levels" type="xs:integer"/>
        <xs:attribute name="bidirection_indicator" type="xs:boolean"/>
    </xs:complexType>

    <xs:complexType name="Bitstream_descriptorType">
        <xs:choice>
            <xs:complexType name="Bitstream_descriptorVBRType">
                <xs:attribute name="sustainable_rate" type="xs:float"/>
                <xs:attribute name="buffer_size" type="xs:float"/>
                <xs:attribute name="peak_rate" type="xs:float"/>
                <xs:attribute name="max_MFU_size" type="xs:integer"/>
                <xs:attribute name="mfu_period" type="xs:integer"/>
            </xs:complexType>
            <xs:complexType name="Bitstream_descriptorCBRType">
                <xs:attribute name="peak_rate" type="xs:float"/>
                <xs:attribute name="max_MFU_size" type="xs:integer"/>
                <xs:attribute name="mfu_period" type="xs:integer"/>
            </xs:complexType>
        </xs:choice>
    </xs:complexType>
</xs:schema>
</xml>
```

6.6.4 Semantics

`loss_tolerance` – indicates the required loss tolerance of the Bundle for the delivery. The value of the `loss_tolerance` attribute is listed in Table 4.

Table 4 — Value of loss_tolerance attribute

Value	Description
0	This Bundle requires lossless delivery.
1	This Bundle allows lossy delivery.

`jitter_sensitivity` – indicates the required jitter level of the underlying delivery network for the Bundle delivery between end-to-end. The value of the `jitter_sensitivity` attribute is listed in Table 5.

Table 5 — Value of jitter_sensitivity attribute

Value	Description
0	This Bundle requires the preserve time variation between MMT protocol packets.
1	This Bundle does not require the preserve time variation between MMT protocol packets.

`class_of_service` – classifies the services in different classes and manages each type of bitstream in a particular way. For example, MANE can manage each type of bitstream in a particular way. This field indicates the type of bitstream attribute as listed in Table 6.

Table 6 — Value of class_of_service attribute

Value	Description
0	The constant bit rate (CBR) service class shall guarantee peak bit rate at any time to be dedicated for transmission of the Bundle. This class is appropriate for real-time services which require fixed bit rate such as VoIP without silence suppression.
1	The variable bit rate (VBR) service class shall guarantee sustainable bit rate and allow peak bit rate for the Bundle with delay constraints over a shared channel. This class is appropriate for most real-time services such as video telephony, video conferencing, streaming service, etc.

`Bidirection_indicator` – If set to “1”, bidirectional delivery is required. If set to “0”, bidirectional delivery is not required.

`Bitstream_descriptorVBRType` – when `class_of_service` is "1", "`Bitstream_descriptorVBRType`" shall be used for "`Bitstream_descriptorType`".

`Bitstream_descriptorCBRType` – when `class_of_service` is "0", "`Bitstream_descriptorCBRType`" shall be used for "`Bitstream_descriptorType`".

`sustainable_rate` – defines the minimum bit rate that shall be guaranteed for continuous delivery of the Asset. The `sustainable_rate` corresponds to the drain rate in the token bucket model. The `sustainable_rate` is expressed in bytes per second.

`buffer_size` – defines the maximum buffer size for delivery of the Bundle. The buffer absorbs excess instantaneous bit rate higher than the `sustainable_rate` and the `buffer_size` shall be large enough to avoid overflow. The `buffer_size` corresponds to bucket depth in the token bucket model. The `buffer_size` of a constant bit rate (CBR) Bundle shall be zero. The `buffer_size` is expressed in bytes.

`peak_rate` – defines the peak bit rate during continuous delivery of the Bundle. The `peak_rate` is the highest bit rate during every `MFU_period`. The `peak_rate` is expressed in bytes per second.

`MFU_period` – defines the period of MFUs during continuous delivery of the Bundle. The `MFU_period` is measured as the time interval of sending time between the first byte of two consecutive MFUs. The `MFU_period` is expressed in millisecond.

`max_MFU_size` – indicates the maximum size of MFU, which is `MFU_period*peak_rate`. The `max_MFU_size` is expressed in byte.

`MMT_package_id` – this field is a unique identifier of the Package. This BDC describes delivery characteristics of all the possible Bundles within the scope of this package.

`Element_Asset_id` – is an identifier of asset which is an element of current bundle.

`Bundle_id` – is an identifier of the bundle which distinguishes bundles within the package.

`Intra_Bundle_Priority` – defines the relative priority level among assets within a bundle, which ranges from "0" (highest) to "12" (lowest).

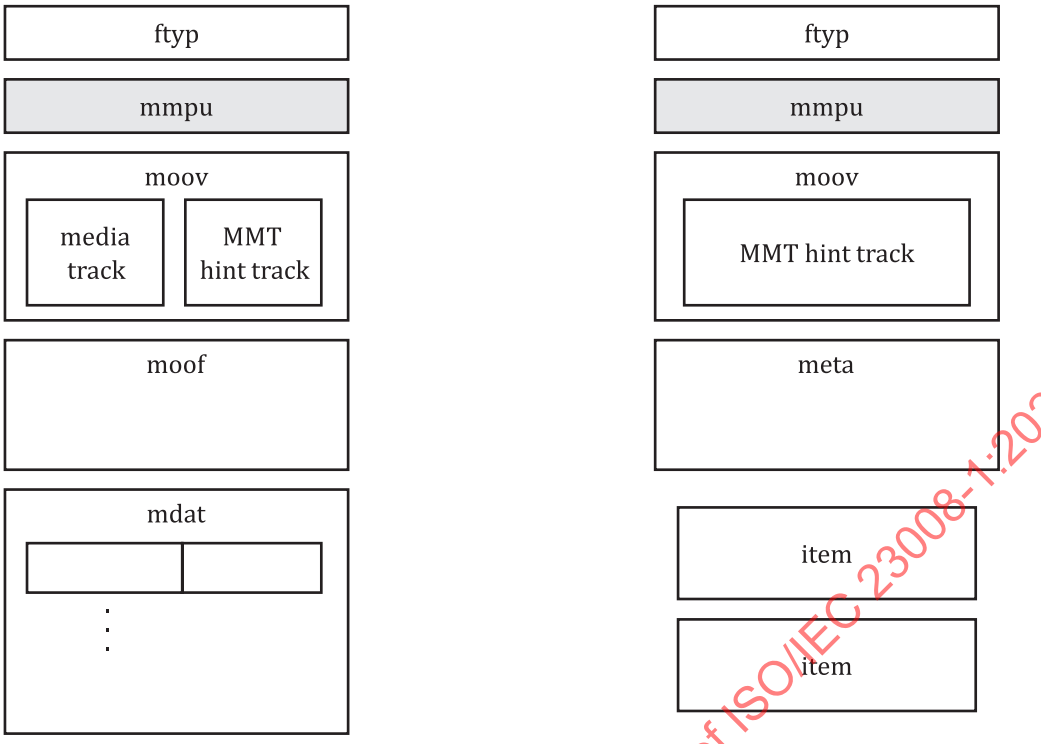
`Inter_Bundle_Priority` – defines the relative priority level among bundles, which ranges from "0" (highest) to "12" (lowest).

7 ISOBMFF-based MPU

7.1 General

An MPU shall be a conformant ISOBMFF file that is generated according to the rules in 7.2. The sequence number and Asset ID of the MPU are provided in the "`mmpu`" box to uniquely identify the MPU encapsulated in the file. Additionally, in case of timed media, a "`sidx`" box may be present to index movie fragments comprising the MPU. The "`moov`" box shall contain all codec configuration information for decoding and presentation of media data.

Timed media data are stored as track of the ISOBMFF (a single media track is allowed). Non-timed media are stored as part of metadata in an ISOBMFF. Figure 7 depicts two examples of MMT encapsulation, one for timed and the other for non-timed media. For packetized delivery of MPU, an MMT hint track provides the information to convert encapsulated MPU to MMTP payloads and MMTP packets.



a) MPU with timed media **b) MPU with non-timed media**

Figure 7 — Examples of MPU encapsulation

7.2 MPU brand definition

The brand "mpuf" (MPU file) defined in this document identifies files that conform to the encapsulation rules for MPU. The "mpuf" brand requires the support of the "isom" brand. Support to the other brand, such as the "dash" brand as defined in ISO/IEC 23009-1:2022, 6.3.5.2, may be indicated.

An MPU file is composed of a set of metadata boxes that enables the MPU to be self-contained. An MPU file shall contain an "ftyp" and "moov" boxes, should contain an "mmpu" box, and may optionally contain an "sidx" box, all of which are part of the MPU metadata. Other boxes are allowed but will be ignored if the parser does not recognize them.

The "moov" box shall contain at most one media track and may contain MMT hint tracks for MFUs. The tracks in the "moov" box shall contain no samples to ensure small overhead (i.e. the entry_count in the "stts", "stsc" and "stco" boxes shall be set to "0"). The "mvex" box shall be contained in the "moov" box for the file storing an MPU with timed media data to indicate that the movie fragment structure is used. The "mvex" box also sets default values for the tracks and samples of the following movie fragments.

Additionally, an "mmpu" box should occur at the file level and the following rules including orders of boxes shall be applied.

a) If present, the "mmpu" box should be placed right after "ftyp" box.

b) For timed media data, zero or more "sidx" boxes may be present in the file and if present, they shall index the movie fragments that build the current MPU.

In addition to the box orders, the following restrictions shall be applied to the "mpuf" brand.

a) The maximum number of independent (e.g. empty "tref" box) media tracks in this file shall be one. Additionally, tracks with non-empty "tref" box (e.g. hint track or SVC/SHVC enhancement layer tracks) may be available.

b) For timed media data, the file shall have at least one movie fragment.

c) For non-timed media data, one "meta" box shall be present at the file level and shall contain the non-timed media items of the MPU.

d) If present, an Edit List Box ("elst") shall only provide an initial offset.

e) Runs of sample data shall be placed in "mdat" box, in decoding order and without any other data in between runs.

f) Any sample auxiliary data, as described by "saio" and "saiz", shall be placed at the beginning of the "mdat" boxes, before any sample data.

g) Any hint data shall be placed after sample data in the "mdat" (or in another mdat placed after sample data) so as not to change sample offsets between before and after transmission.

A "tfdt" box should be present inside the "traf" box of each movie fragment to provide the decode time of the first sample of the movie fragment in decoding order.

If any "elst" box is available, the indicated offset shall be applied to the composition time of the first sample in presentation order of the MPU in addition to the presentation time provided by any presentation information.

Timed media data are stored as a track of the ISOBMFF and indexed by the "moov" and "moof" boxes in a fully backward-compatible manner. An MMT hint track guides the MMT sending entity in converting the file encapsulating MPU into a packetized media stream to be delivered using a transport protocol such as the MMT protocol.

Non-timed media data are stored as metadata items that are described by a "meta" box. The "meta" box shall appear at the file level. Each file of the non-timed media data shall be stored as a separate floating item of the MPU, i.e. it shall not be contained by any box and shall appear after any boxes of the MPU. The entry point to the non-timed media shall be marked as the primary item of the "meta" box (see ISO/IEC 14496-12:2022, 8.11.4).

7.3 MPU box

7.3.1 Definition

Box type: "mmpu"

Container: File

Mandatory: Yes

Quantity: One or more

The media processing unit ("mmpu") box contains the Asset identifier of the Asset to which the current MPU belongs, as well as other information of the current MPU. The Asset identifier is used to uniquely identify the Asset globally. The MPU information includes the sequence number of the MPU in the corresponding Asset.

When it is required to store the ADC together with MPU, it shall be stored in the "meta" box at the file level and its presence shall be indicated through the "is_adc_present" flag and the MIME type of the item that stores the ADC.

7.3.2 Syntax

```
aligned(8) class MPUBox
   extends FullBox('mmpu', version, 0){


   unsigned int(1) is_complete;
```

```
   unsigned int(1) is_adc_present;
   unsigned int(6) reserved;

   unsigned int(32) mpu_sequence_number;


   AssetIdentifier();
}


aligned(8) class AssetIdentifier {
   unsigned int(32) asset_id_scheme;
   unsigned int(32) asset_id_length;
   unsigned int(8)  asset_id_value[asset_id_length];
}
```

7.3.3 Semantics

`is_complete` – indicates whether this MPU has all the media samples and MFUs or not (e.g. when it is being generated from live content).

`mpu_sequence_number` – contains the sequence number of the current MPU. For the first MPU in an Asset, the sequence number shall be "0" and it is incremented by "1" for each following MPU. The sequence number is unique within an Asset.

`asset_id_scheme` – identifies the scheme of Asset ID used in `asset_id_value`. Valid schemes are listed in Table 7. There are many schemes to express identification of content. It is recommended to use scheme-length-value and not to define a new identification scheme. URI scheme for MMT shall be as specified in Annex G.

Table 7 — Value of asset_id_scheme

Value	Description
0x00000000	UUID (universally unique identifier)
0x00000001	URI (uniform resource identifier)

`asset_id_length` – the length of `asset_id_value`.

`asset_id_value` – contains identifier for the Asset. Format of the value in this field is specific to the value in the `asset_id_scheme` field.

`is_adc_present` – indicates whether the ADC is present as an XML box in a "`meta`" box. The MIME type of the ADC file as defined in B.2 shall be indicated in an item information box "`iinf`".

8 MMT hint track

8.1 General

An MMT hint track provides an MMT sending entity with hints for the fragmentation of an MPU into MFUs. An MFU enables media-aware fragmentation of an MPU for transportation purposes. One or more MFUs may then be used to build an MMTP payload. Media data in the MPU are extracted and put on an MMTP payload at transport time by the MMT sending entity. This allows an MMT sending entity to perform fragmentation of MPUs with consideration for media-aware delivery.

An MFU is a media sample or subsample as defined in 9.3.1.

An MMT hint track also provides hints for extracting and building of MFUs for encapsulation using the MMTP payload format. An MMTP payload may contain either MPU metadata, fragment metadata, or one or more MFUs. MPU metadata shall include the "`ftyp`" and "`moov`" boxes, should include the "`mmpu`" box, and may include "`sidx`" and other boxes.

Each MFU is composed of a header and the associated media data. The MFU header shall be a copy of the MFU hint sample and the media data is a copy of the media data that is referenced by that MFU hint sample.

If fragmentation is not required, hint track may be omitted.

An MFU is delivered in an MMTP payload of an MMTP packets.

8.2 Sample description format

8.2.1 Definition

MMT hint tracks are hint tracks with an entry format in the sample description of "mmth" box:

8.2.2 Syntax

```
aligned(8) class MMTHintSampleEntry() extends SampleEntry('mmth') {
   unsigned int(16) hinttrackversion = 1;
   unsigned int(16) highestcompatibleversion = 1;
   unsigned int(16) packet_id;
   unsigned int(1) has_mfus_flag;
   unsigned int(1) is_timed;
   unsigned int(6) reserved;

}
```

8.2.3 Semantics

`packet_id` – is a unique identifier for the Asset for which this hint track applies.

`has_mfus_flag` – is a flag indicating whether the MPUs are fragmented into MFUs or not. If this flag is set to FALSE, the hint track applies to the complete MPU, i.e. each track fragment will have a single sample. Otherwise, each hint sample applies to an MFU.

`is_timed` – indicates whether the media data hinted by this track is timed data or non-timed data.

8.3 Sample format

8.3.1 Definition

Each media sample will be assigned to one or more MFUs. Each sample of the MMT hint track will generate one or more MFUs. The hint sample may omit certain bytes of an MFU if deemed redundant, such as the length field of a NAL unit in the case of AVC or HEVC video bitstreams. For EVC, constraints specified in Annex M shall be applied.

8.3.2 Syntax

```
aligned(8) class MMTHSample {

   unsigned int(32) sequence_number;
   if (is_timed) {
      signed int(8) trackrefindex;
      unsigned int(32) movie_fragment_sequence_number
      unsigned int(32) samplenumber;
      unsigned int(8)  priority;
      unsigned int(8)  dependency_counter;
      unsigned int(32) offset;
      unsigned int(32) length;
      multiLayerInfo();
   } else {
      unsigned int(16) item_ID;
   }
```

```
}


aligned(8) class multiLayerInfo extends Box("muli") {
   bit(1) multilayer_flag;
   bit(7) reserved0;

   if (multilayer_flag==1) {
      bit(3) dependency_id;
      bit(1) depth_flag;
      bit(4) reserved1;
      bit(3) temporal_id;
      bit(1) reserved2;
      bit(4) quality_id;
      bit(6) priority_id;
      bit(10) view_id;
   }
   else{
      bit(6) layer_id;
      bit(3) temporal_id;
      bit(7) reserved3;
   }
}
```

8.3.3 Semantics

`sequence_number` – an integer number that indicates the sequencing order of this MFU within the MPU. Discontinuity of sequence numbers in an MPU is allowed to indicate that certain MFUs (whose sequence number was not in the sequence) were not processed after packetization of MPU. Examples of MFU processing are delivery and caching by underlying network entity.

`movie_fragment_sequence_number` – is the sequence number of the movie fragment to which the media data of this MFU belongs (see ISO/IEC 14496-12:2022, 8.8.5). The `movie_fragment_sequence_number` in an MPU must start by "1" for the first movie fragment of the MPU and shall be incremented by "1" for every succeeding movie fragment in that MPU.

`trackrefindex` – the ID of the media track from which the MFU data is extracted.

`samplenumber` – the number of the sample from which this MFU is extracted. Sample number `n` points to the `n-th` sample from accumulated samples of the current movie fragment. The sample number of the first sample in the movie fragment is set to "1" (see ISO/IEC 14496-12:2022, 8.8.8).

`item_ID` – for non-timed media data, this is the ID of the item that is contained in this MFU.

`priority` – indicates the priority of the MFU relative to other MFUs within an MPU.

`dependency_counter` – indicates the number of MFUs whose decoding is dependent on this MFU. The value of this field is equal to the number of subsequent MFUs in the order of `sequence_number` that may not be correctly decoded without this MFU. For example, if the value of this field is equal to `n`, then `n` subsequent MFUs may not be correctly decoded without this MFU.

`offset` – the offset of the media data contained in this MFU. The offset base is the beginning of the containing "`mdat`" box. MFU shall be placed at the position that offset indicates.

`length` – the length of the data corresponding to this MFU in bytes.

`multilayer_flag` – when set to "1", this flag indicates that detailed multi-layer information is present in this box.

`dependency_id` – dependency identification of this MFU. If the value of this field is a non-zero value, then the data associated with this sample enhance the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).

`depth_flag` – when set to "1", this flag indicates that the given MFU conveys depth data of video.

`quality_id` – quality identification of this MFU. If this field contains a non-zero value, then the data associated with this sample enhance the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).

`priority_id` – priority identification of this MFU. If the value of this field is non-zero value, then the data associated with this sample enhance the video by one or more scalability level in at least on direction (temporal, quality or spatial resolution).

`temporal_id` – temporal identification of this MFU. If the value of this field is a non-zero value, then the data associated with this sample enhance the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).

`view_id` – view identification of this MFU. If the value of this field is a non-zero value, then the data associate with this sample enhance the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).

`layer_id` – indicates the identification of a scalable layer whose information about scalability dimensions is provided in initialization information.

9 Packetized delivery of Package

9.1 General

This clause specifies an application layer transport protocol (the MMT protocol) for packetized delivery of Packages and defines the MMTP payload format.

NOTE For non-packetized delivery of Packages, e.g. using other file delivery protocols, the MMTP payload format and MMT protocol are not required.

The MMT protocol is an application layer transport protocol supporting delivery of Packages over heterogeneous packet-switched delivery networks, including IP-based network environments. The MMT protocol provides enhanced features for delivery of Packages such as protocol level multiplexing, which, for example, enables various Assets to be delivered over a single MMTP packet flow, and delivery timing model independent of presentation time to adapt to a wide range of network jitters and information to support Quality of Service (QoS).

MMT provides a generic media streaming solution that supports the transport of different media types and codecs. For this purpose, MMT defines a transport protocol, MMTP, that is designed to support a limited set of payload types agnostic to the media type or coding format but providing information serving the needs of different multimedia delivery services.

The MMTP payload format is defined as a generic payload format for the packetization of media data components of a Package. It is agnostic to media codecs used for encoded media data, so that any type of media data that are encapsulated as an MPU can be packetized into MMTP. The MMTP payload format is also used to packetize signalling messages. The MMTP payload format also supports fragmentation and aggregation of data to be delivered.

MMTP supports both streaming and download modes, where the streaming mode is optimized for packetized streaming of ISO base media file formatted files (MPU mode) and the download mode allows for flexible delivery of generic files (GFD mode). In addition, MMTP enables delivery of streaming support data such as application layer forward error correction (AL-FEC), repair data and signalling messages. AL-FEC framework shall be as specified in Annex C.

9.2 MMT protocol

9.2.1 General

The MMT protocol is an application layer transport protocol that is designed to efficiently and reliably transport Packages. The MMT protocol can be used for both timed and non-timed media data. It

supports several enhanced features, such as media multiplexing, network jitter calculation, and QoS indication. These features are designed to deliver content composed of various types of encoded media data more efficiently. The MMT protocol may run on top of existing network protocols, e.g. UDP and IP. In this document, the carriage of the data formatted in a format other than the MMTP payload format as specified in 9.3 is not defined.

The MMT protocol is designed to support a wide variety of applications and does not specify congestion control. Congestion control is left to implementation.

The MMT protocol supports the multiplexing of different media data such as MPUs from various Assets over a single MMTP packet flow. It delivers multiple types of data in the order of consumption to the receiving entity to help synchronization between different types of media data without introducing a large delay or requiring large buffer. The MMT protocol also supports the multiplexing of media data and signalling messages within a single packet flow.

A single MMTP payload shall be carried in only one MMTP packet. Fragmentation and aggregation are only provided by the payload format and not by MMTP itself. The MMT protocol defines two packetization modes, generic file delivery (GFD) mode as specified in 9.3.3 and MPU mode as specified in 9.3.2. The GFD mode identifies data units using their byte position inside a transport object. The MPU mode identifies data units using their role and media position inside an MPU. The MMT protocol supports mixed use of packets with two different modes in a single delivery session. A single packet flow of MMT packets can be arbitrarily composed of payloads with two types.

The MMT protocol also provides means to calculate and remove jitter introduced by the underlying delivery network, so that constant end-to-end packet delivery delay can be achieved. By using the timestamp field in the packet header, jitter can be precisely calculated without requiring any additional signalling information and protocols.

The MMT protocol provides priority-related information to enable underlying network layers or the intermediate network entities to map the priority information in the MMTP packet header such as `type_of_bitrate`, `delay_sentivity`, `transmission_priority` and `flow_label` to the network protocol according to predetermined priority mapping policy to support the flow control for MMTP packet delivery. For example, when DiffServ (IETF RFC 2474) is used, this priority information may be used to set the 6-bit DSCP value of the DS field in the IP header. The underlying network entity supporting Diffserv shall then process the IP packets according to the mapping defined by the priority-related information in the MMTP packet header.

9.2.2 Structure of an MMTP packet

The packetization of an MPU that contains timed date is performed in a format aware mode. The type filed of the MMTP packet header is set according to

Table 9 (V=0) and Table 10 (V=1) to indicate that the packetization is format aware and that the packet will be delivering using the MPU mode. An MPU can also be delivered in a format agnostic mode using the GFD mode as described in 9.3.3

Figure 8 and Figure 9 illustrate the structure of an MMTP packet of version 0 and version 1, respectively.

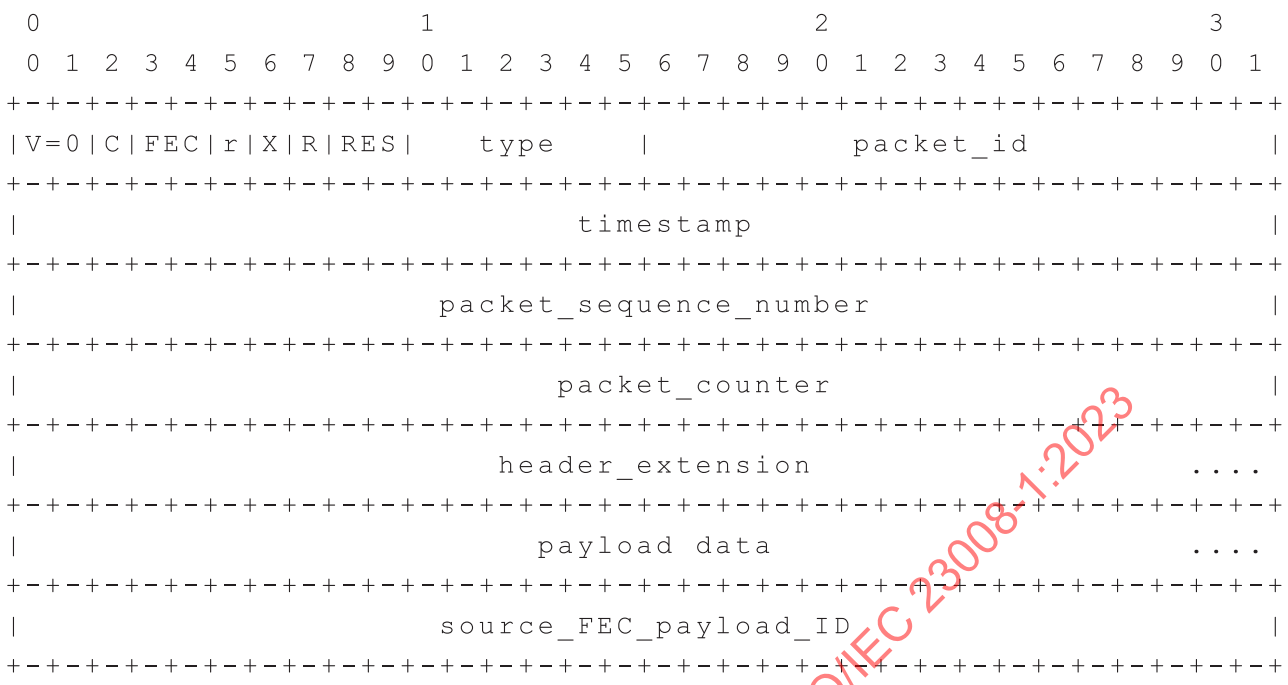


Figure 8 — Structure of MMTP packet (V = 0)

Note that MMTP does not provide an indication of the packet length and relies on lower layer framing protocols to do this. For example, when using MMTP over UDP, the MMTP packet length is provided by the Length field of the UDP datagram.

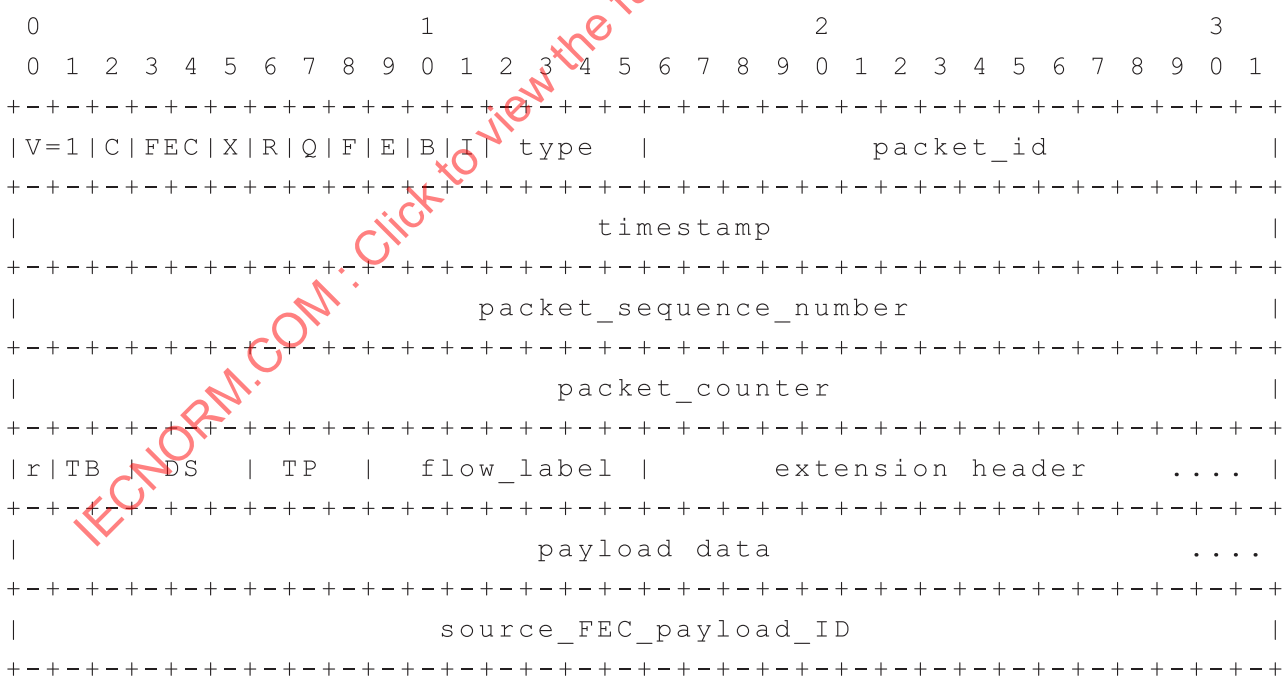


Figure 9 — MMTP packet header, payload and footer (V = 1)

9.2.3 Semantics

`version (V: 2 bits)` – indicates the version number of the MMTP protocol. This field shall be set to "00" to comply with this document and "01" for QoS support.

NOTE If version is set to "01", the length of type is set to 4 bits.

`packet_counter_flag (C: 1 bit)` – "1" in this field indicates that the `packet_counter` field is present.

`FEC_type (FEC: 2 bits)` – indicates the type of the FEC scheme used for error protection of MMTP packets. Valid values of this field are listed in Table 8.

Table 8 — FEC_types

Value	Description
0	MMTP packet without source_FEC_payload_ID field
1	MMTP packet with source_FEC_payload_ID field
2	MMTP packet for repair symbol(s) for FEC Payload Mode 0 (FEC repair packet)
3	MMTP packet for repair symbol(s) for FEC Payload Mode 1 (FEC repair packet)

NOTE If FEC_type is set to 0, it indicates that FEC is not applied to this MMT packet or that FEC is applied to this MMT packet without adding source_FEC_payload_ID.

`reserved (r: 1 bit)` – reserved for future use.

`extension_flag (X: 1 bit)` – when set to "1", this flag indicates that the `header_extension` field is present.

`RAP_flag (R: 1 bit)` – when set to "1", this flag indicates that the payload contains a Random Access Point (RAP) to the data stream of that data type. The exact semantics of this flag are defined by the data type itself.

`reserved (RES: 2 bits)` – reserved for future use.

`Compression_flag (B: 1 bit)` – this field is added at the beginning of the header in order to indicate whether or not header compression is used. When set to "0", the full-size header is used; when set to "1", the reduced-size header is used.

`Indicator_flag (I: 1 bit)` – this field is added to tell the receiver whether or not the current full header will later be used as a reference. This field shall be set to "1" when the full header will be used as a reference. This allows receivers to discover that this header information shall be stored as it will be later used as a reference by packets with reduced headers.

`type(6 / 4 bits)` – this field indicates the type of payload data. Payload type values are defined in Table 9 for version "00" and Table 10 defined value of payload type for version "01".

Table 9 — Data type and definition of data unit (V=0)

Value	Data type	Definition of data unit
0x00	MPU	a media-aware fragment of the MPU
0x01	generic object	a generic object such as a complete MPU or an object of another type
0x02	signalling message	one or more signalling messages or a fragment of a signalling message (see 10.2)
0x03	repair symbol	a single complete repair symbol (see C.4.3)
0x04 ~ 0x1F	reserved for ISO use	for ISO use
0x20 ~ 0x3F	reserved for private use	for private use

Table 10 — Data type and definition of data unit (V=1)

Value	Data type	Definition of data unit
0x0	MPU	a media-aware fragment of the MPU

Table 10 *(continued)*

Value	Data type	Definition of data unit
0x1	generic object	a generic object such as a complete MPU or an object of another type
0x2	signalling message	one or more signalling messages or a fragment of a signalling message (see 10.2)
0x3	repair symbol	a single complete repair symbol (see C.4.3)
0x4 ~ 0x9	reserved for ISO use	for ISO use
0xA ~ 0xF	reserved for private use	for private use

`packet_id (16 bits)` – this field is an integer value that can be used to distinguish one Asset from another. The value of this field is derived from the `asset_id` of the Asset where this packet belongs to. The mapping between the `packet_id` and the `asset_id` is signalled by the MMT Package Table as part of a signalling message (see 10.3.4). Separate value will be assigned to signalling messages and FEC repair flows. The `packet_id` is unique throughout the lifetime of the delivery session and for all MMT flows delivered by the same MMT sending entity. For AL-FEC, the mapping between `packet_id` and the FEC repair flow is provided in the AL-FEC message.

`packet_sequence_number (32 bits)` – an integer value that is used to distinguish packets that have the same `packet_id`. The value of this field starts from arbitrary value and will be incremented by one for each MMTP packet received. It wraps around to "0" after the maximum value is reached. In the FEC repair packet for FEC Payload ID Mode 1, this field shall be replaced with `RS_ID`.

`timestamp (32 bits)` – specifies the time instance of MMTP packet delivery based on UTC. The format is the "short-format" as defined in IETF RFC 5905, NTP version 4, Clause 6. This timestamp specifies the sending time at the first byte of MMTP packet. It is required that an MMT sending entity should provide accurate time information that is synchronized with UTC.

`packet_counter (32 bits)` – an integer value for counting MMTP packets. It is incremented by 1 when an MMTP packet is sent regardless of its `packet_id` value. This field starts from arbitrary value and wraps around to "0" after its maximum value is reached. All packets of an MMTP flow shall have the same setting for `packet_counter_flag (C)`, which means that either all packets will have the `packet_counter` field or none of them will have it.

`Source_FEC_payload_ID (32 bits)` – this field shall be used only when the value of FEC type is set to "1" (see C.4.2). The MMTP packet with FEC type = 1 shall be used for AL-FEC protection for FEC Payload ID Mode 0 and this field shall be added to the MMTP packet after AL-FEC protection.

`header_extension` – this field contains user-defined information. The header extension mechanism is provided to allow for proprietary extensions to the payload format to enable applications and media types that require additional information to be carried in the payload format header. The header extension mechanism is designed in such a way that it may be discarded without impacting the correct processing of the MMTP payload. The header extension shall have the format as shown in Figure 10. This document does not specify any particular header extension.

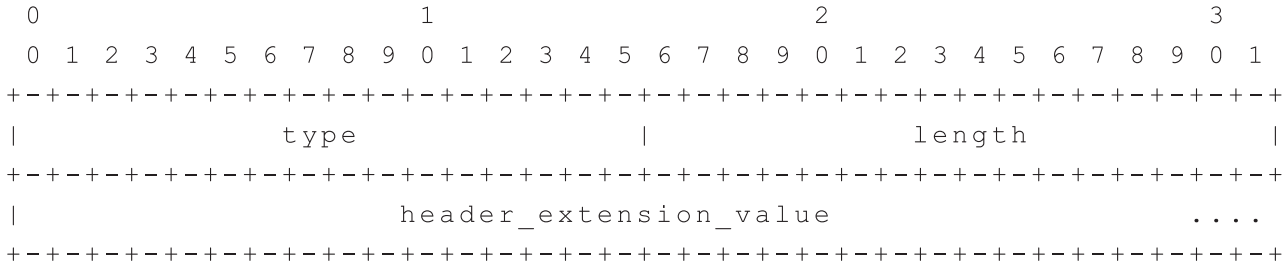


Figure 10 — Structure of extension header

`type (16 bits)` – indicates the unique identification of the following header extension.

length (16 bits) – indicates the length of header_extension_value field in byte.

header_extension_value – provides the extension information. The format of this field is outside the scope of this document.

QoS_classifier_flag (Q:1bit) – when set to "1", it indicates that QoS classifier information is used. The QoS classifier contains the delay_sensitivity field, reliability_flag field, and transmission_priority field. It indicates the QoS class property. The application can perform per-class QoS operations according to the particular value of one property. The class values are universal to all independent sessions.

flow_identifier_flag (F:1 bit) – when set to "1", it indicates that flow identifier information is used. The flow identifier contains the flow_label field and flow_extension_flag field. It indicates the flow identifier. The application can perform per-flow QoS operations in which network resources are temporarily reserved during the session. A flow is defined to be a bitsream or a group of bitstreams whose network resources are reserved according to transport characteristics or ADC in Package.

flow_extension_flag (E: 1 bit) – if there are more than "127" individual flows, this bit is set to "1" and one more byte can be used in extension_header.

reliability_flag (r: 1 bit) – when "reliability_flag" is set to "0", it shall indicate that the data are loss tolerant (e.g. media data), and that the following "transmission_priority" field shall be used to indicate relative priority of loss. When "reliability_flag" is set to "1", the "transmission_priority" field will be ignored, and shall indicate that the data are not loss tolerant (e.g. signalling data, service data or program data).

type_of_bitrate (TB: 2 bits) – indicates the type of bit rate as listed in Table 11.

Table 11 — Value of type_of_bitrate

Value	Description
00	constant bit rate (CBR)
01	non-constant bit rate (nCBR)
10 ~ 11	reserved

delay_sensitivity (DS: 3 bits) – indicates the delay sensitivity of the data between end-to-end delivery for the given service as listed in Table 12. This field is derived from the application as the same content may have different delay requirements in different applications.

Table 12 — Value of delay_sensitivity

Value	Description
111	conversational service (~100 ms)
110	live-streaming service (~1 s)
101	delay-sensitive interactive service (~2 s)
100	interactive service (~5 s)
011	streaming service (~10 s)
010	non-real-time service
001	reserved
000	reserved

transmission_priority (TP: 3 bits) – provides the transmission_priority for the media packet, when the reliability_flag is set to "0". It may be mapped to the NRI of NAL, DSCP of IETF, or other loss priority fields in another network protocol. This field shall take values from "7" ("1112") to "0" ("0002"), where 7 is the highest priority and "0" is the lowest priority.

`flow_label (7 bits)` – indicates the flow identifier. The application can perform per-flow QoS operations in which network resources are temporarily reserved during the session. A flow is defined to be a bitsream or a group of bitstreams whose network resources are reserved according to transport characteristics or ADC in Package. It is an implicit serial number from "0" to "127". An arbitrary number is assigned temporarily during a session and refers to every individual flow for whom a decoder (processor) is assigned and network resource can be reserved.

9.2.4 MMTP session description information

The MMTP session description information may be delivered to receivers in different ways to accommodate different deployment environments. Before a receiver is able to join an MMTP session, the receiver needs to obtain the following information:

— the destination information. In an IP environment, the destination IP address and port number;

— an indication that the session is an MMTP session;

— the version number of the MMT protocol used in the MMTP session;

Additionally, the MMTP session description information should contain the following information:

— the start and end time of the MMTP session.

9.3 MMTP payload

9.3.1 General

The MMTP payload is a generic payload to packetize and carry media data such as MPUs, generic objects, and other information for consumption of a Package using the MMT protocol. The appropriate MMTP payload format shall be used to packetize MPUs, generic objects, and signalling messages described in 10.2.

An MMTP payload may carry complete MPUs or fragments of MPUs, signalling messages, generic objects, repair symbols of AL-FEC schemes, etc. The type of the payload is indicated by the type field in the MMT protocol packet header. For each payload type, a single data unit for delivery as well as a type specific payload header is defined. For example, the fragment of an MPU (e.g. an MFU) is considered as a single data unit when the MMTP payload carries MPU fragments. The MMT protocol may aggregate multiple data units with the same data type into a single MMTP payload. It can also fragment a single data unit into multiple MMTP packets.

An MFU is a sample or subsample of timed data or an item of non-timed data. An MFU shall contain media data that may be smaller than an AU for timed data and the contained media data may be processed by the media decoder. An MFU shall contain an MFU header that contains information on the boundaries of the carried media data. An MFU shall contain an identifier to uniquely distinguish the MFU inside an MPU. It may also provide dependency and priority information relative to other MFUs within the same MPU.

The MMTP payload consists of a payload header and payload data. Some data types may allow for fragmentation and aggregation, in which case a single data unit is split into multiple fragments or a set of data units are delivered in a single MMTP packet.

Each data unit may have its own data unit header depending on the type of the payload. The payload type specific headers are defined in 9.2.2. For types that do not require a payload type, a specific header no payload type header is present and the payload data follows the MMTP header immediately.

Some fields of the MMTP packet header are interpreted differently depending on the payload type. The semantics of these fields will be defined by the payload type in use.

9.3.2 MPU mode

9.3.2.1 General

The delivery of MPUs to MMT receivers using the MMT protocol requires a packetization and depacketization procedure to take place at the MMT sending entity and MMT receiving entity, respectively. The packetization procedure transforms an MPU into a set of MMTP payloads that are then carried in MMTP packets. The MMTP payload format allows for fragmentation of the MMTP payload to enable the delivery of large payloads. It also allows for the aggregation of multiple MMTP payload data units into a single MMTP payload to cater for smaller data units. At the receiving entity, depacketization is performed to recover the original MPU data. Several depacketization modes are defined to address the different requirements of the overlaying applications.

If the payload type is "0x00", the MPU is fragmented in a media-aware way allowing the transport layer to identify the nature and priority of the fragment that is carried. A fragment of an MPU may either be MPU metadata, a Movie Fragment metadata, an MFU, or a non-timed media data item.

9.3.2.2 MMTP payload header for MPU mode

The payload type specific header is provided in Figure 11.

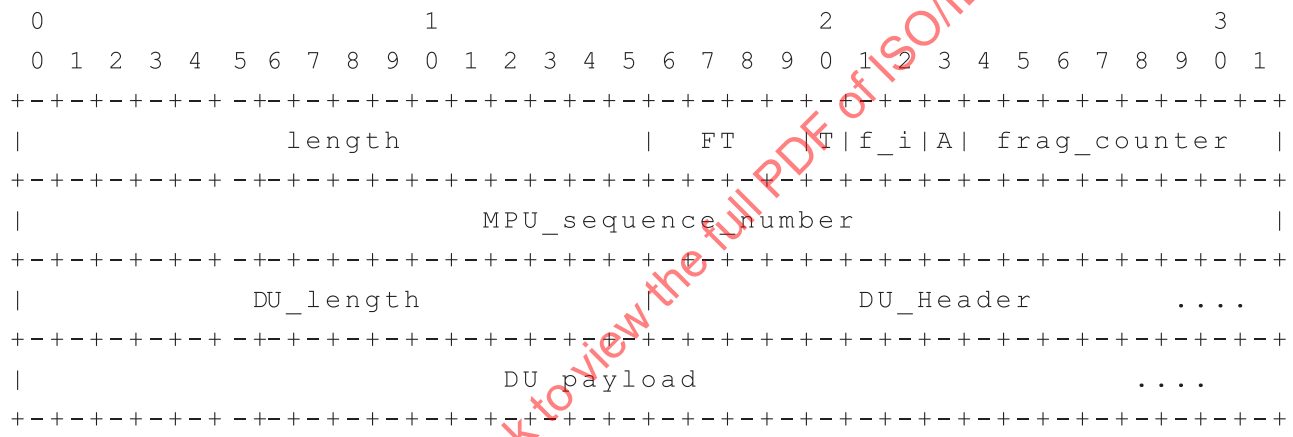


Figure 11 — Structure of the MMTP payload header for MPU mode

For payload that carries an MFU, the DU header is specified depending on the value of the T flag indicating whether the carried data are timed or non-timed media. For timed media (i.e. when the value of T is set to "1"), the DU header fields are shown in Figure 12. For non-timed media (T is set to "0"), the DU header is defined as shown in Figure 13.

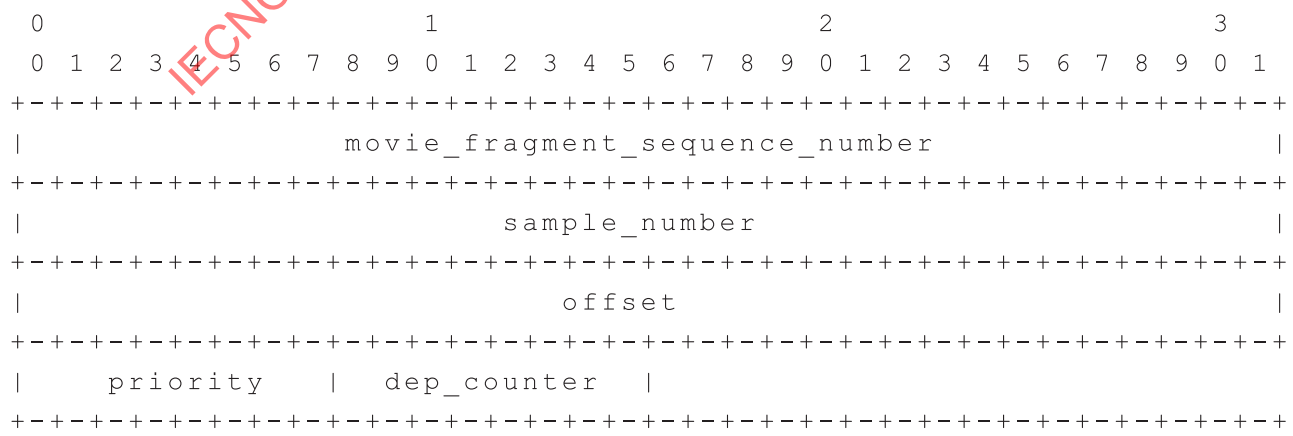


Figure 12 — DU header for timed-media MFU

For non-timed media, the DU header fields are shown in Figure 13.

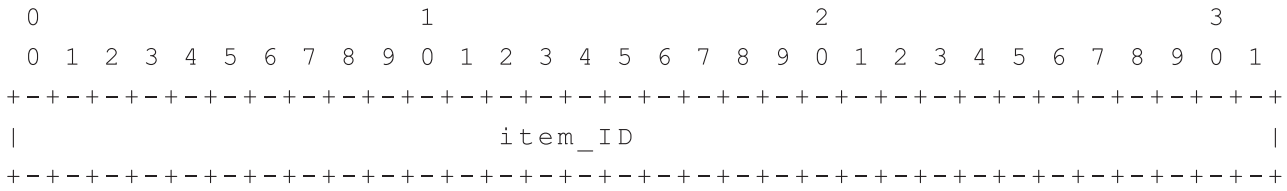


Figure 13 — DU header for non-timed media MFU

9.3.2.3 Semantics

`length (16 bits)` – indicates the length of payload excluding this field in byte.

`MPU Fragment Type (FT: 4 bits)` – this field indicates the fragment type and its valid values are shown in Table 13.

Table 13 — Data type and definition of data unit

FT	Description	Content
0	MPU metadata	contains the ftyp, mmpu, moov, and meta boxes, as well as any other boxes that appear in between.
1	Movie fragment metadata	contains the moof box and the mdat box, excluding all media data inside the mdat box but including any chunks of auxiliary sample information.
2	MFU	contains a sample or subsample of timed media data or an item of non-timed media data.
3 ~ 15	Reserved for private use	reserved

`Timed Flag (T: 1 bit)` – this flag indicates if the fragment is from an MPU that carries timed (value 1) or non-timed media (value 0).

`Fragmentation Indicator (f_i: 2 bits)` – this field indicates that the fragmentation indicator contains information about fragmentation of data unit in the payload. The four values are listed in Table 14. If the value is set to "00", the `aggregation_flag` can be presented.

Table 14 — Values for fragmentation indicator

Value	Description
"00"	Payload contains one or more complete data units.
"01"	Payload contains the first fragment of data unit.
"10"	Payload contains a fragment of data unit that is neither the first nor the last part.
"11"	Payload contains the last fragment of data unit.

The following flags are used to indicate the presence of various information carried in the MMTP payload. Multiple bits can be set simultaneously.

`aggregation_flag (A: 1 bit)` – when set to "1", it indicates that more than 1 data unit is present in the payload, i.e. multiple data units are aggregated.

`fragment_counter` (`frag_count`: 8 bits) – this field specifies the number of payload containing fragments of the same data unit succeeding this MMTP payload. This field shall be "0" if `aggregation_flag` is set to "1".

`MPU_sequence_number (32 bits)` – the sequence number of the MPU to which this MPU fragment belongs.

`DU_length (16 bits)` – this field indicates the length of the data following this field. When `aggregation_flag` is set to "0", this field shall not be present. When `aggregation_flag` is set to "1", this field shall appear as many times as the number of the data units aggregated in the payload and preceding each aggregated data unit.

`DU_Header` – the header of the data unit, which depends on the FT field. A header is only defined for the MFU fragment type, with different semantics for timed and non-timed media as identified by the T flag.

`movie_fragment_sequence_number (32 bits)` – the sequence number of the movie fragment to which the media data of this MFU belongs (see ISO/IEC 14496-12:2022, 8.8.5).

`sample_number (32 bits)` – the sample number of the sample to which the media data of the MFU belongs (see ISO/IEC 14496-12:2022, 8.8.8). The sample number shall be "1" for the first sample of the movie fragment and shall be incremented by "1" for every succeeding sample in order of appearance in the "`mdat`".

`offset (32 bits)` – offset of the media data of this MFU inside the referenced sample.

`subsample_priority (priority: 8 bits)` – provides the priority of the media data carried by this MFU compared with other media data of the same MPU. The value of `subsample_priority` shall be between "0" and "255", with higher values indicating higher priority.

`dependency_counter (dep_counter: 8 bits)` – indicates the number of data units that depend on their media processing upon the media data in this MFU.

`Item_ID (32 bits)` – the identifier of the item that is carried as part of this MFU.

For the FT types "0" and "1", no additional DU header is defined.

9.3.3 Generic file delivery mode

9.3.3.1 General

MMTP also supports the transport of generic files and Assets and uses payload type "`0x01`" as defined in Table 9. An Asset consists of one or more files that are logically grouped and share some commonality for an application, e.g. segments of a dynamic adaptive streaming over HTTP (DASH) Representation, a sequence of MPUs, etc.

In the generic file delivery (GFD) mode, an Asset is transported by using MMTP's GFD payload type.

Each file delivered using the GFD mode requires association of transport delivery information. This includes, but is not limited to, information such as the transfer length.

Each file delivered using the GFD mode may also have associated content-specific parameters such as name, identification, and location of file, media type, size of the file, encoding of the file or message digest of the file. In alignment with HTTP/1.1 protocol as defined in IETF RFC 2616, each file within one generic Asset may have assigned any meta-information about the entity body, i.e. the delivered file. The details are also defined in 9.3.3.2.

9.3.3.2 GFD information

9.3.3.2.1 General

In the GFD mode, each file gets assigned the following parameters.

— `packet_id`: the asset to which each object belongs to. Objects that belong to the same asset are considered as logically connected, e.g. all DASH segments of a Representation and also across Representations that extend over multiple DASH Periods and which carry pieces of the same content.

— `Transport Object Identifier (TOI)`: Each object is associated with a unique identifier within the scope of the `packet_id`.

— `CodePoint`: each object is associated with a CodePoint. A CodePoint associates a specific object and object transport properties. Packets with the same TOI shall have the same CodePoint value. For more details, see Table 16.

NOTE See Annex F for DASH segments over MMTP.

9.3.3.2.2 GFD table

The GFD table provides a list of CodePoints as defined in Table 16. Each CodePoint gets dynamically assigned a CodePoint value. Table 15 shows the structure and semantics of the GFD table.

Table 15 — GFD table

Element or attribute name	Use	Description
GFDTable		the element carries a GFDTable
CodePoint	1...N	defines all CodePoints in the MMTP session
Legend:		
For attributes: M, mandatory; O, Optional; OD, optional with default value; CM, conditionally mandatory.		
For elements: <minOccurs>...<maxOccurs> (N=unbounded).		
Elements are bold; attributes are non-bold and preceded with an @.		

9.3.3.2.3 CodePoints

A CodePoint value can be used to obtain the following information:

— the maximum transfer length of any object delivered with this CodePoint signalling.

In addition, a CodePoint may include the following information:

— the actual transfer length of the objects;

— any information that may be present in the `entity-header` as defined in IETF RFC 2616, section 7.1;

— a `Content-Location-Template` as defined in 9.3.3.2.4 using the TOI and `packet_id` parameter, if present. The TOI and `packet_id` may be used to generate the `Content-Location` for each TOI and `packet_id`. If such a template is present, the processing in 9.3.3.2.4 shall be used to generate the `Content-Location` and the value of the URI shall be treated as the `Content-Location` field in the `entity-header`;

— specific information on the content, for example, how the content is packaged, etc.

Within one session, at most "256" CodePoints may be defined. The definition of CodePoints is dynamically set-up in the MMTP session description.

The CodePoint semantics are described in Table 16.

Table 16 — CodePoint semantics

Element or attribute name	Use	Description
`CodePoint`		Defines the CodePoints in a MMTP session.
Legend:		
For attributes: M, mandatory; O, optional; OD, optional with default value; CM, conditionally mandatory.		
For elements: <minOccurs>...<maxOccurs> (N=unbounded).		
Elements are bold; attributes are non-bold and preceded with an @.		

Table 16 *(continued)*

Element or attribute name	Use	Description
`@value`	M	Defines the value of the CodePoint in the MMTP session as provided in the CodePoint value of the MMTP packet header containing the GFD payload. The value shall be between "1" and "255". The value "0" is reserved.
`@fileDeliveryMode`	**M**	Specifies the file delivery mode according to 9.3.3
`@maximumTransferLength`	M	Specifies the maximum transfer length in bytes of any object delivered with this CodePoint in this MMTP session.
`@constantTransferLength`	OD default: "false"	Specifies if all objects delivered by this CodePoint have constant transfer length. If this attribute is set to TRUE, all objects shall have transfer length as specified in the @maximumTransferLength attribute.
`@contentLocationTemplate`	O	Specifies a template to generate the Content-Location of the entity header.
`EntityHeader`	0 ... 1	Specifies a full entity header in the format as defined in IETF RFC 2616, section 7.1. The entity header applies for all objects that are delivered with the value of this CodePoint.
Legend: For attributes: M, mandatory; O, optional; OD, optional with default value; CM, conditionally mandatory. For elements: <minOccurs>...<maxOccurs> (N=unbounded). Elements are bold; attributes are non-bold and preceded with an @.		

9.3.3.2.4 Content-Location template

A CodePoint may include a `@contentLocationTemplate` attribute. The value of `@contentLocationTemplate` attribute may contain one or more of the identifiers listed in Table 17.

In each URL, the identifiers from Table 17 shall be replaced by the substitution parameter defined in Table 17. Identifier matching is case-sensitive. If the URL contains unescaped $ symbols, which do not enclose a valid identifier, then the result of URL formation is undefined. The format of the identifier is also specified in Table 17.

Each identifier may be suffixed, within the enclosing "$" characters following this prototype:

```
%0[width]d
```

The `width` parameter is an unsigned integer that provides the minimum number of characters to be printed. If the value to be printed is shorter than this number, the result shall be padded with zeros. The value is not truncated even if the result is larger.

The `@contentLocationTemplate` shall be authored such that the application of the substitution process results in valid URIs.

Strings outside identifiers shall only contain characters that are permitted within URLs according to IETF RFC 3986.

Table 17 — Identifiers for URL templates

$<Identifier>$	Substitution parameter	Format
$$	Is an escape sequence, i.e. "$$" is replaced with a single "$".	Not applicable.

Table 17 *(continued)*

$<Identifier>$	Substitution parameter	Format
$PacketID$	This identifier is substituted with the value of the `packet_id` of the associated MMT flow.	The format tag may be present. If no format tag is present, a default format tag with `width = 1` shall be used.
TOI	This identifier is substituted with the `Object Identifier` of the corresponding MMTP packet containing the GFDpayload.	The format tag may be present. If no format tag is present, a default format tag with `width = 1` shall be used.

9.3.3.3 File metadata

9.3.3.3.1 General

Files can be transported using the GFD mode of the MMT protocol. Furthermore, the GFD mode can also be used to transport entities where an entity is defined according to IETF RFC 2616, section 7. An entity consists of meta-information in the form of entity-header fields and content in the form of an entity-body (the file), as described in IETF RFC 2616, section 7.

This enables that files may get assigned information by in-band delivery in a dynamic fashion. For example, it enables the association of a Content-Location, the Content-Size, etc.

The file delivery mode shall be signalled in the CodePoint. The value is also specified in Table 18.

Table 18 — File delivery modes for GFD

Value	Description	Definition
1	The transport object is a file.	in 9.3.3.2
2	The delivered object is an entity consisting of an entity-header and the file.	in 9.3.3.3

9.3.3.3.2 Regular file

In case of the regular file, the object represents a file.

If the CodePoint defined in the GFD table contains `entity-header` fields or `entity-header` fields can be generated, then all of these `entity-header` fields shall apply to the delivered file.

9.3.3.3.3 Regular entity

In case of the regular entity, the object represents an entity as defined in IETF RFC 2616, section 7. An entity consists of `entity-header` fields and an entity-body.

If the CodePoint defined in the GFD table contains `entity-header` fields or `entity-header` fields can be generated, then all of these `entity-header` fields apply to the delivered file. If the `entity-header` field is present in both locations, then the `entity-header` field in the `entity-header` delivered with the object overwrites the one in the CodePoint.

9.3.3.4 MMTP payload header for GFD mode

The GFD mode of MMTP delivers regular files. When delivering regular files, the object represents a file.

If the CodePoint defined in the MMTP session description contains `entity-header` fields or `entity-header` fields can be generated, then all of these `entity-header` fields shall apply to the delivered file.

The payload packets sent using MMTP shall include a GFD payload header and a GFD payload as shown in Figure 14.

In some special cases, a MMT sending entity may need to produce packets that do not contain any payload. This may be required, for example, to signal the end of a session.

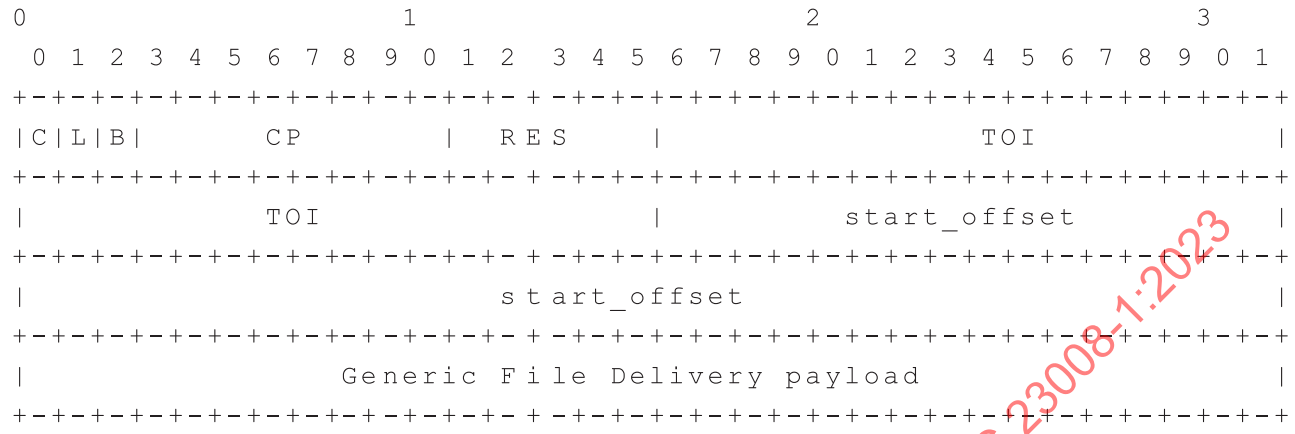


Figure 14 — MMTP payload header for GFD mode

As shown in Figure 14, the GFD payload header has a variable size. Bits designated as "padding" or "reserved" (r) must be set to 0 by the MMT sending entity and ignored by receivers. Unless otherwise noted, numeric constants in this document are in decimal form.

9.3.3.5 Semantics

`C (1 bit)` – indicates that this is the last packet for this session.

`L (1 bit)` – indicates that this is the last delivered packet for this object.

`B (1 bit)` – indicates that this packet contains the last byte of the object.

`CodePoint (CP: 8 bits)` – an opaque identifier that is passed to the packet payload decoder to convey information on the packet payload. The mapping between the CodePoint and the actual codec is defined on a per session basis and communicated out-of-band as part of the session description information.

`RES (5 bits)` – a reserved field that should be set to “0”.

`Transport Object Identifier (TOI: 32 bits)` – the object identifier should be set to a unique identifier of the generic object that is being delivered. The mapping between the object identifier and the object information (such as URL and MIME type) may be done explicitly or implicitly. For example, a sequence of DASH segments may use the segment index as the object identifier and a numerical representation identifier as the `packet_id`. This mapping may also be performed using a signalling message.

`start_offset (48 bits)` – the location of the current payload data in the object.

9.3.4 Signalling message mode

9.3.4.1 General

The signalling message mode of MMTP is defined for the delivery of signalling messages. Signalling messages may be encoded in one of different formats, such as binary format or XML format. It is therefore important to enable quick access and filtering of signalling messages at the transport layer and thus, to avoid parsing the signalling message itself for filtering purposes.

Signalling messages may also be large in size, exceeding the MTU size. Other signalling messages may be much smaller than the MTU. The signalling message payload format provides fragmentation and aggregation functionality to support efficient packetization.

9.3.4.2 MMTP payload header for signalling message mode

Figure 15 depicts the signalling message payload format header.

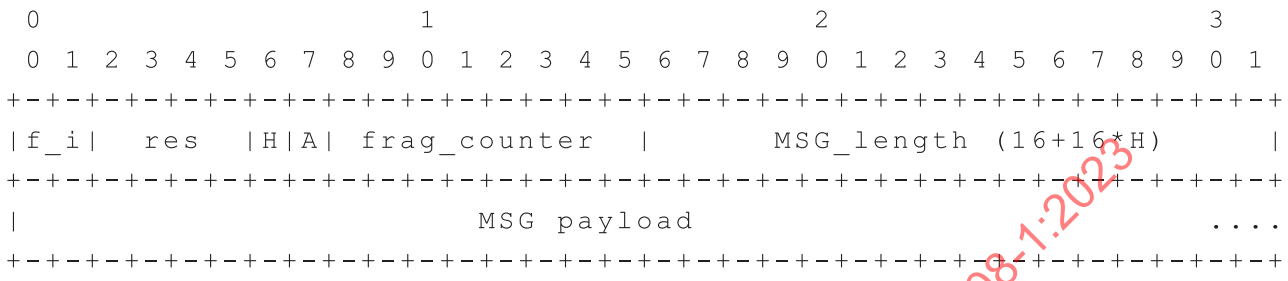


Figure 15 — MMTP payload header for signalling message mode

9.3.4.3 Semantics

The semantic of each field are as follows.

`Fragmentation Indicator (f_i: 2 bits)` – contains information about fragmentation of a signalling message in the MMTP payload. The valid values of this field are shown in Table 19.

Table 19 — Value of fragmentation indicator

Value	Description
00	Payload contains one or more complete signalling messages.
01	Payload contains the first fragment of a signalling message.
10	Payload contains a fragment of a signalling message that is neither the first nor the last fragment.
11	Payload contains the last fragment of a signalling message.

`aggregation_flag (A: 1 bit)` – when set to "1" indicates that more than 1 signalling message is present in the payload, i.e. multiple signalling messages are aggregated.

`fragmentation_counter (frag_count: 8 bits)` – this field specifies the number of payload containing fragments of the same signalling message following the current fragment. This field shall be "0" if `aggregation_flag` is set to "1".

`RES (4 bits)` – this field contains bits that are reserved for future use and shall be set to "0".

`H (1 bit)` – this flag indicates if an additional 16 bit is used to indicate the signalling message length.

`MSG_length (16+16*H bits)` – this field indicates the length of the signalling message following this field. When `aggregation_flag` is set to "0", this field shall not be present. When `aggregation_flag` is set to "1", this field shall appear as many times as the number of the signalling messages aggregated in the payload and preceding each aggregated signalling message.

9.4 MMTP operation

9.4.1 General

In this subclause, the behaviour of an MMT receiving entity and of an MMT sending entity when operating the MMTP protocol using different payload types is described.

An MMTP session consists of one MMTP transport flow. An MMTP transport flow is defined as all packet flows that are delivered to the same destination and which may originate from multiple MMT sending entities. In the case of IP, destination is the IP address and port number.

A single Package may be delivered over one or multiple MMTP transport flows.

A single MMTP transport flow may deliver data from multiple Packages.

An MMTP transport flow may carry multiple Assets. Each Asset is associated with a unique `packet_id` within the scope of the MMTP session. MMTP provides a streaming-optimized mode (the MPU mode) and a file download mode (the GFD mode).

The Asset is delivered as a set of related objects denoted as an object flow. The object may either be an MPU, file or signalling message.

Each object flow shall either be carried in MPU mode or GFD mode; however, the delivery of one Package may be performed using a mix of the two modes, i.e. some Assets may be delivered using the MPU mode and others using the GFD mode.

The MMTP packet sub-flow is the subset of the packets of an MMTP packet flow that share the same `packet_id`. The object flow is transported as an MMTP packet sub-flow.

The MPU mode supports the packetized streaming of an MPU. The GFD mode supports flexible file delivery of any type of file or sequence of files.

MMTP is suitable for unicast, as well as multicast media distribution. To ensure scalability in multicast/broadcast environments, MMTP relies mainly on FEC instead of retransmissions for coping with packet error.

Before joining the MMTP session, the MMT receiving entity should obtain sufficient information to enable reception of the delivered data. This minimum required information is specified in 9.2.4.

MMTP requires the MMT receiving entity to be able to uniquely identify and de-multiplex MMTP packets that belong to a specific object flow. In addition, MMT receiving entities are required to be able to identify packets carrying signalling messages and others carrying AL-FEC repair packets by interpreting the type field of the MMTP packet header.

The MMT receiving entity shall be able to simultaneously receive, de-multiplex, and reconstruct the data delivered by MMTP packets of different types and from different object flows.

A single MMTP packet shall carry exactly one MMTP payload.

9.4.2 Delivering MPUs

9.4.2.1 MMT sending entity operation

9.4.2.1.1 Timed media data

The MPU mode is used to transport MPUs sent by a sending entity to a receiving entity.

The packetization of an MPU that contains timed media may be performed in a MPU format aware mode or MPU format agnostic mode. In the media format agnostic mode, the MPU is packetized into data units of equal size (except for the last data unit, of which the size may differ) or predefined size according to the size of MTU of the underlying delivery network by using GFD mode as specified in 9.3.3. It means that the packetization of the MPU format agnostic mode only considers the size of data to be carried in the packet. The `type` field of the MMTP packet header specified in 9.3.2 is set to "`0x00`" to indicate that the packetization is format agnostic mode.

In the format agnostic mode, the packetization procedure takes into account the boundaries of different types of data in MPU to generate packets by using the MPU mode as specified in 9.3.2. The resulting packets shall carry delivery data units of either MPU metadata, movie fragment metadata, or MFU. The

resulting packets shall not carry more than two different types of delivery data units. The delivery data unit of MPU metadata consists of the "ftyp" box, the "mmpu" box, the "moov" box, and any other boxes that are applied to the whole MPU. The FT field of the MMTP payload carrying a delivery data unit of MPU metadata is set to "0x00". The delivery data unit of movie fragment metadata consists of the "moof" box and the "mdat" box header (excluding any media data). The FT field of the MMTP payload carrying a delivery data unit of movie fragment metadata is set to "0x01". The media data, MFUs in the "mdat" box of the MPU, is then split into multiple delivery data units of MFU in a format aware way. This may, for example, be performed with the help of the MMT hint track. The FT field of the MMTP payload carrying a delivery data unit of MFU is set to "0x02".

Each MFU is prepended with an MFU header, which has the syntax and semantics as defined in 8.3. It is followed by the media data of the MFU.

This procedure is described in Figure 16.

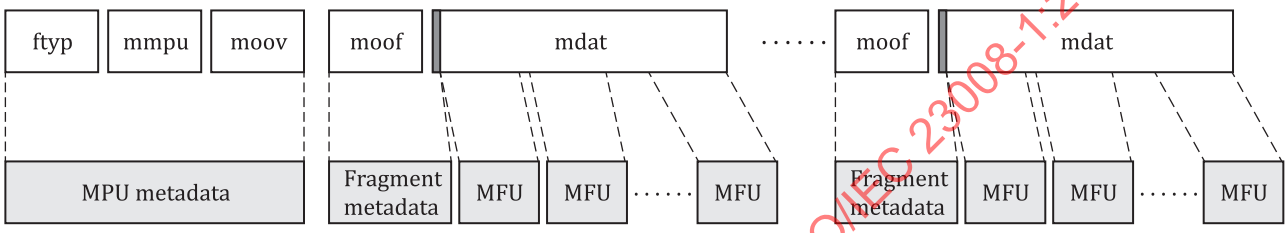


Figure 16 — Payload generation for timed media

9.4.2.1.2 Non-timed media data

The packetization of non-timed media data may also be performed in two different modes. In the MPU format agnostic mode, the MPU is packetized into delivery data units of equal size (except for the last data unit, of which the size may differ) or predefined size according to the size of the MTU of the underlying delivery network by using the GFD mode as specified in 9.3.3. The type field of the MMTP packet header specified in 9.2.2 is set to "0x00" to indicate that the packetization is format agnostic mode.

In the format agnostic mode, the MPU shall be packetized into the packet containing delivery data units of either MPU metadata or MFU by using MPU mode as defined in 9.3.2. The delivery data unit of MPU metadata contains the "ftyp" box, the "moov" box, and the "meta" box and any other boxes that are applied to the whole MPU. The FT field of the MMTP payload carrying a delivery data unit of MPU metadata is set to "0x01". Each delivery data unit of MFU contains a single item of the non-timed media. The FT field of the MMTP payload carrying a delivery data unit of MFU is set to "0x02".

Each item of the non-timed data is then used to build an MFU. Each MFU consists of an MFU header and the item's data. The MFU header is defined in 8.3. This procedure is described in Figure 17.

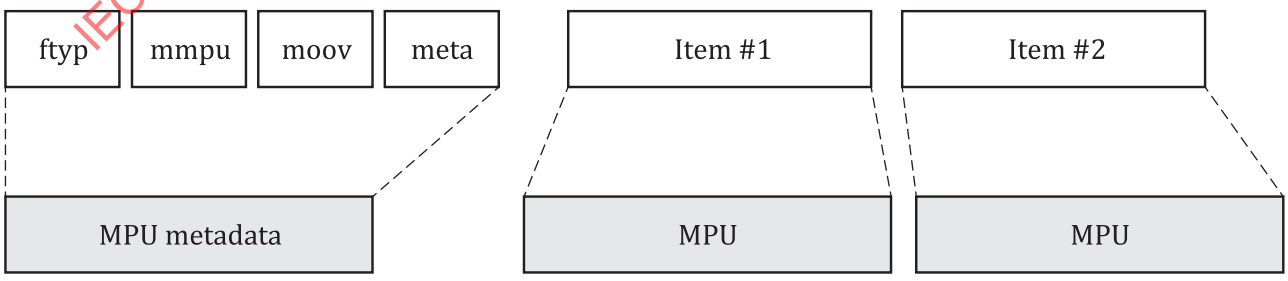


Figure 17 — Payload generation for non-timed media

9.4.2.2 MMT receiving entity operation

The depacketization procedure is performed at the MMT receiving entity to rebuild the transmitted MPU. The depacketization procedure may operate in one of the following modes, depending on the application needs:

— MPU mode: in the MPU mode, the depacketizer reconstructs the full MPU before forwarding it to the application. This mode is appropriate for non-time critical delivery, i.e. the MPU's presentation time as indicated by the presentation information document is sufficiently behind its delivery time.

— Movie Fragment mode: in the Fragment mode, the depacketizer reconstructs a complete fragment including the fragment metadata and the "`mdat`" box with media samples before forwarding it to the application. This mode does not apply to non-timed media. This mode is suitable for delay-sensitive applications where the delivery time budget is limited but is large enough to recover a complete fragment.

— MFU mode: in the media unit mode, the depacketizer extracts and forwards media units as fast as possible to the application. This mode is applicable for very low delay media applications. In this mode, the recovery of the MPU is not required. The processing of the fragment media data is not required but may be performed to resynchronize. This mode tolerates out of order delivery of the fragment metadata MFUs, which may be generated after the media units are generated. This mode applies to both timed and non-timed media.

Using the MFU sequence numbers, it is relatively easy for the receiver to detect missing packets and apply any error correction procedures such as ARQ to recover the missing packets. The payload type may be used by the MMT sending entity to determine the importance of the payload for the application and to apply appropriate error resilience measures.

9.4.3 Delivering generic objects

9.4.3.1 Basic principles for GFD and applications

The files delivered using the GFD mode may have to be provided to an application, for example, presentation information documents or a Media Presentation Description as defined in ISO/IEC 23009-1 may refer to the files delivered using MMTP as GFD objects.

The file shall be referenced through the URI provided or derived from `Content-Location`, either provided in-band as part of an entity header or as part of a GFDT.

In certain cases, the files have an availability start time in the application. In this case, the MMTP session shall deliver the files such that the last packet of the object is delivered such that it is available latest at the receiver at the availability start time as announced in the application.

Applications delivered through the GFD mode may impose additional and stricter requirements on the sending of the files within a MMTP session.

9.4.3.2 MMT sending entity operation

9.4.3.2.1 General

If more than one object is to be delivered using the GFD mode, then the MMT sending entity shall use different TOI fields. In this case, each object shall be identified by a unique TOI scoped by the `packet_id` and the MMT sending entity shall use that TOI value for all packets pertaining to the same object.

The mapping between TOIs and files carried in a session is either provided in-band or in a GFDT.

The GFD payload header as defined in 9.3.3.4 shall be used. The GFD payload header contains a CodePoint field that shall be used to communicate to a MMT receiving entity the settings for information that is

established for the current MMTP session and may even vary during a MMTP session. The mapping between settings and CodePoint values is communicated in the GFDT as described in 9.3.3.2.2.

Let T > 0 be the Transfer-Length of any object in bytes. The data carried in the payload of a packet consists of a consecutive portion of the object.

Then, for any arbitrary X and any arbitrary Y > 0 as long as X + Y is at most T, an MMTP packet may be generated. In this case, the following shall hold:

a) The data carried in the payload of a packet shall consist of a consecutive portion of the object starting from the beginning of byte X through the beginning of byte X + Y.

b) The `start_offset` field in the GFD payload header shall be set to X and the payload data shall be added into the packet to send.

c) If X + Y is identical to T,

   — the payload header flag B shall be set to “1”

   else

   — the payload header flag B shall be set to “0”.

The following procedure is recommended for a MMT sending entity to deliver an object to generate packets containing `start_offset` and corresponding payload data.

a) Set the byte offset counter X to “0”.

b) For the next packets to be delivered, set the length in bytes of a payload to a value Y, which is

   1) reasonable for a packet payload (e.g. ensure that the total packet size does not exceed the MTU),

   2) such that the sum of X and Y is at most T, and

   3) such that it is suitable for the payload data included in the packet.

c) Generate a packet according to the rules 1) to 3) above.

d) If X + Y is equal to T,

   — set the payload header flag B to “1”

   else

   — set the payload header flag B to “0”

   — increment X = X + Y

   — go to 2.

The order of packet delivery is arbitrary, but in the absence of other constraints, delivery with increasing `start_offset` number is recommended.

NOTE The transfer length may be unknown prior to sending earlier pieces of the data. In this case, T can be determined later. However, this does not affect the sending process above.

Additional packets may be sent following the rules in a) to c) from above. In this case, the B flag shall only be set for the payload that contains the last portion of the object.

9.4.3.2.2 GFD payload

The bytes of the object are referenced such that byte 0 is the beginning of the object and byte T-1 is the last byte of the object with T as the transfer length (in bytes) of the object.

The data carried in the payload of an MMTP packet shall consist of a consecutive portion of the object starting from the beginning of byte X and ending at the beginning of byte X + Y where

— X is the value of `start_offset` field in the GFD payload header, and

— Y is the length of the payload in bytes.

Note that Y is not carried in the packet, but framing shall be provided by the underlying transport protocol.

9.4.3.2.3 GFD table delivery

When the GFD mode is used, the GFD table (GFDT) shall be provided. A file that is delivered using the GFD mode, but not described in the GFD table is not considered a "file" belonging to the MMTP session. Any object received with an unmapped CodePoint should be ignored by the MMT receiving entity.

The delivery of the GFD table in the MMT signalling message is defined in 10.5.4.

Other ways of delivery of the GFD table may be possible, but are outside the scope of this document.

9.4.3.3 MMT receiving entity operation

The GFDT may contain one or multiple CodePoints identified by different CodePoint values.

Upon receipt of each GFD payload, the receiver proceeds with the following steps in the order listed.

a) The MMT receiving entity shall parse the GFD payload header and verify that it is a valid header. If it is not valid, then the GFD payload shall be discarded without further processing.

b) The MMT receiving entity shall parse the CodePoint value and verify that the GFDT contains a matching CodePoint. If it is not valid, then the GFD payload shall be discarded without further processing.

c) The MMT receiving entity should process the remainder of the payload, including interpreting the other payload header fields appropriately and using the `start_offset` and the payload data to reconstruct the corresponding object as follows.

   1) The MMT receiving entity can determine from which object a received GFD payload was generated by using the GFDT and by the TOI carried in the payload header.

   2) Upon receipt of the first GFD payload for an object, the MMT receiving entity uses the Maximum Transfer Length received as part of the GFDT to determine the maximum length T′ of the object.

   3) The MMT receiving entity allocates space for the T′ bytes that the object may require.

   4) The MMT receiving entity also computes the length of the payload, Y, by subtracting the payload header length from the total length of the received payload.

   5) The MMT receiving entity allocates a Boolean array RECEIVED[0..T′-1] with all T entries initialized to false to track received object symbols. The MMT receiving entity keeps receiving payloads for the object block as long as there is at least one entry in RECEIVED still set to false or until the application decides to give up on this object.

   6) For each received GFD payload for the object (including the first payload), the steps to be taken to help recover the object are as follows.

      i) Let X be the value of the `start_offset` field in the GFD payload header of the MMTP packet and let Y be the length of the payload, computed by subtracting the MMTP packet and GFD payload header lengths from the total length of the received packet.

ii) The MMTP receiving entity copies the data into the appropriate place within the space reserved for the object and sets RECEIVED[X ... X+Y-1] = true.

iii) If all T entries of RECEIVED are true, then the receiver has recovered the entire object.

7) Once the MMT receiving entity receives a GFD payload with the B flag set to "1", it can determine the transfer length T of the object as X+Y of the corresponding GFD payload and adjust the boolean array RECEIVED[0..T'-1] to RECEIVED[0..T-1].

9.4.4 Header compression for MMTP packet

9.4.4.1 General

Header compression provides the method to reduce the size of the header; techniques such as the Robust Header Compression (RoHC defined in IETF RFC 3095) may be used. While such technique can severely reduce the size of headers, it has two major drawbacks.

— It relies on complex computations/coding techniques (described in protocol stacks profiles) that are quite heavy on the receiver's side.

— It is not a transparent technique and headers need to be entirely decoded, even when it is only to do some filtering and most of the decoded packets are rejected.

It provides two types of headers as follows.

— Full-size headers are sent regularly and may be used as a reference for reduced-size headers.

— Reduced-size headers contain differential information with regard to the last full-size header received that is marked as a reference header.

Therefore, by sending only differential information instead of full information, bits savings can be achieved. Additionally, a link to the reference header packet is added in all compressed packets to make sure that the last full header (reference) packets received is indeed the one that shall be used as a reference. Such robustness mechanism is needed as reference packets may actually be lost.

The header compression method applies on the MMTP packet header. For this, a bit (B) is introduced at the beginning of the original header (or at least in the initial fixed part of the header that is common to the full-size header and reduced-size header) to simply inform whether or not the present header is full-size or reduced size.

Then, many fields present in the full-size headers are either removed or replaced by much smaller fields that contain enough information for the receiver to reconstruct the original full-size header field.

In order to let the receiver know that a full-size header will be used as a reference by a further reduced-size header, an extra bit (I) is also added at the beginning of the full-size header in order to mark headers that will be used as a reference later.

Since packet losses may also happen in the network, it is important that even when reduced-size headers are used, it is still possible to detect and identify packet losses. Thus, a smaller sequence number is introduced for MMTP packets and mandates that the full-size header is used in place of the reduced-size header whenever the reduced sequence number is about to wrap around its initial value.

Similarly, although it is costly in number of bits, the timestamp information associated to packets must be preserved. For this, only the last 19 bits of the full-size 32 bits NTP timestamp are used. This allows keeping the same timestamp precision with a wrap around duration of 8 s. Consequently, a full-size header must be present at least every 8 s. Header compression of the MMTP packet is defined as shown in Figure 18.

9.4.4.2 Syntax

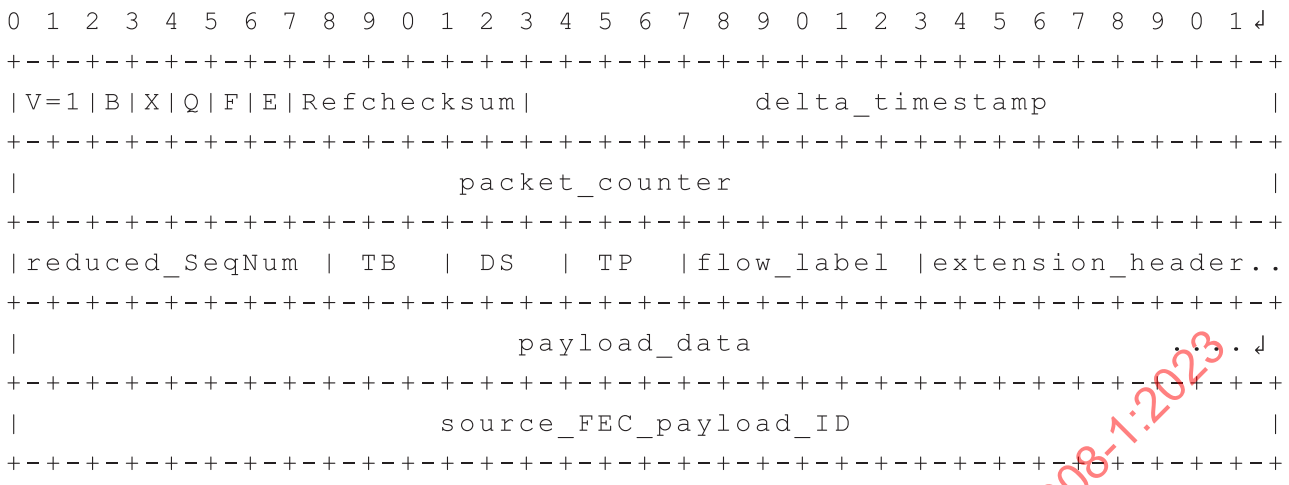


Figure 18 — MMTP packet with reduced header (B = 1)

9.4.4.3 Semantics

The full-size MMTP packet header introduces new fields with their own semantics.

`Compression_flag (B: 1 bit)` – this field is added at the beginning of the header in order to indicate whether or not header compression is used. When set to "0", the full-size header is used; when set to "1", the reduced-size header is used.

`Indicator_flag (I: 1 bit)` – this field is added to tell the receiver whether or not the current full header will later be used as a reference. This field shall be set to "1" when the full header will be used as a reference. This allows receivers to discover that this header information shall be stored as it will be later used as a reference by packets with reduced headers.

The reduced-size MMTP packet header introduces new additional fields with their own semantics.

`Reference_checksum (RefChecksum: 6 bits)` – contains a 6 bits checksum value that allows MMT receiving entity to verify that the last reference MMTP packet received is indeed the one that shall be used for decompressing current MMTP packet header. The checksum value shall be calculated based on the timestamp value and packet sequence number (total 64 bits) value of the MMTP packet with full header to be used as reference. BSD checksum (8) algorithm is used to compute the checksum but `Reference_checksum` field is set to last 6 bits of the result which is 8 bits.

`delta_timestamp (19 bits)` – contains the difference between the timestamp field of the reference full-size header and the value that would be in the current packet timestamp field if the full-size header was used. This difference is coded in a way similar to the 19 least significant bits of an NTP timestamp. If the difference between these two timestamps is larger than 8 s (and therefore goes beyond the maximum duration that can be coded on 19 bits), then a packet with a full header shall be sent in order to provide a new timestamp reference value for further packets with a reduced-size header.

`reduced_sequence_number (reduced_SeqNum: 8 bits)` – contains the 8 least significant bits of the `packet_sequence_number` field that would be in the header if a full-size header were used. Since this new field is coded on 8 bits, reduced decoder shall consider the number of times this field wrapped around 0 to compute the original `packet_sequence_number` value.

The reduced-size MMTP packet header also suppresses the fields that are present in full-size header.

— The `version` field is suppressed as reduced-size headers shall have the same version as their referenced header.

— The `I` field is suppressed, as only full-size headers shall be used as a reference.

— The `RAP_flag` is removed as full-size headers shall be sent whenever the payload contains a random access point.

9.4.4.4 MTP packet header compression rules and normative aspects

A packet with a full MMTP header shall at least be sent when one of the following conditions is met:

a) the difference between the timestamps of the current packet and the reference packet is larger than 8 s (and therefore cannot be coded on the 19 bits long `delta_timestamp` field);

b) `packet_id` is not in the range of 0 to 255;

c) the packet contains a random access point (RAP).

Packet header compression is optional on MMTP sending entities and MMTP receiving entities. Consequently, MMT receivers shall ignore packets with the B field set to "1" if they do not support MMTP header compression.

MMTP receiving entities shall not try to decode a reduced-size header for which the full reference header has not been received, whether because the receiver has just joined the stream or the packet with a full reference header has been lost. MMTP receiving entities shall always wait for packets with a reference header (`I` field set to "1") before they can start or restart (in case of packet loss of reference header) the header decoding.

10 Signalling

10.1 General

This clause specifies the signalling function of MMT. It defines a set of message formats to be used to communicate signalling information necessary for the delivery and consumption of Packages. The delivery of signalling messages using the MMT protocol is also specified in this document. This document specifies the message format for carrying signalling tables, descriptors or the delivery-related information. A signalling table has a set of elements and attributes for specific signalling information. A signalling table may also include descriptors that carry more detailed information. XML representation of the signalling messages shall be as specified in Annex B.

10.2 Signalling message format

10.2.1 General

MMT signalling messages use a general format consisting of three common fields, one specific field (for each signalling message type), and a message payload. A message payload carries signalling information.

The syntax and semantics of a general signalling message format are given in 10.2.2 and 10.2.3, respectively.

10.2.2 Syntax

The syntax of the general format of MMT signalling messages is given in Table 20.

Table 20 — Syntax of the general format of MMT signalling messages

Syntax	Value	No. of bits	Mnemonic
`signalling_message () {`			
`message_id`		16	uimsbf
`version`		8	uimsbf

Table 20 *(continued)*

Syntax	Value	No. of bits	Mnemonic
`if(message_id != PA_message &&`			
`message_id != MPI_message) {`			
length		16	uimsbf
`} else {`			
length		32	uimsbf
`}`			
`extension`			
message_payload {			
`}`			
`}`			

10.2.3 Semantics

`message_id` – this field indicates the identifier of the signalling message. Valid message identifier values are listed in Table 101.

`version` – this field indicates the version of the signalling message. Both the MMT sending entity and MMT receiving entity can verify whether a received message has a new version of not.

`length` – this field indicates the length of the signalling message. This field for all signalling messages except PA messages and MPI message is 2 bytes long. The length of PA messages and MPI messages is 4 bytes long because it is expected that occasionally an MPI table whose length cannot be expressed by a 2 bytes length fields. Also, note that a PA message includes at least one MPI table.

`extension` – this field provides extension information for signalling messages that require extension. The content and length of this field are specified for these signalling messages.

`message_payload` – the payload of the signalling message. The format of this field can be identified by the value of the `message_id` field.

10.3 Signalling messages for Package consumption

10.3.1 General

This subclause specifies signalling messages that are used to exchange signalling information related to Package consumption. As mentioned above, a signalling message may contain signalling tables. Each signalling table carries information on a specific aspect of the Package, such as a Package structure, presentation information document or clock. Signalling messages can aggregate multiple tables for efficient signalling information exchange. For example, an MPI message delivers an MPI table only or an MPI table and a corresponding MP table. The relationship between a message and a table is shown in Figure 19.

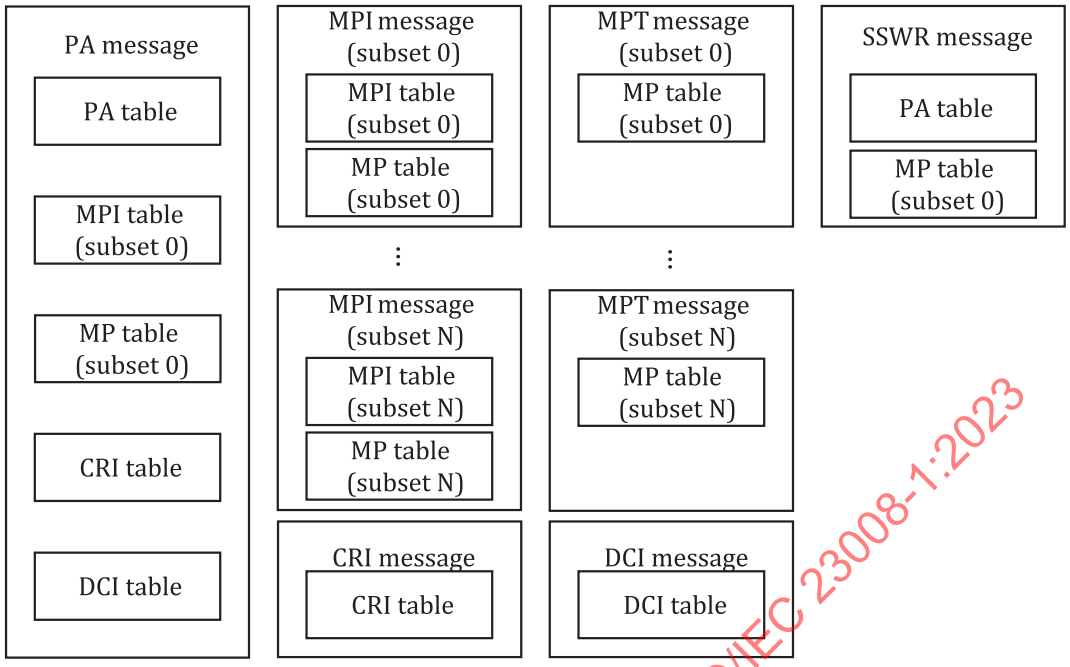


Figure 19 — Structure of the signalling messages and tables for Package consumption

A PA message shall carry a PA table, an MP table (either complete or subset 0), an MPI table (either complete or subset 0) and a DCI table. A PA message may carry a CRI table. An MPI message shall carry an MPI table and may carry an MP table. An MPT message shall carry an MP table. A CRI message shall carry a CRI table. A DCI message shall carry a DCI table. An MPI message and an MPT message can carry several subsets of the complete information needed for efficient delivery and redundancy reduction.

Some signalling tables share the same structure of signalling information. For efficient exchange of those signalling information, a set of descriptors are defined in 10.5.

10.3.2 PA message

10.3.2.1 General

A PA message carries a PA table, which contains information for all other signalling tables for a Package. A PA message also carries an MPI table (either complete or subset 0) and an MP table (either complete or subset 0) for delivery of the minimum information for the processing of the Package.

An MMT receiving entity shall process a PA message before it processes any other signalling messages.

10.3.2.2 Syntax

The syntax of the PA message is defined in Table 21.

Table 21 — PA message syntax

Syntax	Value	No. of bits	Mnemonic
PA_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**32**	**uimsbf**
extension {			
number_of_tables	N1	**8**	**uimsbf**

Table 21 *(continued)*

Syntax	Value	No. of bits	Mnemonic
for (i=0; i<N1; i++) {			
table_id		**8**	**uimsbf**
table_version		**8**	**uimsbf**
table_length		**16**	**uimsbf**
}			
}			
message_payload {			
for (i=0; i<N1; i++) {			
table()			
}			
}			
}			

10.3.2.3 Semantics

`message_id` – indicates the identifier of the PA messages.

`version` – indicates the version of the PA messages.

`length` – a 32-bit field for conveying the length of the PA message in bytes, counting from the beginning of the next field to the last byte of the PA message. The value "0" is not valid for this field.

`number_of_tables` – indicates the number of signalling tables included in this PA message.

`table_id` – indicates the table identifier of a table included in this PA message. It is a copy of the `table_id` field in the table included in the payload of this PA message.

`table_version` – indicates the version of a table included in this PA message. It is a copy of the version field in the table included in the payload of this PA message.

`table_length` – if the table is not an MPI table, it is a copy of the length field in the table included in the `message_payload` of this PA message. If the table is an MPI table, it is the length derived from the length and the extension length part. In this case, the derived length is a value that equals "the actual length" – 5.

`table()` – an MMT signalling table instance. The tables in the payload appear in the same order as the `table_ids` in the extension field. An PA table shall be an instance for `table()`.

10.3.3 MPI message

10.3.3.1 General

An MPI signalling message delivers a whole or a subset of a presentation information document. An MPI message uses an MPI table for encapsulating presentation information documents.

When a subset of a presentation information document is carried by an MPI message, the presentation information document is partitioned into multiple MPI tables. MPI tables for different subsets of a presentation information document shall have different `table_id` values. The `table_id` values are allocated in a contiguous space in increasing order. The MPI table having the lowest `table_id` value provides the base subset among the subsets of a presentation information document and other MPI tables for the remaining subsets of a presentation information document have different `table_id` values. The maximum number of MPI tables for subsets of a PI is 15.

Each MPI message carrying a subset of a presentation information document may have a different transmission period and include the MP table associated with the presentation information document that the MPI message carries.

10.3.3.2 Syntax

The syntax of the MPI message is defined in Table 22.

Table 22 — MPI message syntax

Syntax	Value	No. of bits	Mnemonic
MPI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length	N1	**32**	**uimsbf**
extension {			
reserved	"111 1111"	**7**	**uimsbf**
associated_MP_table_flag		**1**	**bslbf**
}			
message_payload {			
MPI_table()			
if (associated_MP_table_flag) {			
MP_table ()			
}			
}			
}			

10.3.3.3 Semantics

`message_id` – indicates the identification of the MPI message.

`version` – indicates the version of the MPI message.

`length` – indicates the length of the MPI message in bytes, counting from the beginning of the next field to the last byte of the MPI message. The value "0" is not valid for this field.

`associated_MP_table_flag` – if this flag is set to "1", the MPI message also carries an associated MP table. Whenever an associated MP table exists, the associated MP table shall be delivered by a message together with the MPI table. Value "0" explicitly means that the previously stored MP table in the receiving entity can be continuously used with the newly delivered MPI table. The simultaneous delivery of the MPI table and MP table in a single MPI message helps an MMT receiving entity reduce the signalling acquisition time for Package consumption.

`MPI_table()` – an MPI table defined in 10.3.8.

`MP_table()` – an MP table defined in 10.3.9.

10.3.4 MPT message

10.3.4.1 General

The MPT message carries a whole or a subset of an MP table. Subsets of MP tables can be delivered by different MPT messages.

An MPT provides information for a single Package. A single presentation information document may be split into several subsets of such document. For partial delivery of a presentation information document, Assets referenced by a subset of the presentation information document may be described by a subset of an MP table. MP tables associated with different subsets shall have different `table_ids`. Sixteen (16) contiguous values from 17 are assigned to `table_id` values for MP tables. The value of `table_id` "32" (0x20) is assigned for the complete MP table.

In order to support efficient operation of signaling acquisition, a complete MPT or a subset 0 MPT is also found as part of PA messages.

10.3.4.2 Syntax

The syntax of the MPT message is defined in Table 23.

Table 23 — MPT message syntax

Syntax	Value	No. of bits	Mnemonic
`MPT_message () {`			
`  message_id`		16	uimsbf
`  version`		8	uimsbf
`  length`		16	uimsbf
`  message_payload {`			
`  MP_table ()`			
`  }`			
`}`			

10.3.4.3 Semantics

`message_id` – indicates the identifier of the MPT message.

`version` – indicates the version of the MPT message. The MMT receiving entity can check the version of the received message contained in this field.

`length` – indicates the length of the MPT message. The size of this field is 16 bits. The length of the MPT message is in bytes, counting from the beginning of the next field to the last byte of the MPT message. The value "0" is not valid for this field.

`MP_table()` – MP table defined in 10.3.9.

10.3.5 CRI message

10.3.5.1 General

This message carries clock-related information to be used for mapping between the NTP timestamps and MPEG-2 STC.

To achieve synchronization between Assets that use NTP timestamps and MPEG-2 ES that uses the MPEG-2 Presentation Time Stamp (PTS), it is necessary to inform the relationship between the NTP timestamp and the MPEG-2 STC to an MMT receiving entity by periodically delivering values of the `NTP_timestamp_sample` and the `STC_sample` at the same temporal points. If more than one MPEG-2 ES with different MPEG-2 STCs are used, more than one CRI messages will be used.

10.3.5.2 Syntax

The syntax of the CRI message is defined in Table 24.

Table 24 — CRI message syntax

Syntax	Value	No. of bits	Mnemonic
CRI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
CRI_table()			
}			
}			

10.3.5.3 Semantics

message_id – indicates the identifier of the CRI message.

version – indicates the version of the CRI message. An MMT receiving entity can check the version of a received message using this field.

length – indicates the length of the CRI message, counted in bytes starting from the beginning of the next field to the last byte of the CRI message. The value "0" is not valid for this field.

CRI_table()- a CRI table is defined in 10.3.10.

10.3.6 DCI message

10.3.6.1 General

The DCI message delivers a DCI table that provides required device capabilities for the Package consumption.

10.3.6.2 Syntax

The syntax of the DCI message is defined in Table 25.

Table 25 — DCI message syntax

Syntax	Value	No. of bits	Mnemonic
DCI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
DCI_table()			
}			
}			

10.3.6.3 Semantics

message_id – indicates the identifier of the DCI message.

version – indicates the version of the DCI message. An MMT receiving entity can check the version of a received message using this field.

`length` – indicates the length of the DCI message, counted in bytes starting from the beginning of the next field to the last byte of the DCI message. The value "0" is not valid for this field.

`DCI_table()` – provides the required device capabilities for the Package consumption. The content of a DCI table is defined in 10.3.11.

10.3.7 PA table

10.3.7.1 General

A PA table provides information on all other signalling tables for the consumption of a Package.

10.3.7.2 Syntax

The syntax of the PA table is defined in Table 26.

Table 26 — PA table syntax

Syntax	Value	No. of bits	Mnemonic
PA_table () {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
information_table_info {			
number_of_tables	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
signalling_information_table_id		**8**	**uimsbf**
signalling_information_table_version		**8**	**uimsbf**
location {			
MMT_general_location_info()			
}			
reserved	"1111 111"	**7**	**bslbf**
alternative_location_flag		**1**	**bslbf**
if (alternative_location_flag == 1) {			
alternative_location {			
MMT_general_location_info()			
}			
}			
}			
}			
reserved	"1111 111"	**7**	**bslbf**
private_extension_flag		**1**	**bslbf**
if (private_extension_flag == 1)			
private_extension {			
}			
}			
}			

10.3.7.3 Semantics

`table_id` – indicates the identifier of the PA table.

`version` – indicates the version of the PA table. The newer version obsoletes the information in any older version.

`length` – indicates the length of the PA table in bytes, counting from the beginning of the next field to the last byte of the PA table. The value "0" is not valid for this field.

`number_of_tables` – indicates the number of signalling tables whose information is provided in this PA table.

`signalling_information_table_id` – indicates the ID of a signalling table whose information is provided in this PA table.

`signalling_information_table_version` – indicates the version of a signalling table whose information is provided in this PA table.

`MMT_general_location_info` – provides the location of a signalling table whose information is provided in this PA table. Syntax and semantics of `MMT_general_location_info` are defined in 10.6.1.

`alternative_location_flag` – if this flag is set to "1", an alternative address from where an MMT receiving entity can get the information table is provided.

`MMT_general_location_info_alternative_location` – provides the information of an alternative address from where an MMT receiving entity can get the signalling table. Only `location_type` from "`0x07`" to "`0x0B`" shall be used in `MMT_general_location_info` for a second location.

`private_extension_flag` – if this flag is "1", the private extension is present.

`private_extension` – a syntax element group serving as a container for proprietary or application-specific extensions.

10.3.8 MPI table

10.3.8.1 General

An MPI table carries a complete or a subset of a presentation information document. In case of a subset of a presentation information document, MPI tables for each subset are delivered in separate messages.

10.3.8.2 Syntax

The syntax of the MPI table is defined in Table 27.

Table 27 — MPI table syntax

Syntax	Value	No. of bits	Mnemonic
MPI_table () {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length	**N1**	**16**	**uimsbf**
reserved	**"1111"**	**4**	**bslbf**
PI_mode		**2**	**uimsbf**
reserved	**"11"**	**2**	**bslbf**
MPIT_descriptors {			
MPIT_descriptors_length	**N2**	**16**	**uimsbf**

Table 27 *(continued)*

Syntax	Value	No. of bits	Mnemonic
for (i=0; i<N2; i++) {			
MPIT_descriptors_byte		8	uimsbf
}			
}			
PI_content_count	N3	8	uimsbf
for (i =0; i<N3; i++) {			
PI_content_type {			
PI_content_type_length	N4	8	uimsbf
for (j=0; j<N4; j++) {			
PI_content_type_length_byte		8	uimsbf
}			
}			
PI_content_name_length	N5	8	uimsbf
for (j=0; j<N5; j++) {			
PI_content_name_byte		8	uimsbf
}			
PI_content_descriptors {			
PI_content_descriptors_length	N6	16	uimsbf
for (i=0; i<N6; i++) {			
PI_contnent_descriptors_byte		8	uimsbf
}			
}			
PI_content_length	N7	16	uimsbf
for (j=0; j<N7; j++) {			
PI_content_byte		8	uimsbf
}			
}			
}			

10.3.8.3 Semantics

`table_id` – indicates the identifier of the MPI table. A complete presentation information document and each subset of a presentation information document shall have distinct table identifiers. The processing order of subsets of a presentation information (PI) document is specified by this information. Since the `table_id` values are assigned contiguously, the PI subset number can be deduced from this field, i.e. the PI subset number equals this field minus the `table_id` of the base MPI table. The number 0 indicates base PI and the numbers "1"~"14" indicate the subset of PI. The number "15" has a special meaning since it indicates a complete PI.

`version` – indicates the version of the MPI table. The newer version overrides the older one as soon as it is received if `table_id` indicates a complete MPI, if subset 0 MPI has the same version value as this field (when `PI_mode` is "1"), or if all MPIs with the lower-subset number have the same version value as this field (when `PI_mode` is "0"), or if processing of the MPIs are independent (when `PI_mode` is "2"). If subset 0 MPI table has a newer version, all PIs with higher subset numbers up to 14 previously stored within an MMT receiving entity are treated as outdated except for the independent mode. When the PI subset number is not 0 and `PI_mode` is "1", the contents of the MPI table with a version different from that of subset 0 PI stored in an MMT receiving entity shall be ignored. Also, when the PI subset number is not 0 and `PI_mode` is "0", the contents of the MPI table with a version different from that of lower-subset

PIs stored in an MMT receiving entity shall be ignored. It shall be modulo-256 incremented per version change.

`length` – indicates the length of the MPI table in bytes, counting from the beginning of the next field to the last byte of the MPI table. The value "0" is not valid for this field.

`PI_mode` – indicates the mode of a PI subset processing. In "`sequential_order_processing_mode`" and with a non-zero subset number of this PI, an MMT receiving entity shall receive all PIs with lower subset numbers that have the same version as this PI before it processes this PI. For example, an MMT receiving entity cannot process subset-3 PI, if it has not received subset-2 PI with the same version. In "`order_irrelevant_processing_mode`" and with the layer number of this MMT-PI set to non-zero, an MMT receiving entity should process a PI right after it receives the PI as long as the subset 0 PI stored in an MMT receiving entity has the same version as this PI. In "`independent_processing_mode`", versions of each subset of PIs are managed individually. Fragmented PI is adapted in this mode. The value of `PI_mode` is specified in Table 28.

Table 28 — Value of PI_mode

Value	Description
00	"sequential_order_processing_mode"
01	"order_irrelevant_processing_mode"
10	"independent_processing_mode"
11	Reserved

`MPIT_descriptors_length` – indicates the length of the MPIT descriptors syntax loop. The length is counted from the next field to the end of the MPIT descriptors syntax loop. Several descriptors that include information on the whole MPIT can be inserted in this syntax loop.

`MPIT_descriptors_byte` – a byte in the MPIT descriptors syntax loop.

`PI_content_count` – indicates the number of PI contents delivered in this MPI table.

`PI_content_type_length` – indicates the length of the content type of this PI content excluding the terminating null character. The content type must be one of the MIME media types registered at the IANA website, i.e. http://www.iana.org/assignments/media-types.

`PI_content_type_byte` – a byte in the content type string of this PI content excluding the terminating null character.

`PI_content_name_length` – indicates the length of the name string of this PI content excluding the terminating null character.

`PI_content_name_byte` – a byte in the name string of this PI content excluding the terminating null character.

`PI_content_descriptors_length` – indicates the length of the PI content descriptors syntax loop. The length is counted from the next field to the end of the PI content descriptors syntax loop. Several descriptors that include information on the PI content can be inserted in this syntax loop.

`PI_content_descriptors_byte` – a byte in the PI content descriptors syntax loop.

`PI_content_length` – indicates the length in bytes of this PI content.

`PI_content_byte` – a byte in this PI content.

10.3.9 MP table

10.3.9.1 General

A complete MP table has the information related to a Package including the list of all Assets. A subset MP table has a portion of information from a complete MP table. In addition, MP table subset 0 has the minimum information required for Package consumption.

10.3.9.2 Syntax

The syntax of the MP table is defined in Table 29

Table 29 — MP table syntax

Syntax	Value	No. of bits	Mnemonic
MP_table() {			
table_id		8	uimsbf
version		8	uimsbf
length		16	uimsbf
reserved	"11 1111"	6	bslbf
MP_table_mode		2	bslbf
if ((table_id == 0x20)or(table_id == 0x11)){			
MMT_package_id {			
MMT_package_id_length	N1	8	uimsbf
for (i=0; i<N1; i++) {			
MMT_package_id_byte		8	uimsbf
}			
}			
MP_table_descriptors {			
MP_table_descriptors_length	N2	16	uimsbf
for (i=0; i<N2; i++) {			
MP_table_descriptors_byte		8	uimsbf
}			
}			
}			
number_of_assets	N3	8	uimsbf
for (i=0; i<N3; i++) {			
Identifier_mapping()			
asset_type		32	char
reserved	"1111 11"	5	bslbf
asset_modification_flag		1	bslbf
default_asset_flag		1	bslbf
asset_clock_relation_flag		1	bslbf
if (asset_clock_relation_flag == 1) {			
asset_clock_relation_id		8	uimsbf
reserved	"1111 111"	7	bslbf
asset_timescale_flag		1	bslbf
if (asset_time_scale_flag == 1) {			
asset_timescale		32	uimsbf

Table 29 *(continued)*

Syntax	Value	No. of bits	Mnemonic
}			
}			
asset_location {			
location_count	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
MMT_general_location_info()			
}			
}			
asset_descriptors {			
asset_descriptors_length	**N5**	**16**	**uimsbf**
for (j=0; j<N5; j++) {			
asset_descriptors_byte		**8**	**uimsbf**
}			
}			
}			
}			

10.3.9.3 Semantics

`table_id` – indicates the identifier of the MP table. A complete MP table and each subset MP tables shall use different table identifiers. The subset number of the MP table is implicitly represented by this field. Since the `table_id` values are assigned contiguously, the MP table subset number can be deduced from this field, i.e. the MP table subset number equals this field minus the `table_id` of the base MP table. The MP table subset number provides the subset number of this MP table. The number "0" indicates the base MP table and the numbers "1"~"14" indicate a subset of MP table. The number "15" has a special meaning since it indicates a complete MP table.

`version` – indicates the version of the MP table. The newer version overrides the older one as soon as it has been received. If the `MP_table_mode` is not set to independent mode and a subset 0 MP table with a newer version number is received, all MP table subsets with a higher subset number (excluding complete MP tables) that were previously stored by the MMT receiving entity shall be treated as outdated. When the MP table subset number is not "0" and `MP_table_mode` is "01", the contents of the MP table subset with a version different from that of the subset 0 MP table stored by an MMT receiving entity shall be ignored. Also, when the MP table subset number is not "0" and `MP_table_mode` is "0", the contents of the MP table subset with a version different from that of lower-subset MP table subsets stored by an MMT receiving entity shall be ignored. The version number wraps around after reaching "255" and shall be incremented by one for every version change.

`length` – contains the length of the MP table in bytes, counting from the beginning of the next field to the last byte of the MP table. The value "0" is not valid for this field.

`MP_table_mode` – indicates the mode of an MP table subset processing when the MP table subset mechanism is used. In "`sequential_order_processing_mode`" and with a non-zero subset number of this MP table, an MMT receiving entity shall receive all MP table subsets with lower subset numbers that have the same version as this MP table subset before it processes this MP table subset. For example, an MMT receiving entity cannot process a subset-3 MP table if it has not received a subset-2 MP table with the same version. In "`order_irrelevant_processing_mode`" and with the subset number of this MP table subset set to non-zero, an MMT receiving entity should process an MP table subset right after it receives the MP table subset as long as the subset 0 MP table stored in an MMT receiving entity has the same version as this MP table subset. In "`independent_processing_mode`", the version of each MP table subset is managed individually. The fragmented MP table, where each MP table subset is delivered by one of multiple MMT sending entities, is adapted in this mode. The independent mode of subsets of the

MP table can be used for the multi-channel instantiation, i.e. MP table subsets from subset 0 MP table to subset-N MP table are assigned as logical channels from Ch "0" to Ch "N". When an MPI message carries both an MPI table subset and the associated MP table subset, the `PI_mode` in the MPI table and the `MP_table_mode` in the MP table shall have the same value. The value of `MP_table_mode` is specified in Table 30.

Table 30 — Value of MP_table_mode

Value	Description
00	"sequential_order_processing_mode"
01	"order_irrelevant_processing_mode"
10	"independent_processing_mode"
11	Reserved

`MMT_package_id` – this field is a unique identifier of the Package.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id string`, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`asset_type` – provides the type of Asset. This is described in a four character code ("4CC") type registered in MP4REG (http://www.mp4ra.org).

`MP_table_descriptors` – this field provides descriptors for the MP table.

`MP_table_descriptors_length` – contains the length of the descriptor syntax loop. The length is counted from the next field to the end of the descriptor syntax loop. Several descriptors can be inserted in this syntax loop.

`MP_table_descriptors_byte` – one byte in the descriptors loop.

`number_of_assets` – provides the number of Assets whose information is provided by this MP table.

`Identifier_mapping` – provides information of `identifier_mapping` as defined in 10.6.3.

`asset_clock_relation_flag` – indicates whether an Asset uses an NTP clock or other clock systems as the clock reference. If this flag is "1", `asset_clock_relation_id` field is included. If this field is "0", the NTP clock is used for the Asset.

`asset_clock_relation_id` – provides a clock relation identifier for the Asset. This field is used to reference the clock relation delivered by a `CRI_descriptor()` for the Asset. The value of this field is one of the `clock_relation_id` values provided by the CRI descriptors (see 10.5.1).

`asset_timescale_flag` – indicates whether "`asset_timescale`" information is provided or not. If this flag is "1", `asset_timescale` field is included and if this flag is set to "0", `asset_timescale` is 90,000 (90 kHz).

`location_count` – provides the number of location information for an Asset. Set to "1" when an Asset is delivered through one location. When bulk delivery is achieved, in which MPUs contained in an Asset are delivered through multi-channels, not "1" is set. When one Asset is delivered over multiple locations, an MMT receiving entity shall receive all MPUs of the Asset from all indicated locations.

`asset_timescale` – provides information of time unit for all timestamps used for the Asset; expressed in the number of units in one second.

`MMT_general_location_info_for_asset_location` – provides the location information of the Asset. General location reference information for the Asset defined in 10.6.1 is used. Only the value of `location_type` between "`0x00`" and "`0x06`" shall be used for an Asset location.

`asset_descriptors_length` – the number of bytes counted from the beginning of the next field to the end of the Asset descriptors syntax loop.

`asset_descriptors_byte` – specifies a byte in Asset descriptors.

`default_asset_flag` – indicates whether an Asset is marked as a default asset or not. In case an asset is marked as a default asset, the MPU timestamp descriptor should be present for the corresponding timed asset. If this flag is "0", the asset is marked as a default asset.

`asset_modification_flag` – indicates whether the corresponding Asset can be replaced or modified or changed by MMT sending entity or MANE.

10.3.10 CRI table

10.3.10.1 General

The CRI table defined in Table 31 is delivered by the CRI message. Also, it may be delivered by a PA message.

10.3.10.2 Syntax

The syntax of the CRI table is defined in Table 31. A CRI table may include multiple CRI descriptors (see 10.5.1).

Table 31 — CRI table syntax

Syntax	Value	No. of bits	Mnemonic
`CRI_table () {`			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
number_of_CRI_descriptor	**N1**	**8**	**uimsbf**
`for (i=0; i<N1; i++) {`			
CRI_descriptor()		**152**	**uimsbf**
`}`			
`}`			

10.3.10.3 Semantics

`table_id` – indicates the table identifier of the CRI table.

`version` – indicates the version of the CRI table. The newer version overrides the older one as soon as it is received.

`length` – indicates the length of the CRI table counted in bytes starting from the beginning of the next field to the last byte of the CRI table. The value "0" is not valid for this field. This value shall be a multiple of 19.

`number_of_CRI_descriptor` - indicates the number of `CRI_descriptor`.

`CRI_descriptor()` – a CRI descriptor as defined in 10.5.1.

10.3.11 DCI table

10.3.11.1 General

The DCI table contains information on required device capabilities for the consumption of the Package. Depending on the MIME type of the Asset, a different set of information may be provided to support the delivery and consumption of the Package. This document differentiates between video, audio and download (applicable to non-timed data) MIME types.

10.3.11.2 Syntax

The syntax of DCI table is defined in Table 32.

Table 32 — DCI table syntax

Syntax	Value	No. of bits	Mnemonic
DCI_table() {			
table_id		8	uimsbf
version		8	uimsbf
length		16	uimsbf
number_of_assets	N1	8	uimsbf
for (i=0; i<N1; i++) {			
asset_id()			
mime_type()			
reserved	"111 1111"	7	bslbf
codec_complexity_flag		1	bslbf
if (codec_complexity_flag == 1) {			
if (top_level_mime_type == video) {			
video_codec_complexity {			
video_average_bitrate		16	uimsbf
video_maximum_bitrate		16	uimsbf
horizontal_resolution		16	uimsbf
vertical_resolution		16	uimsbf
temporal_resolution		8	uimsbf
video_minimum_buffer_size		16	uimsbf
}			
}else if (top_level_mime_type == audio) {			
audio_codec_complexity {			
audio_average_bitrate		16	uimsbf
audio_maximum_bitrate		16	uimsbf
audio_minimum_buffer_size		16	uimsbf
}			
}			
}			
else {			
download_capability {			
required_storage		32	uimsbf
}			
}			

Table 32 *(continued)*

Syntax	Value	No. of bits	Mnemonic
} }			

10.3.11.3 Semantics

`table_id` – indicates the identifier of the DCI table.

`version` – indicates the version of the DCI table. The newer version overrides the older one as soon as the DCI table is received.

`length` – indicates the length of the DCI table in bytes, counting from the beginning of the next field to the last byte of the DCI table. The value "0" is not valid for this field.

`number_of_assets` – indicates the number of Assets.

`asset_id` – provides the identifier of the Asset as defined in 10.6.2.

`top_level_mime_type` – indicates the media type part of the MIME type as given in the `mime_type()` syntax element. MIME types are defined in RFC 2046.

`codec_complexity_flag` – if this flag is "1", codec complexity information is provided.

`video_codec_complexity` – indicates the complexity the video decoder has to deal with.

`video_average_bitrate` – indicates the average bitrate of the video in kilo-bit/s for the whole Asset.

`video_maximum_bitrate` – indicates the maximum bitrate of the video in kilo-bit/s for the whole Asset.

`horizontal_resolution` – indicates the horizontal resolution of the video in pixels.

`vertical_resolution` – indicates the vertical resolution of the video in pixels.

`temporal_resolution` – indicates the average temporal resolution of the video in frames per second.

`video_mimimum_buffer_size` – indicates the minimum size of video decoder buffer that needs to be available in kilo-bytes.

`audio_codec_complexity` – indicates the complexity the audio decoder has to deal with.

`audio_average_bitrate` – indicates the average bitrate in kilo-bit/s for the whole Asset.

`audio_maximum_bitrate` – indicates the maximum bitrate in kilo-bit/s for the whole Asset.

`audio_minimum_buffer_size` – indicates the minimum size of audio decoder buffer needs to be processed in kilo-bytes.

`download_capability` – indicates the required capability for download.

`required_storage` – indicates the size of storage in kilobytes required to download.

10.3.12 Layout Configuration Table

10.3.12.1 General

This table assigns layout number, device identification, and region number for practical layout information to present one or more assets. This table has a descriptor field. The descriptor in this field may include information on the layout such as layout name, usage of the layout, etc. This table is carried in the PA message, and the information on the layout can be stored in a receiver.

10.3.12.2 Syntax

Table 33 shows the syntax of the Layout Configuration Table.

Table 33 — Syntax of Layout Configuration Table

Syntax	Values	No. of bits	Mnemonic
Layout_Configuration_Table() {			
table_id		8	uimsbf
version		8	uimsbf
length		16	uimsbf
number_of_loop		8	uimsbf
for (i=0; i<N; i++) {			
layout_number		8	uimsbf
device_id		8	uimsbf
number_of_region		8	uimsbf
for (j=0; j<M; j++) {			
region_number		8	uimsbf
left_top_pos_x		8	uimsbf
left_top_pos_y		8	uimsbf
right_down_pos_x		8	uimsbf
right_down_pos_y		8	uimsbf
layer_order		8	uimsbf
}			
}			
descriptor()			
}			

10.3.12.3 semantic

number_of_loop – This field specifies the number of combination of layout and device, which are configured in this Table.

layout_number – This field specifies layout number. The layout number '0' means the default layout. One layout includes multiple devices.

device_id – This field specifies device number. The device id '0' means the main device.

number_of_region – This field specifies the number of regions in the specified device.

region_number – This field specifies region number. The region number '0' means the default region. Different devices in the same layout have different region number.

left_top_pos_x – This field specifies the left-top position of the region as the percentage of left side position pixels to the total horizontal pixels.

left_top_pos_y – This field specifies the left-top position of the region as the percentage of top side position pixels to the total vertical pixels.

right_down_pos_x – This field specifies the right-down position of the region as the percentage of right side position pixels to the total horizontal pixels.

right_down_pos_y – This field specifies the right-down position of the region as the percentage of down side position pixels to the total vertical pixels.

`layer_order` – This field specifies the relative position of the region in depth direction. The layer order '0' means the most back side position. The greater value of this field means the more front side position.

`descriptor` – This descriptor field provides more detailed information on the layout.

10.3.13 SSWR message

10.3.13.1 General

The overall operation of downloadable DRM and CAS for MMT is described in Annex E. There are five steps in Annex E. Among them, the message for the security software request is sent from a receiving MMT entity to a downloadable DRM/CAS server. The message for DRM and CAS SW request is defined in this subclause.

10.3.13.2 Syntax

The syntax of SSWR message is defined in Table 34.

Table 34 — SSWR message syntax

Syntax	Value	No. of bits	Mnemonic
SSWR_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
token_ID {			
token_ID_URI_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
token_ID_URI_byte		**8**	**uimsbf**
}			
Number_of_deviceID	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
device_ID {			
deviceID_length	**N3**	**8**	**uimsbf**
for (j=0; j<N3; j++) {			
deviceID_byte		**8**	**uimsbf**
}			
}			
}			
tokenIssure_ID {			
tokenIssuer_ID_URI_length	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
tokenIssuer_URI_bytes		**8**	**uimsbf**
}			
}			
tokenIssueTime		**64**	**uimsbf**
tokenExpireTime		**64**	**uimsbf**
information_table_info {			
number_of_tables	**N5**	**8**	**uimsbf**

Table 34 *(continued)*

Syntax	Value	No. of bits	Mnemonic
for (i=0; i<N5; i++) {			
MMT_signaling_table_id		**8**	**uimsbf**
MMT_signaling_table_version		**8**	**umisbf**
}			
}			
}			
}			
}			

10.3.13.3 Semantics

`message_id` – indicates the type of MMT signalling messages.

`version` – indicates the version of MMT signalling messages.

`length` – indicates the length of MMT signalling messages. The value "0" is never used for this field.

`tokenID` – identification of a Token and is provided by the Token Provider. The Token should be provided by a trustable entity. It has sub-elements of Device ID, Token Issuer ID, Issue Time and Expire Time.

`deviceID` – provides the identification of device(s) under the Token. If the MMT client wants to consume Asset/Package of two different devices, then multiple Device IDs should be provided.

`tokenIssureID` – identification of trust entity that issues a Token. This field is to be used by the D-DRM/D-CAS server to verify the validity of the Token.

`tokenIssueTime` – a time at which the Token is issued. The unit of this field is second. The NTC format will be used.

`tokenExpireTime` – a time at which the Token is expired. The unit of this field is second. The NTC format will be used.

`MMT_signaling_table_info` – provides the information of MPT related to Package/Asset to be descripted by downloaded DRM or CAS.

10.3.14 LS message

10.3.14.1 General

The LS message carries the license information targeted to a specific receiver or group of receivers. This information is delivered to receivers that do not have a return channel.

10.3.14.2 Syntax

The syntax of LS message is defined as follows in Table 35:

Table 35 — LS message syntax

Syntax	Value	No. of bits	Mnemonic
LS_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**

Table 35 *(continued)*

Syntax	Value	No. of bits	Mnemonic
length		**32**	**uimsbf**
message_payload {			
system_id		**16*8**	**uimsbf**
number_of_licenses	**N1**	**32**	**uimsbf**
for(i=0;i<N1;i++) {			
license_message_hash_length	**N2**	**8**	**uimsbf**
license_message_hash		**8*N2**	**uimsbf**
encrypted_license_data_length	**N3**	**16**	**uimsbf**
encrypted_license_data		**8*N3**	**uimsbf**
}			
}			
}			

10.3.14.3 Semantics

`message_id` – indicates the identifier of the PA messages.

`version` – indicates the version of the PA messages.

`length` – a 32-bit field for conveying the length of the LS message in bytes, counting from the beginning of the next field to the last byte of the LS message. The value "0" is not valid for this field.

`system_id` – provides the UUID that uniquely identifies the DRM system.

`number_of_licenses` – provides the number of licenses in the LS message.

`license_message_hash_length` – provides the length in bytes of the license message hash.

`license_message_hash` – the license message hash code is used to identify the target receiver or group of receivers for the enclosed license. The license message hash is generated by the content decryption module in the same way as the license server. The hash generation algorithm is DRM system specific.

`encrypted_license_data_length` – provides the length of the encrypted license data.

`encrypted_license_data` – contains an encrypted license that corresponds to the license message of which the hash value is included in this message. The license is encrypted using the certificate of the targeted receiver or group of receivers.

10.3.15 LR message

10.3.15.1 General

The license revocation message is defined to allow the DRM system to provide a signal that it has revoked the license for a user or group of users.

10.3.15.2 Syntax

The syntax of the LR signalling message is defined in Table 36.

Table 36 — LR message syntax

Syntax	Value	No. of bits	Mnemonic
LS_message () {			

Table 36 *(continued)*

Syntax	Value	No. of bits	Mnemonic
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**32**	**uimsbf**
message_payload {			
system_id		**16*8**	**uimsbf**
number_of_licenses	**N1**	**32**	**uimsbf**
for(i=0;i<N1;i++) {			
encrypted_license_challenge_length	**N2**	**16**	**uimsbf**
encrypted_license_challenge		**8*N2**	**uimsbf**
}			
}			
}			

10.3.15.3 Semantics

`message_id` – indicates the identifier of the LR messages.

`version` – indicates the version of the LR messages.

`length` – a 32-bit field for conveying the length of the LR message in bytes, counting from the beginning of the next field to the last byte of the LR message. The value "0" is not valid for this field.

`System_id` – provides the UUID that uniquely identifies the DRM system.

`number_of_licenses` – provides the number of licenses in the LS message.

`encrypted_license_challenge_length` – provides the length of the encrypted license challenge.

`encrypted_license_challenge` – contains an encrypted license challenge. The license challenge is encrypted using the certificate of the targeted receiver or group of receivers.

10.3.16 SI table

10.3.16.1 General

The security information table (SIT) provides the required security for the consumption of the Package.

10.3.16.2 Syntax

The syntax of the SIT is defined in Table 37.

Table 37 — SIT syntax

Syntax	Value	No. of bits	Mnemonic
SI_table() {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
number_of_Security_System	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
system_id {			

Table 37 *(continued)*

Syntax	Value	No. of bits	Mnemonic
system_id_length	**N2**	**8**	**uimsbf**
for (j=0; j<N2; j++) {			
system_id_byte		**8**	**uimsbf**
}			
}			
systemProvider {			
systemProvider_URL_length	**N3**	**8**	**uimsbf**
for (i=0; i<N3; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
reserved	**"1111 111"**	**7**	**bslbf**
encryption_flag		**1**	**bslbf**
if (encryption_flag == 1) {			
encAlgorithm {			
encAlgorithm_length	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
encAlgorithm _byte		**8**	**uimsbf**
}			
}			
keySize {			
keySize_length	**N5**	**8**	**uimsbf**
for (i=0; i<N5; i++) {			
keySize_byte		**8**	**uimsbf**
}			
}			
keyUrl {			
key_URL_length	**N6**	**8**	**uimsbf**
for (i=0; i<N6; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
initializationVector {			
initializationVector_length	**N7**	**8**	**uimsbf**
for (i=0; i<N7; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
ivUrl {			
iv_URL_length	**N8**	**8**	**uimsbf**
for (i=0; i<N8; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			

Table 37 *(continued)*

Syntax	Value	No. of bits	Mnemonic
keyUrlTemplate {			
keyUrlTemplate_length	**N9**	**8**	**uimsbf**
for (i=0; i<N9; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
ivUrlTemplate {			
ivUrlTemplate_length	**N10**	**8**	**uimsbf**
for (i=0; i<N10; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
reserved	**"1111 111"**	**7**	**bslbf**
authentication_flag		**1**	**bslbf**
if (authentication_flag == 1) {			
digestUrl {			
digest_URL_length	**N11**	**8**	**uimsbf**
for (i=0; i<N11; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
digestURLTemplate {			
digestUrlTemplate_length	**N12**	**8**	**uimsbf**
for (i=0; i<N12; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
signatureUrl {			
signature_URL_length	**N13**	**8**	**uimsbf**
for (i=0; i<N13; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
signatureURLTemplate {			
signatureUrlTemplate_length	**N14**	**8**	**uimsbf**
for (i=0; i<N14; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
signatureLength {			
signature_length	**N15**	**8**	**uimsbf**
for (i=0; i<N15; i++) {			
Length_byte		**8**	**uimsbf**
}			

Table 37 *(continued)*

Syntax	Value	No. of bits	Mnemonic
}			
signatureKeyUrl {			
signatureKey_URL_length	**N16**	**8**	**uimsbf**
for (i=0; i<N16; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
}			
number_of_License	**N17**		
for (i=0; i<N17; i++) {			
license_Url {			
licenseUrl_length	**N18**	**8**	**uimsbf**
for (j=0; j<N18; j++) {			
URL_byte		**8**	**uimsbf**
}			
}			
licenseURLTemplate {			
lisenceUrlTemplate_length	**N19**	**8**	**uimsbf**
for (i=0; i<N19; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
}			
}			
}			

NOTE How a specific DRM solution or CAS solution works with MMT is out of the scope.

10.3.16.3 Semantics

`table_id` – indicates the ID of SIT.

`version` – indicates a version of SIT. The newer version overrides the older one as soon as it has been received.

`length` – provides the length of SIT counted in bytes starting from the next field to the last byte of the SIT. The value "0" is never used for this field.

`number_of_Security_System` – provides a DRM or CAS system that can process and handle the content protection, access control and rights management.

`systemId` – provides a UUID-formatted opaque string for the system. This can also be used to encode the type of the system.

`systemProvider` – provides a URL location of a provider for the system. This can be used for downloading and installing the system, when needed.

`Encryption` – this optional element provides the information about encryption.

`encAlgorithm` – provides the algorithm used for encryption.

`keySize` – provides the size of initialization vectors in bytes.

`keyUrl` – provides the URL for key derivation.

`IV` – provides the initialization vector.

`ivUrl` – provides the URL for initialization vector derivation.

`keyUrlTemplate` – provides the template for key URL generation.

`ivUrlTemplate` – provides the template for IV URL generation.

`Authentication` – this optional element gives the information required for authentication.

`digestUrl` - provides the URL used for retrieving the digest value.

`digestUrlTemplate` – provides the template for creating the URL used for retrieving the digest value.

`signatureUrl` – provides the URL used for retrieving the signature value.

`signatureUrlTemplate` – provides the template for creating the URL used for retrieving the signature value.

`signatureLength` – provides the length of the signature. It shall appear only if the length is shorter than the normal output size of the signature algorithm.

`signaturekeyUrl` – provides the URL for the key used for the signature.

`License` – this optional element provides the information to get the License.

`licenseUrl` – the license format can be in some standard ones or be dependent on the system that is specified with `systemId` of the System element.

`licenseUrlTemplate` – specifies the template for license URL generation.

10.4 Signalling messages for Package delivery

This subclause specifies signalling messages that are used to exchange signalling information related to Package delivery except AL_FEC message which is defined in Clause C.6

10.4.1 Hypothetical receiver buffer model (HRBM) message

10.4.1.1 General

The HRBM is described in Clause 11. An HRBM message provides information on end-to-end transmission delay and memory requirement to an MMT receiving entity for efficient operation in a broadcasting environment. As the HRBM is applied per MMTP sub-flow, the MMT receiving entity can recognize the corresponding sub-flow for which it received HRBM information through the `packet_id` of the MMTP packet that carried the HRBM signalling message.

10.4.1.2 Syntax

The syntax of the HRBM message is defined in Table 38.

Table 38 — HRBM message syntax

Syntax	Values	No. of bits	Mnemonic
`HRBM_message ( ){`			
message_id		16	**uimsbf**
version		8	**uimsbf**

Table 38 *(continued)*

Syntax	Values	No. of bits	Mnemonic
length		16	**uimsbf**
message_payload{			
max_buffer_size		**32**	**uimsbf**
fixed_end_to_end_delay		**32**	**uimsbf**
max_transmission_delay		**32**	**uimsbf**
}			
}			

10.4.1.3 Semantics

message_id – indicates the identifier of the HRBM message.

version – version of the HRBM messages. An MMT receiving entity can use this field to check the version of the received HRBM message.

length – length of the HRBM messages in bytes, counting from the first byte of the next field to the last byte of the HRBM message. The value "0" is not valid for this field.

max_buffer_size – provides information for the required maximum buffer size in bytes of MMT Assets.

fixed_end_to_end_delay – provides information for fixed_end_to_end_delay between the sending entity and the receiving entity in millisecond.

max_transmission_delay – provides information for the max_transmission_delay between the sending entity and receiving entity in millisecond.

NOTE Fixed end-to-end delay is calculated by summing the max_transmission_delay and FEC protection_window_time (refer to AL-FEC message in C.6).

10.4.2 Measurement configuration (MC) message

10.4.2.1 General

MC messages are used for transport measurement and reporting configurations. The MC message provides multiple measurement configurations on multiple reporting messages at once. The measurement configuration defines reporting type, measurement times, and the number of reporting messages that follows the same measurement configuration. The reporting configurations are included in each measurement configuration, and it provides ID of desired reporting message, reporting period, and the MMT sending entity address where the reporting message shall be sent.

10.4.2.2 Syntax

The syntax of the MC message is defined in Table 39.

Table 39 — MC message syntax

Syntax	Values	No. of bits	Mnemonic
MC_message() {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
number_of_configurations	**N1**	**8**	**uimsbf**

Table 39 *(continued)*

Syntax	Values	No. of bits	Mnemonic
for (i=0; i<N1; i++) {			
reporting_type		**2**	**uimsbf**
reserved	**'11 1111'**	**6**	**uimsbf**
measurement_start_time		**32**	**uimsbf**
measurement_duration		**16**	**uimsbf**
number_of_reports	**N2**	**8**	**uimsbf**
for (j=0; j<N2; j++) {			
reporting_message_id		**16**	**uimsbf**
reporting_period		**16**	**uimsbf**
}			
}			
}			
}			

10.4.2.3 Semantics

`message_id` – indicates the message ID. The length of this field is 16 bits.

`version` – indicates the version of the messages. The MMT receiving entity may check whether the version of the received message is new or not. The newer version overrides the older one as soon as it is been received. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the MC message. The value "0" shall not be used for this field.

`number_of_configurations` - indicates the number of measurement configurations. The measurement configuration consists of `reporting_type`, `measurement_start_time`, `measurement_duration`, and a number of measurement result reports.

`reporting_type` – indicates how often the measurement results shall be reported to the reporting MMT sending entity. Valid values for this field are described in Table 40.

Table 40 — Value of reporting_type

Value	Description
00	Start measurement immediately and stop measurement at the appointed time
01	Start and stop measurement at the appointed time
10	Start measurement at the measurement condition
11	Stop measurement immediately

`measurement_start_time` - indicates the starting time of the measurement. This field shall be formatted as the "short-format" defined in IETF RFC 5905, NTP version 4, Clause 6.

`measurement_duration` - indicates the duration of the measurement in units of milliseconds. The measurement result reports are sent until the time of measurement_duration from the measurement_start_time.

`number_of_reports` - indicates the number of the measurement result reports. The measurement results are contained in the reporting messages.

`reporting_message_id` - indicates the message ID of each reporting message. The MMT sending entity requests the MMT receiving entity to send the MMT signalling message identified by reporting_message_id as a reporting message.

`reporting_period` - defines the period of the measurement result reports. The reporting_period indicates the time interval between the sending times of each measurement result. The unit of this field is milliseconds.

`reporting_MMT sending entity_address` - indicates the location of MMT sending entity that receives the results of requested measurement. Syntax and semantics of reporting_MMT sending entity_address use the syntax and semantics of MMT_general_location_info which is defined in sub-clause 10.6.1. Only the value of `location_type` "0x01", "0x02", "0x05", and "0x06" shall be used. For `location_type` "0x01" and "0x02", only the value of `ipv4_dst_addr`, `ipv6_dst_addr`, and `dst_port` shall be used. The value of `packet_id` shall be ignored.

10.4.3 ARQ configuration (AC) message

10.4.3.1 General

ARQ configuration information, which includes the policy to be adopted by the MMT sending entity and MMT receiving entity in the event of packet loss, shall be transmitted at the beginning of a session as the ARQ configuration (AC) message from the transmitting MMT sending entity to the MMT receiving entity either in-band or out-of-band.

10.4.3.2 Syntax

The syntax of the AC message is defined in Table 41.

Table 41 — AC message syntax

Syntax	Values	No. of bits	Mnemonic
AC_message() {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
flow_label_flag		**1**	**bslbf**
if(flow_label_flag == 1) {			
fb_flow_label		**7**	**uimsbf**
} else {			
reserved	**"1111 111"**	**7**	**bslbf**
}			
delay_constrained_ARQ_flag		**1**	**bslbf**
number_of_packet_id	**N1**	**7**	**uimsbf**
for(i=0; i<N1; i++) {			
packet_id		**16**	**uimsbf**
rtx_window_timeout		**32**	**uimsbf**
}			
arq_server_address{			
MMT_general_location_info()			
}			
}			
}			
}			

10.4.3.3 Semantics

`message_id` – indicates AC message ID. The length of this field is 16 bits.

`version` – indicates the version of AC messages. The MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of AC messages. The length of this field is 16 bits. It indicates the length of the AC message counted in bytes starting from the next field to the last byte of the AC message. The value "0" shall not be used.

`flow_label_flag` – indicates whether `fb_flow_label` exists. If the value is set to 1, the value of `rtx_flow_label` parameter is present.

`fb_flow_label` – indicates the flow label to be used when the MMT receiving entity sends an AF message. The `flow_label` will be allocated to guarantee higher priority for the AF message along the delivery path. This parameter will be presented only if the value of `flow_label_flag` parameter is set to 1.

`delay_constrained_ARQ_flag` – when set to "1", this flag indicates that the server supports delay constrained ARQ.

`number_of_packet_id` – indicates the number of packet id that has lost packets.

`rtx_window_timeout` – indicates the retransmit window timeout. An MMT sending entity will keep an MMTP packet in buffer until the timeout, and thus available for retransmission. The timeout for a certain MMTP packet starts when the MMT sending entity sends the MMTP packet. Thus, the MMT receiving entity can infer whether the MMTP packet is available by referencing the timestamp field in the MMTP packet header. The unit is milliseconds.

`arq_server_address` – indicates address of servers to which the MMT receiving entity can send the AF message to request lost MMTP packets.

NOTE The location_type of `MMT_general_location_info()` will be restricted to 0x01, 0x02 and 0x05 for simplicity.

10.4.4 ARQ feedback (AF) message

10.4.4.1 General

In the event of packet loss, this loss is detected by the MMT receiving entity. An ARQ feedback (AF) message is generated according to the information of the AC message and then transmitted to the MMT sending entity. When the MMT receiving entity detects that one or more packets have been lost, it forms a mask of up to 255 bytes where each bit in a byte corresponds to a sequence number of lost MMTP packets. This allows the AF message to report up to 255x8 lost packets in one AF message. The syntax for the AF message is shown in Table 42.

The method of packet loss detection is outside the scope of this document.

10.4.4.2 Syntax

The syntax of the AF message is defined in Table 42.

Table 42 — AF message syntax

Syntax	Values	No. of bits	Mnemonic
`AF_message() {`			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**

Table 42 *(continued)*

Syntax	Values	No. of bits	Mnemonic
length		**16**	**uimsbf**
message_payload {			
reserved	**'1111 1'**	**5**	**uimsbf**
argument_type		**1**	**bslbf**
delay_constrained_ARQ_mode		**2**	**uimsbf**
if(argument_type == 0) {			
if (delay_constrained_ARQ_mode == 01){			
ARQ_feedback_timestamp		**32**	**uimsbf**
}			
if (delay_constrained_ARQ_mode == 10){			
propagation_delay		**32**	**uimsbf**
}			
packet_counter		**32**	**uimsbf**
masklength	**N1**	**8**	**uimsbf**
if (delay_constrained_ARQ_mode ==01){			
arrival_deadline		**16**	**uimsbf**
}			
for(i=0; i<N1; i++){			
mask_byte		**8**	**uimsbf**
}			
}			
if(argument_type == 1) {			
reserved	**'1'**	**1**	**bslbf**
number_of_packet_id	**N2**	**7**	**uimsbf**
if (delay_constrained_ARQ_mode == 01){			
ARQ_feedback_timestamp		**32**	**uimsbf**
}			
if (delay_constrained_ARQ_mode == 10){			
propagation_delay		**32**	**uimsbf**
}			
for(i=0; i<N2; i++){			
packet_id		**16**	**uimsbf**
packet_sequence_number		**32**	**uimsbf**
masklength	**N3**	**8**	**uimsbf**
if (delay_constrained_ARQ_mode == 01){			
arrival_deadline		**16**	**uimsbf**
}			
for(j=0; j<N3; j++){			
mask_byte		**8**	**uimsbf**
}			
}			
}			
}			

10.4.4.3 Semantics

`message_id` – indicates AF message ID. The length of this field is 16 bits.

`version` – indicates the version of AF messages. The MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of AF messages. The length of this field is 16 bits. It indicates the length of the AF message counted in bytes starting from the next field to the last byte of the AF message. The value "0" shall not be used.

`argument_type` – indicates the type of argument the MMT receiving entity is using when requesting the lost packets to the server. Valid values for this field are described in Table 43.

Table 43 — Value of argument_type

Value	Description
0	Packet counter based ARQ. MMT receiving entity sends the AF messages with *packet_counter*.
1	Packet sequence number based ARQ. MMT receiving entity sends the AF messages with *packet_id* and *packet_sequence_number*.

`delay_constrained_ARQ_mode` – indicates the type of delay constrained ARQ. Valid values for this field are described in Table 44.

Table 44 — Value of delay_constrained_ARQ_mode

Value	Description
00	No time constrained ARQ. ARQ server does not need to consider any delay constraints when retransmitting the lost packets for this request.
01	Playout-time constrained ARQ. MMT receiving entity sends AF message, with *ARQ_feedback_timestamp* and *arrival_deadline*, to help the server decide whether to retransmit or not.
10	Delivery-time constrained ARQ. MMT receiving entity sends AF message, with *propagation_delay* only, to help the server decide whether to retransmit or not.
11	Reserved for future use.

`number_of_packet_id` – indicates the number of packet id that has lost packets.

`delay_constrained_ARQ_flag` – indicates the present of `arrival_deadline` field information.

`ARQ_feedback_timestamp` – indicates the NTP time at which the ARQ feedback is sent from the MMT receiving entity.

`propagation_delay` – propagation delay for the MMT packet to arrive at the MMT receiving entity. The MMT receiving entity calculates the `propagation_delay` by the subtracting the NTP time at the delivery instant of a MMT packet from the NTP time at the arrival instant of the MMT packet. The `propagation_delay` can be an average result of a propagation delay measured within the `measurement_duration`.

`packet_id` – this field is the integer value assigned to each Asset to distinguish packets of one Asset from another. Separate values will be assigned to signalling messages and FEC parity flows.

`packet_sequence_number` – corresponds to the `packet_sequence_number` of the first packet indicated by the `mask_byte` that is identified as having been detected to be lost and hence requiring re-transmission.

`masklength` – indicates the length of the data behind the mask in bytes.

`arrival_deadline` – indicates the deadline by which the retransmitted packet for the first lost packet should arrive at the MMT receiving entity for timely processing. This parameter represents the time

increment from the `ARQ_feedback_timestamp`. The first 8 bits represents integer part and the last 8 bits represents fractional part.

`mask_byte` – mask field, each bit corresponds to a MMTP packet. If the packet behind the packet with `packet_id` is lost, then the corresponding bit will be set to "1".

10.4.5 Reception quality feedback (RQF) message

10.4.5.1 General

An MMT receiving entity can send the reception quality feedback to the MMT sending entity to inform the reception quality of the received MMTP packet flow by using RQF messages. An MMT receiving entity needs to keep track of reception quality per MMT sending entity.

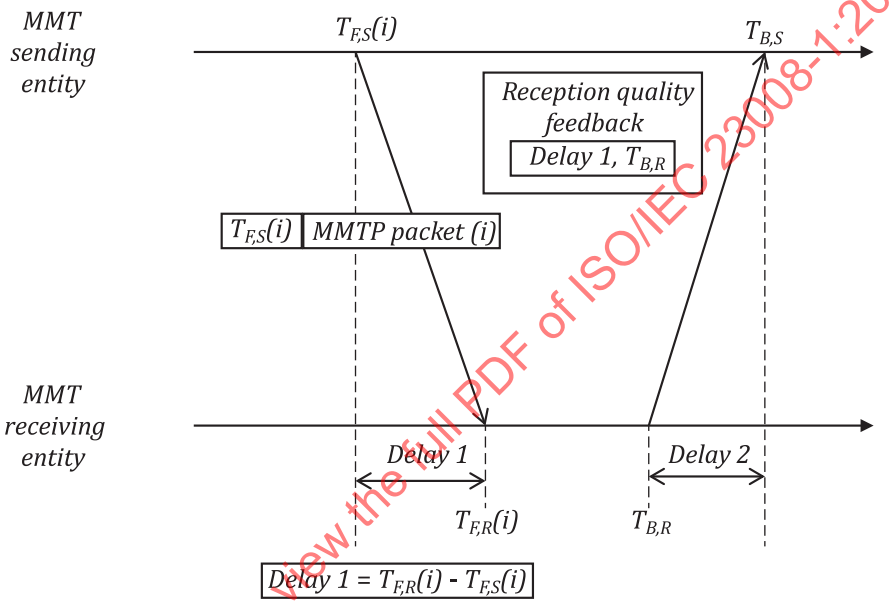


Figure 20 — Round trip time (RTT) calculation

An MMT receiving entity provides information in the feedback to allow the MMT sending entity to calculate the round trip time (RTT) and the process is as shown in Figure 20. In Figure 20, Delay 1 is the delivery time of the MMTP packet from the MMT sending entity to the MMT receiving entity. Delay 1 is calculated by the subtracting $T_{F,S}(i)$ (NTP time at delivery instant of i-th MMTP packet) from $T_{F,R}(i)$ (NTP time at arrival instant of i-th MMTP packet). The Delay 2 is the delivery time for the MMTP packet from the MMT receiving entity to the MMT sending entity. The Delay 2 is calculated by subtracting $T_{B,R}$ (NTP time at delivery instant of the feedback report, i.e. `feedback_timestamp`) from the $T_{B,S}$ (NTP time at arrival instant of the feedback report). Thus, the MMT sending entity can calculate the RTT by adding Delay 1 and Delay 2.

10.4.5.2 Syntax

The syntax of the RQF message is defined in Table 45.

Table 45 — RQF message syntax

Syntax	Values	No. of bits	Mnemonic
`RQF_message ( ) {`			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**

Table 45 *(continued)*

Syntax	Values	No. of bits	Mnemonic
message_payload {			
packet_loss_ratio		**8**	**uimsbf**
inter_arrival_jitter		**32**	**uimsbf**
max_transmission_delay		**32**	**uimsbf**
min_transmission_delay		**32**	**uimsbf**
RTT_parameter {			
propagation_delay		**32**	**uimsbf**
feedback_timestamp		**32**	**uimsbf**
}			
}			
}			

10.4.5.3 Semantics

`message_id` – identifier of the RQF message. The length of this field is 16 bits.

`version` – version of the RQF message. An MMT receiving entity can use this field to check the version of a received message. The length of this field is 8 bits.

`length` – length of the RQF in bytes, counting from the first byte of the next field to the last byte of the RQF message. The length of this field is 16 bits and the value "0" is not valid for this field.

`packet_loss_ratio` – ratio between the number of lost MMTP packets and the total number of transmitted packets. This value is equivalent to taking the integer part after multiplying the loss fraction by 256. The `packet_loss_ratio` is a result measured within the measurement duration which can be calculated between `measurement_start_time` and `measurement_stop_time` or can be provided by `average_bitrate_period`.

`inter_arrival_jitter` – deviation of the difference in packet spacing at the MMT receiving entity compared with the MMT sending entity for a pair of packets, measured in timestamp units. It can be estimated based on the time difference between the arrivals of adjacent MMTP packets. The `inter_arrival_jitter` is an average result measured within the measurement duration which can be calculated between `measurement_start_time` and `measurement_stop_time` or can be provided by `average_bitrate_period`.

RTT_parameter – parameter used for calculating the round trip time (RTT). RTT is the length of time required for the MMTP packet to be sent and the length of time it takes for an acknowledgement to be received. When computing the RTT, the MMT sending entity records the time when the feedback is received. RTT is calculated by subtracting the `feedback_timestamp` from the recorded time and adding the `propagation_delay`.

`propagation_delay` – propagation delay for the MMTP packet to arrive at the MMT receiving entity. The MMT receiving entity calculates the `propagation_delay` by the subtracting the NTP time at the delivery instant of an MMTP packet from the NTP time at the arrival instant of the MMTP packet. The `propagation_delay` can be an average result of a propagation delay measured within the `measurement_duration`.

`feedback_timestamp` – NTP time at which the feedback is sent from the MMT receiving entity. This parameter is used to measure the propagation delay from the MMT receiving entity to the MMT sending entity.

`max_transmission_delay` – the maximum transmission delay for the MMTP packet to arrive at the MMT receiving entity. The MMT receiving entity calculates the transmission delay by subtracting the NTP time at the delivery instant of a MMTP packet from the NTP time at the arrival instant of the MMTP

packet. The max_transmission_delay is the maximum transmission_delay measured within the measurement_duration.

min_transmission_delay – the minimum transmission delay for the MMTP packet to arrive at the MMT receiving entity. The MMT receiving entity calculates the transmission delay by subtracting the NTP time at the delivery instant of a MMTP packet from the NTP time at the arrival instant of the MMTP packet. The min_transmission_delay is the minimum transmission_delay measured within the measurement_duration.

10.4.6 NAM feedback (NAMF) message

10.4.6.1 Syntax

The syntax of the NAM feedback is defined in Table 46.

Table 46 — NAM_Feedback message syntax

Syntax	Value	No. of bits	Mnemonic
NAMF_message ( ) {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
CLI_id		**8**	**uimsbf**
NAM_flag		**1**	**boolean**
mobile_info_descriptor_flag		**1**	**boolean**
media_quality descriptor_flag		**1**	**boolean**
reserved	**'11111'**	**5**	**uimsbf**
if(NAM_flag == 0){			
relative_available_bitrate		**8**	**uimsbf**
relative_buffer_fullness		**8**	**uimsbf**
relative_peak_bitrate		**8**	**uimsbf**
average_bitrate_period		**16**	**uimsbf**
current_delay		**32**	**uimsbf**
generation_time		**32**	**uimsbf**
BER		**32**	**uimsbf**
packet_loss_ratio		**8**	**uimsbf**
jitter_duration		**32**	**uimsbf**
}else if(NAM_flag == 1){			
available_bitrate		**32**	**uimsbf**
buffer_fullness		**32**	**uimsbf**
peak_bitrate		**32**	**uimsbf**
average_bitrate_period		**16**	**uimsbf**
current_delay		**32**	**uimsbf**
generation_time		**32**	**uimsbf**
BER		**32**	**uimsbf**
packet_loss_ratio		**8**	**uimsbf**
jitter_duration		**32**	**uimsbf**
SDU_size		**32**	**uimsbf**

Table 46 *(continued)*

Syntax	Value	No. of bits	Mnemonic
SDU_loss_ratio		**8**	**uimsbf**
}			
if(mobile_info_descriptor_flag == 1)			
mobile_info_descriptor()			
if(media_quality descriptor_flag == 1)			
media_quality_measurement()			
}			
}			

10.4.6.2 Semantics

`message_id` – indicates the NAMF message ID. The length of this field is 16 bits.

`version` – indicates the version of NAMF messages. The MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of NAMF messages. The length of this field is 16 bits. It indicates the length of the NAM feedback message counted in bytes starting from the next field to the last byte of the NAM Feedback message. The value "0" shall not be used.

`NAM_flag` – indicates whether the NAMF message contains absolute NAM information or relative NAM information. The value "1" should be set, if the NAMF message contains absolute NAM information.

`CLI_id` – is an arbitrary integer number to identify this NAM among the underlying network.

`relative_available_bitrate` – the available bitrate change ratio (%) between the current NAM information and the previous NAM information.

`relative_buffer_fullness` – the remaining buffer fullness change ratio (%) between the current NAM information and the previous NAM information.

`relative_peak_bitrate` – the peak bitrate change ratio (%) between the current NAM information and the previous NAM information.

`available_bitrate` – is the bitrate that the scheduler of the underlying network can guarantee to the MMT stream. The `available_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

`buffer_fullness` – the buffer is used to absorb excess bitrate higher than the `available_bitrate`. The `buffer_fullness` is expressed in bytes.

`peak_bitrate` – the `peak_bitrate` is the maximum allowable bitrate that the underlying network can assign to the MMT stream. The `peak_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

`current_delay` – this parameter indicates the last hop transport delay. The `current_delay` is expressed in milliseconds.

`average_bitrate_period` – provides the period of time over which the average bitrate of the input of the MMT procotol session that carries the MMTP packet shall be calculated. The `average_bitrate_period` is provided in units of milliseconds.

`SDU_size` – Service data unit (SDU) is data unit in which the underlying network delivers the MMT data. The `SDU_size` specifies the length of the SDU and is expressed in bits. Overhead for the protocols of the underlying network is not included.

SDU_loss_ratio – the SDU_loss_ratio is a fraction of SDUs lost or detected as errorneous. Loss ratio of MMT packets can be calculated as a function of SDU_loss_ratio and SDU_size.
The SDU_loss_ratio is expressed in percentile.

generation_time – the time when the parameters are generated. The generation_time is expressed in milliseconds.

BER – Bit Error Rate obtained from the PHY or MAC layer. For BER from PHY layer, this value is presented as a positive value. For BER from the MAC layer, this value is presented as a negative value which can be used as an absolute value.

mobile_info_descriptor_flag – indicates whether mobile_info_descriptor is included or not. If it is set to '1', the descriptor is included.

media_quality_descriptor_flag – indicates whether media_quality_descriptor is included or not. If it is set to '1', the descriptor is included.

mobile_info_descriptor – indicates a descriptor defined in 10.5.7 which includes cellular device specific information. It is reported to a MMT sending entity when triggered by MC message in NAMF message while set mobile_descriptor_flag as '1'.

media_quality_descriptor – indicates a descriptor defined in 10.5.8 which includes media quality specific information. It is reported to a MMT sending entity when triggered by MC message in NAMF message while set media_quality_descriptor_flag as '1'.

jitter_duration – indicates total duration of playback bufferunderrun measured during the period specified in MC message in milliseconds. The length of this field is 32 bits.

10.4.7 Low delay consumption (LDC) message

10.4.7.1 General

The LDC Message provides information required to decode and present media data by the MMT receiving entity before it receives metadata such as movie fragment headers. This message indicates that the duration of each sample is fixed as signalled by default_sample_duration in Track Extends Box and the coding dependency structure is fixed across an Asset. When this message is used, the value of decoding time of the first sample of MPU is smaller than the presentation time of the first sample of the MPU by the sum of base_presentation_time_offset and the largest value of sample_composition_time_offset_value paired sample_composition_time_offset_sign is "1."

10.4.7.2 Syntax

The syntax of the low delay consumption message is defined in Table 47.

Table 47 — Low delay consumption message syntax

Syntax	Values	No. of bits	Mnemonic
LDC_message ( ){			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
base_presentation_time_offset		**31**	**uimsbf**
coding_dependency_structure_flag		**1**	**bslbf**
if (coding_dependency_structure_flag == 1){			
period_of_intra_coded_sample	**N1**	**8**	**uimsbf**

Table 47 *(continued)*

Syntax	Values	No. of bits	Mnemonic
for (i=0 ; i<N1;i++){			
sample_composition_time_offset_sign		**1**	**bslbf**
sample_composition_time_offset_value		**31**	**uimsbf**
}			
}			
}			
}			

10.4.7.3 Semantics

`message_id` – indicates the identifier of the LDC message.

`version` – version of the LDC messages. An MMT receiving entity can use this field to check the version of the received LDC message.

`length` – length of the LDC messages in bytes, counting from the first byte of the next field to the last byte of the LDC message. The value "0" is not valid for this field.

`base_presentation_time_offset` – provides information about the time difference between decoding time and presentation time in microseconds. Presentation time of each sample shall be greater than the decoding time with this value. This shall not include any difference between the decoding time and presentation time of samples incurred due to reordering of decoded media data.

`coding_dependency_structure_flag` – provides an indication that the decoding order and presentation order of samples are different each other. If this flag is set to "0", the decoding order shall be same with the presentation order of samples. If this flag is set to "1", the decoding order shall be different from the presentation order of samples and detailed composition time offset shall be provided in this message for the client to calculate the appropriate decoding time and presentation time of the samples.

`period_of_intra_coded_sample` – provides the number of samples between two independently coded samples.

`sample_composition_time_offset_sign` – provides the arithmetic sign of offset added to the difference between decoding time and composition time of sample.

`sample_composition_time_offset_value` – provides the value of the offset added to the difference between the decoding time and the composition time. If the value of `sample_composition_time_offset_sign` is "0", then the difference between the value of composition time and the value of decoding time is decreased in microseconds. If the value of `sample_composition_time_offset_sign` is "1", then the difference between the value of composition time and the value of decoding time is increased in microseconds.

10.4.8 Hypothetical receiver buffer model (HRBM) removal message

10.4.8.1 General

The HRBM removal message provides information on the management of the MMT de-capsulation buffer depending on the operation mode of the client as specified in Clause 11. This message provides information required to calculate both the initial delay before starting the removal of data from the MMTP de-capsulation buffer and the rate of removing data from the MMTP de-capsulation buffer. If this message is signalled, the client shall choose one of operation modes with the maximum required buffer size signalled by this message to prevent overflow or underflow of the MMTP de-capsulation buffer. Depending on the mode chosen by the client, either a complete MPU, a movie fragment or a single MFU is recovered and the reconstructed data is forwarded to the upper layer such as the media engine.

10.4.8.2 Syntax

The syntax for HRBM Removal message is defined in Table 48.

Table 48 — HRBM Removal message syntax

Syntax	Values	No. of bits	Mnemonic
HRBM_Removal_message( ){			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
number_of_operation_modes		**8**	**uimsbf**
for(i=0; i<number_of_operation_mode; i++){			
data_removal_type		**8**	**uimsbf**
max_decapsulation_buffer_size		**32**	**uimsbf**
}			
buffer_management_valid		**1**	**bslbf**
reserved		**7**	**uimsbf**
}			
}			

10.4.8.3 Semantics

`message_id` – indicates the identifier of the `HRBM_Data_Removal` message.

`version` – version of the `HRBM_Data_Removal` messages. An MMT receiving entity can use this field to check the version of the received `HRBM_Data_Removal` message.

`length` – length of the `HRBM_Data_Removal` messages in bytes, counting from the first byte of the next field to the last byte of the `HRBM_Data_Removal` message. The value "0" is not valid for this field.

`number_of_operation_modes` – provides the number of operation modes a client can choose to operate.

`data_removal_type` – provides the information for the type of operation mode of client removing data reconstructed at the MMTP de-capsulation buffer defined in HRBM in Clause 11. For each mode, a required buffer size is provided (The list of available values are in Table 49).

Table 49 — data_removal_type values

Value	Description
0x00	Reserved
0x01	Client can remove complete MPUs (MPU mode)
0x02	Client can remove complete movie fragments (movie fragment mode)
0x03	Client can remove complete MFUs (MFU mode)
0x04 ~ 0x9F	Reserved for ISO use
0xA0 ~0xFF	Reserved for private use

`max_decapsulation_buffer_size` – provides the information for the required maximum size of the MMTP de-capsulation buffer in bytes of MMT Assets.

`buffer_management_valid` – provides the information whether the buffer management mechanism defined for an Asset is applied. If this flag is set to "0", no restriction to both the initial delay before starting the removal of data from the MMTP de-capsulation buffer and the rate of removing data from

the MMTP de-capsulation buffer are applied. Reconstructed data shall be available at the MMTP de-capsulation buffer until the buffer becomes full. Reconstructed data shall be removed from the oldest one according to the operation mode chosen by the client when the buffer is full to add newly-recovered data. If this flag is set to "1", appropriate information to calculate both the initial delay before starting the removal of data from the MMTP de-capsulation buffer and the rate of removing data from the MMTP de-capsulation buffer shall be carried in the media data based on external specification.

10.4.9 ADC message

10.4.9.1 General

An ADC message carries information on ADC which defines QoS requirements and statistics of Asset for delivery, and their associated QoE quality information in alignment with ISO/IEC 23001-10. Additional operating points can be derived from stream sub-representation via a new `sample_group_index structure`. This is especially useful for low delay streaming applications where quality can be traded for delay reduction. This information can be used by the MMT-aware intermediate network entities for QoS-managed delivery of Assets. To provide more accurate information, the MMT sending entity can update parameter values within ADC message and send it periodically or aperiodically. ADC information delivery can be done both in per-Asset basis and per-MPU basis considering its size.

10.4.9.2 Syntax

The syntax of the ADC message is defined in Table 50.

Table 50 — ADC message syntax

Syntax	Value	No. of bits	Mnemonic
ADC_message () {			
message_id		16	uimsbf
version		8	uimsbf
length		32	uimsbf
message_payload {			
validity_start_time		32	uimsbf
validity_duration		32	uimsbf
ADC_level_flag		1	boolean
flow_label_flag		1	boolean
reserved		6	uimsbf
if (ADC_level_flag == 1){			
MPU_sequence_number		32	uimsbf
}			
packet_id		16	uimsbf
qos_descriptor{			uimsbf
loss_tolerance		8	bslbf
jitter_sensitivity		8	bslbf
class_of_service		1	boolean
bidirection_indicator		1	boolean
reserved		6	bslbf
}			
qoe_descriptor{			
n_samples		16	uimsbf
for (i=0;i<N1; i++) {			

Table 50 *(continued)*

Syntax	Value	No. of bits	Mnemonic
sample_group_index		**16**	**uimsbf**
}			
spatial_quality		**16**	**uimsbf**
temporal_quality		**16**	**uimsbf**
aggregate_rate		**32**	**uimsbf**
}			
`if (class_of_service == 1)`			
bitstream_descriptor_vbr{			
sustainable_rate		**16**	**uimsbf**
buffer_size		**16**	**uimsbf**
peak_rate		**16**	**uimsbf**
max_MFU_size		**8**	**uimsbf**
mfu_period		**8**	**uimsbf**
`}else`			
bitstream_descriptor_cbr{			
peak_rate		**16**	**uimsbf**
max_MFU_size		**8**	**uimsbf**
mfu_period		**8**	**uimsbf**
`}`			
`if(flow_label_flag == 1) {`			
flow_label		**7**	**uimsbf**
reserved		**1**	**uimsbf**
`}`			
`}`			
`}`			

10.4.9.3 Semantics

`message_id` – indicates the identifier of the ADC messages.

`version` – indicates the version of the ADC messages.

`length` – a 32-bit field for conveying the length of the ADC message in bytes, counting from the beginning of the next field to the last byte of the ADC message. The value "0" is not valid for this field.

`validity_start_time` – indicates the time when the updated ADC message starts to be valid in UTC.

`validity_period_duration` – indicates the validity period duration of the updated ADC message from the `validity_start_time` in milliseconds. The value of this parameter within this ADC message is valid until this duration and the MANE does not need to capture the newer ADC message necessarily.

`ADC_level_flag (1 bit)` – indicates whether included ADC information is for an Asset or for an MPU. If set to "0", the ADC signalling message includes information for an Asset. If set to "1", it includes ADC information for a single MPU.

`flow_label_flag (1 bit)` – indicates whether included `flow_label` is used. If this flag is set to "1", the `flow_label` information is used.

`loss_tolerance` – indicates the required loss tolerance of the Asset for the delivery. The value of `loss_tolerance` attribute is listed in Table 1.

jitter_sensitivity – indicates the required jitter level of the underlying delivery network for the Asset delivery between end-to-end. The value of this attribute is listed in Table 2.

class_of_service – classifies the services in different classes and manages each type of bitstream with a particular way. For example, MANE can manage each type of bitstream with a particular way. This field indicates the type of bitstream attribute as listed in Table 3.

Bidirection_indicator – if set to "1", the bidirectional delivery is required. If set to "0", the bidirectional delivery is not required.

bitstream_descriptor_vbr – when class_of_service is "1",
this field shall be used for "Bitstream_descriptorType".

bitstream_descriptor_cbr – when class_of_service is "0", this field shall be used for "Bitstream_descriptorType".

n_samples – defines the samples associated with this particular operating point.

spatial_quality – defines the spatial quality associated with samples that are conforming to the ISOBMFF quality format. Examples are PSNR and MSE.

temporal_quality – defines the temporal quality or distortion associated with samples that are computed from the ISOBMFF frame significance values.

aggregate_rate – defines the bitrate associated with the operating point.

sustainable_rate – defines the minimum bitrate that shall be guaranteed for continuous delivery of the Asset. The value of this field corresponds to drain rate in the token bucket model. The value of this field is expressed in bytes per second.

buffer_size – defines the maximum buffer size for delivery of the Asset. The buffer absorbs excess instantaneous bitrate higher than the value of sustainable_rate and the buffer_size shall be large enough to avoid overflow. The value of this field corresponds to bucket depth in the token bucket model. The value of this field of a constant bit rate (CBR) Asset shall be zero. The value of this field is expressed in bytes.

peak_rate – defines peak bitrate during continuous delivery of the Asset. The value of this field is the highest bitrate during every MFU_period. The value of this field is expressed in bytes per second.

MFU_period – defines period of MFUs during continuous delivery of the Asset. The MFU_period measured as the time interval of sending time between the first byte of two consecutive MFUs. The value of this field is expressed in millisecond.

max_MFU_size – indicates the maximum size of MFU, which is MFU_period*peak_rate. The value of this field is expressed in bytes.

flow_label – indicates the flow identifier. The application can perform per-flow QoS operations in which network resources are temporarily reserved during the session. A flow is defined to be a bitstream or a group of bitstreams whose network resources are reserved according to transport characteristics or ADC in Package. It is an implicit serial number from "0" to "127". An arbitrary number is assigned temporarily during a session and refers to every individual flow for whom a decoder (processor) is assigned and network resource can be reserved.

packet_id (16 bits) – this field is an integer value that can be used to distinguish one Asset from another. The value of this field is derived from the asset_id of the Asset where this packet belongs to. The mapping between the packet_id and the asset_id is signalled by the Package Table as part of a signalling message. The packet_id is unique throughout the lifetime of the delivery session and for all MMT flows delivered by the same MMT sending entity. For AL-FEC, the mapping between the packet_id and the FEC repair flow is provided in the AL-FEC message (see C.6).

10.4.10 NAT_Keepalive (NK) message

10.4.10.1 General

NK message contains session notification related information to avoid the NAT traversal problem in heterogeneous networks. MMTP over UDP has less overhead and delay so that it can provide real-time delivery, but there can be NAT traversal problem especially in heterogeneous mobile networks.

10.4.10.2 Syntax

The syntax for the NK message is shown in Table 51

Table 51 — NK Message Syntax

Syntax	Value	No. of bits	Mnemonic
NK_message () {			
message_id		16	uimsbf
version		8	uimsbf
length		16	uimsbf
message_payload{			
ip_addr_version		2	uimsbf
reserved	'11 1111'	6	uimsbf
MMT_package_id {			
MMT_package_id_length	N1	8	uimsbf
for (i=0; i<N1; i++) {			
MMT_package_id_byte		8	uimsbf
}			
}			
if (ip_addr_version == 00) {			
ipv4_src_addr		32	uimsbf
} else if (ip_addr_version == 01) {			
ipv6_src_addr		128	uimsbf
}			
src_port		16	uimsbf
mobile_info_descriptor()			
}			
}			

10.4.10.3 Semantics

`message_id` – indicates the message ID. The length of this field is 16 bits.

`version` – indicates the version of the messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the NK message. The value "0" shall not be used for this field.

`ip_addr_version` – this field indicates the version of the IP address as defined in Table 52.

Table 52 — Value of address_type

Value	Description
00	MMTP packet flow over UDP/IP (version 4)
01	MMTP packet flow over UDP/IP (version 6)
10~11	reserved

`MMT_package_id` – this field is a unique identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`ipv4_src_addr` – This field is the IPv4 address of the sender of the packet. The length of this field is 32 bits.

`ipv6_src_addr` – This field is the IPv6 address of the sender of the packet. The length of this field is 128 bits.

`src_port` – This field indicates the sending port number. The length of this field is 16 bits.

`mobile_info_descriptor` – indicates a descriptor defined in 10.5.7.

10.4.11 Media Resource Identification (MRI) message

10.4.11.1 General

An MMT sending entity can send the media resource identification information to the MMT receiving entity to provide the information of media resource (MMTP) allocation information by using MRI messages. By using the MRI message, the MMT sending entity notify the media content locations, transmission delivery type information, schedule information to MMT receiving entity.

10.4.11.2 Syntax

Syntax of the MRI message is shown in Table 53.

Table 53 — MRI message syntax

Syntax	Value	No. of bits	Mnemonic
MRI_message ( ) {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
number_of_MMT_package	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++ {			
MMT_package_id {			
MMT_package_id_length	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
delivery_type		**4**	**uimsbf**

Table 53 *(continued)*

Syntax	Value	No. of bits	Mnemonic
reserved		**4**	**uimsbf**
number_of_assets	**N3**	**8**	**uimsbf**
for (i=0; i<N3; i++) {			
asset_id()			
location_count	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
mpu_sequence_start_number		**32**	**uimsbf**
mpu_sequence_end_number		**32**	**uimsbf**
MMT_general_location_info()			
valid_time_start		**64**	**uimsbf**
valid_time_duration		**64**	**uimsbf**
}			
}			
consumption_MMT sending entity_address{			
MMT_general_location_info()			
}			
}			
}			
}			

10.4.11.3 Semantics

`message_id` - indicates MRI message ID. The length of this field is 16 bits.

`version` - indicates the version of MRI messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` - indicates the length of MRI messages. The length of this field is 16 bits. It indicates the length of the MRI message counted in bytes starting from the next field to the last byte of the MRI message. The value '0' shall not be used.

`number_of_MMT_package_id` – indicate the number of MMT_package.

`MMT_package_id` – this field is a unique identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the MMT_package_id. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`delivery_type` – indicates type of MMT session of the MMT package delivery as shown in Table 54.

Table 54 — Value of delivery_type field

Value	Description
0001	Broadcast delivery only
0010	Unicast delivery only
0011	Hybrid delivery
0100 ~ 1111	Reserved

`number_of_assets` – provides the number of Assets in each MMT Package.

`asset_id` – identifier of Asset which is included in the current MMT package.

`location_count` – provides the number of location information for an Asset. Set to '1' when an Asset is delivered through one location. When bulk delivery is achieved, in which MPUs contained in an Asset are delivered through multi-channels, not '1' is set.

`mpu_sequence_start_number` – indicates the first MPU sequence number of each location in an asset.

`mpu_sequence_end_number` – indicates the last MPU sequence number of each location in an asset.

`MMT_general_location_info_` – provides the location information of Asset. Syntax and semantics of `MMT_general_location_info` are defined in subclause 10.6.1.

`valid_time_start` – indicates a UTC time in NTP format corresponding to valid start time of MMT session.

`valid_time_duration` – indicates a UTC time in NTP format corresponding to valid stop time of MMT session.

`consumption_MMT_sending_entity_address` – indicates the location of MMT sending entity that receives measurement results from each MMT receiving entity. The syntax and semantics are the same of `MMT_general_location_info` that is defined in subclause 10.6.1.

10.4.12 Consumption reporting (CR) message

10.4.12.1 General

An MMT receiving entity can send the consumption reporting information to the MMT sending entity or 3rd party application MMT sending entity to provide the information of received MMTP by using CR messages. An MMT receiving entity provides information in this message to allow the MMT sending entity to count the number of MMT receiving entities which receive specific media content's MMTP packet. By using the CR message, the MMT sending entity decides transmission access type which means that MMT sending entity send the MMTP packet of specific media content to MMT receiving entity through the determined transmission access type.

10.4.12.2 Syntax

Syntax of the CR message is shown in Table 55.

Table 55 — CR message syntax

Syntax	Value	No. of bits	Mnemonic
CR_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
MMT_package_id {			
MMT_package_id_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++)			
MMT_package_id_byte		**8**	**uimsbf**
}			
mobile_information_descriptor()			
number_of_assets	**N2**	**8**	**uimsbf**

Table 55 *(continued)*

Syntax	Value	No. of bits	Mnemonic
for (i=0; i<N2; i++){			
asset_id()			
mpu_timestamp_descriptor()			
delivery_type		**4**	**uimsbf**
reserved		**4**	**uimsbf**
}			
}			
}			

10.4.12.3 Semantics

`message_id` – indicates CR message ID. The length of this field is 16 bits.

`version` – indicates the version of CR messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of CR messages. The length of this field is 16 bits. It indicates the length of the MRI message counted in bytes starting from the next field to the last byte of the CR message. The value '0' shall not be used.

`MMT_package_id` – this field is a unique identifier of the package that MMT receiving entity received

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the MMT_package_id. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`mobile_information_descriptor()` – contains information which is associated with MMT receiving entity in case it is a mobile device. When MMT receiving entity is a cellular mobile device, it can have mobile-specific attributes including MSISDN or Cell ID which are also defined in 3GPP.

`asset_id` – identifier of Asset as defined in 10.6.2 which is included in the current MMT package.

`mpu_timestamp_descriptor` – provides presentation time of the first AU of MPU in presentation order after application of any offset such as the one provided by 'elst' box. When presentation information presents this descriptor shall be ignored.

`delivery_type` – indicates type of MMT session of the MMT package delivery as shown in Table 56.

Table 56 — Value of delivery_type field

Value	Description
0001	Broadcast delivery only
0010	Unicast delivery only
0011	Hybrid delivery
0100 ~ 1111	Reserved

10.4.13 Distributed Resource Identification (DRI) message

10.4.13.1 General

DRI message contains the alternative media source location information for supporting the dynamic media resource access control. When the media source location is dynamically allocated, MMT sending

entity or MANE sends the DRI message to MMT receiving entity to inform the alternative media source locations.

10.4.13.2 Syntax

Syntax of the DRI message is shown in Table 57.

Table 57 — DRI message syntax

Syntax	Value	No. of bits	Mnemonic
DRI_message ( ) {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
MMT_package_id {			
MMT_package_id_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++)			
MMT_package_id_byte		**8**	**uimsbf**
}			
number of assets	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
asset_id()			
packet_id		**16**	**uimsbf**
priority_type		**2**	**uimsbf**
reserved	**'11 1111'**	**6**	**uimsbf**
valid_time_start		**64**	**uimsbf**
valid_time_duration		**64**	**uimsbf**
if(priority_type == 00){			
primary_media_source_location{			
MMT_general_location_info()			
delivery_type ()			
}			
} elseif(priority_type == 01){			
alternative_media_source_location{			
number_of_alternative	**N3**	**8**	**uimsbf**
for(i=0; N3; i++) {			
alternative_media_source_id		**8**	**uimsbf**
MMT_general_location_info()			
delivery_type		**4**	**uimsbf**
}			
} elseif(priority_type == 10){			
origin_media_source_location {			
MMT_general_location_info()			
delivery_type		**4**	**uimsbf**
}			
}			

Table 57 *(continued)*

Syntax	Value	No. of bits	Mnemonic
} } }			

10.4.13.3 Semantics

`message_id` – indicates DRI message ID. The length of this field is 16 bits.

`version` – indicates the version of DRI messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of DRI messages. The length of this field is 16 bits. It indicates the length of the DRI message counted in bytes starting from the next field to the last byte of the DRI message. The value '0' shall not be used.

`MMT_package_id` – this field is a unique identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the MMT_package_id. When the `MMT_package_id`_byte is string, the terminating null character is not included in the string.

`number_of_assets` – provides the number of Assets whose information is provided by MP table.

`asset_id` – indicates the `asset_id` of another Asset on which the dependent Asset associated with this descriptor depends. The order of the id-s provided in this descriptor is such that the concatenation of MPUs as defined in section 6.3 leads to a valid MPU and follows the media dependency hierarchy.

`packet_id` – provides the identifier of the MMTP session in the MMTP packet header

`priority_type` – indicates the priority of media source location indicator. The four values are listed in Table 58. If the value is set to '00', the MMT receiving entity can receive media source from the primary media source location.

Table 58 — Values of priority_type

Value	Description
'00'	Primary media source location
'01'	Alternative media source location
'10'	Origin media source location
'11'	Reserved

`valid_time_start` – indicates a UTC time in NTP format corresponding to transmission session valid start time.

`valid_time_duration` – indicates a UTC time in NTP format corresponding to transmission session valid stop time.

`primary_media_source_location` – provides the information of the media source location and delivery type that the MMT receiving entity can get the Asset from the primary media source location.

`alternative_media_source_location` – provides the information of the media source location and delivery type that the MMT receiving entity can get the Asset from the alternative media source location.

`Number_of_alternative` – provides the number of alternative m edia source location information for an Asset. Set to '1' when an Asset is delivered through one location. When bulk delivery is achieved, in which MPUs contained in an Asset are delivered through multi-channels, not '1' is set.

`number_of_alternative` – provides the number of alternative media source location.

`alternative_media_source_id` – indicates a unique identifier of the alternative media sources

`origin_media_source_location` – provides the information of the media source location and delivery type that the MMT receiving entity can get the Asset from the origin media source location.

`MMT_general_location_info_` – provides the location information of Asset. General location reference information for Asset defined in 10.6.1 is used. Only the value of `location_type` between 0x00 and 0x06 shall be used for an Asset location.

`delivery_type` – indicates type of transmission access channel of media content delivery as shown in Table 59.

Table 59 — Value of delivery_type field

Value	Description
0001	Broadcast delivery only
0010	Unicast delivery only
0011	Hybrid delivery
0100 ~ 1111	Reserved

10.4.14 Distributed Signaling Information (DSI) message

10.4.14.1 General

DSI message contains the alternative media signaling information locations for supporting the dynamic media resource access control. When the media source delivery characteristic is dynamically changed, MMT sending entity or MANE sends the DSI message to MMT receiving entity to inform the alternative media signaling locations.

10.4.14.2 Syntax

Syntax of the DSI message is shown in Table 60.

Table 60 — DSI message syntax

Syntax	Value	No. of bits	Mnemonic
DSI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
MMT_package_id {			
MMT_package_id_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++)			
MMT_package_id_byte			
}			
number_of_mmt_signaling	**N2**	**8**	**uimsbf**

Table 60 *(continued)*

Syntax	Value	No. of bits	Mnemonic
for (i=0; i<N2; i++) {			
message_id()			
priority_type		**2**	**uimsbf**
reserved	**'11 1111'**	**6**	**uimsbf**
valid_time_start		**64**	**uimsbf**
valid_time_duration		**64**	**uimsbf**
if(priority_type == 00){			
primary_signaling_location {			
MMT_general_location_info()			
delivery_type			
}			
}			
elseif(priority_type == 01){			
alternative_signaling_location {			
number_of_alternative	**N3**	**8**	**uimsbf**
for(i=0; N3; i++) {			
alternative_MMT_sending_entity_id		**8**	**uimsbf**
MMT_general_location_info()			
delivery_type		**4**	**uimsbf**
}			
}			
}			
elseif(priority_type == 10){			
origin_signaling_location {			
MMT_general_location_info()			
access_type()			
}			
}			
}			
}			
}			

10.4.14.3 Semantics

`message_id` – indicates DSI message ID. The length of this field is 16 bits.

`version` – indicates the version of DSI messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of DSI messages. The length of this field is 16 bits. It indicates the length of the DSI message counted in bytes starting from the next field to the last byte of the DSI message. The value '0' shall not be used.

`MMT_package_id` – this field is a unique identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the MMT_package_id. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`number of mmt signaling` – provides the number of signaling message referenced by DSI message.

`priority_type` – indicates the priority of media source location indicator. The four values are listed in Table 61. If the value is set to '00', the MMT receiving entity can receive media source from the primary media source location.

Table 61 — Values of priority_type

Value	Description
'00'	Primary media source location
'01'	Alternative media source location
'10'	Origin media source location
'11'	Reserved

`valid_time_start` – indicates a UTC time in NTP format corresponding to valid start time of the signaling message transmission session.

`valid_time_duration` – indicates a UTC time in NTP format corresponding to valid time duration of the signaling message transmission session.

`message_id` – indicates the message ID. The length of this field is 16 bits

`primary_signaling_location` – provides the information of the signaling message location and delivery type that the MMT receiving entity can get the signaling message from the primary media signaling location.

`alternative_signaling_location` – provides the information of the signaling message location and delivery type that the MMT receiving entity can get the signaling message from the alternative media signaling location.

`Number_of_alternative` – provides the number of alternative media signaling locations.

`origin_signaling_location` – provides the information of the signaling message location and delivery type that the MMT receiving entity can get the signaling message from the origin media signaling location.

`MMT_general_location_info` – provides the location information of Asset. General location reference information for Asset defined in 10.6.1 is used. Only the value of `location_type` between 0x00 and 0x06 shall be used for an Asset location.

`delivery_type` – indicates type of transmission access channel of media content delivery as shown in Table 62.

Table 62 — Value of delivery_type field

Value	Description
0001	Broadcast delivery only
0010	Unicast delivery only
0011	Hybrid delivery
0100 ~ 1111	Reserved

10.4.15 Bandwidth Probing reQuest (BPQ) message

10.4.15.1 General

In order to support adaptive streaming in a mobile network, its available bandwidth should be monitored. BPQ (Bandwidth Probing reQuest) message is defined to provide a mechanism for probing the available bandwidth in mobile networks. BPQ message informs an MMT MMT receiving entity that a probing event will be happen and contains a configuration information for the probing event.

10.4.15.2 Syntax

The syntax for the proposed BPQ is shown in Table 63.

Table 63 — BPQ Message Syntax

Syntax	Values	No. of bits	Mnemonic
BPQ_message(){			
message_id		16	uimsbf
version		8	uimsbf
length		16	uimsbf
message_payload {			
MMT_package_id_flag		1	bslbf
CLI_id_flag		1	bslbf
reporting_MMT sending entity_location_flag		1	bslbf
reserved	'1 1111'	5	
if (MMT_package_id_flag) {			
MMT_package_id {			
MMT_package_id_length	N1	8	uimsbf
for (i=0;i<N1;i++){			
MMT_package_id_byte		8	uimsbf
}			
}			
}			
If (CLI_id_flag) {			
CLI_id		8	uimsbf
}			
reporting_MMT sending entity_location {			
MMT_general_location_info()		var	
}			
probing_packet_location {			
MMT_general_location_info()		var	
}			
probing_start_time		32	uimsbf
probing_slot_duration		16	uimsbf
probing_slot_interval		16	uimsbf
probing_slot_number		4	uimsbf
reserved	'1111'	4	
}			

Table 63 *(continued)*

Syntax	Values	No. of bits	Mnemonic
}			

10.4.15.3 Semantics

`MMT_package_id_flag` – indicates whether `MMT_package_id` is present or not. When this flag is set to '1', MMT_package_id shall be present. When this flag is set to '0', `MMT_package_id` shall not be present.

`CLI_id_flag` – indicates whether `CLI_id` is present or not. When this flag is set to '1', `CLI_id` shall be present. When this flag is set to '0', `CLI_id` shall not be present.

`reporting_MMT_sending_entity_location_flag` – indicates whether `reporting_MMT_sending_entity_location` is present or not. When this flag is set to '1', `reporting_MMT_sending_entity_location` shall be present. When this flag is set to '0', `reporting_MMT_ sending_entity_location` shall not be present.

`MMT_package_id` – a unique identifier of the Package. It shall be present only when the value of `MMT_package_id_flag` is set to 1.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`.

`CLI_id` – an arbitrary integer number to identify NAM which will be used to transmit probing packets among the underlying network. It shall be present only when the value of `CLI_id_flag` is set to 1.

`reporting_MMT_sending_entity_location` – provides the location information of a reporting MMT sending entity where the probing result has to be reported. Its syntax and semantics are given in 10.6.1. Only the value of location_type between "0x05" and "0x06" shall be used for an location of the reporting MMT sending entity for probing. It shall be present only when the value of `reporting_MMT_sending_entity_location_flag` is set to 1.

`probing_packet_location` – provides the location information of MMTP packets for probing. Its syntax and semantics are given in 10.6.1. Only the values of location_type between "0x00" and "0x02" shall be used for an location of MMTP packets for probing.

`probing_start_time` – indicates the starting time of a probing event. This field shall be formatted as the "short-format" defined in IETF RFC 5905, NTP version 4, Clause 6. The value of this field shall be the same as the value of the timestamp field in the MMTP header of the first probing packet.

`probing_slot_duration` – indicates the duration of a probing slot consisting a probing event signaled by this message in units of milliseconds. A probing event is a group of one or more probing slots which have the same duration and interval.

`probing_slot_interval` – indicates the interval between probing slots in units of milliseconds.

`probing_slot_number` – indicates one less than the number of probing slots consisting a probing event signaled by this message minus 1. A value 0 indicates that the probing event has only one probing slots, A value 1 indicates that it has two probing slots, etc.

10.4.16 Bandwidth Probing Response (BPR) message

10.4.16.1 General

In order to support adaptive streaming in a mobile network, its available bandwidth should be monitored. BPR (Bandwidth Probing Response) message is defined to provide a mechanism for probing

the available bandwidth in mobile networks. BPR message contains a measurement result of a probing event and information on the MMT receiving entity.

10.4.16.2 Syntax

The syntax for the BPR message is shown in Table 64.

Table 64 — BPR Message Syntax

Syntax	Values	No. of bits	Mnemonic
BPQ_message(){			
message_id		16	uimsbf
version		8	uimsbf
length		16	uimsbf
message_payload {			
MMT_package_id_flag		1	bslbf
CLI_id_flag		1	bslbf
Mobile_info_descriptor_flag		1	bslbf
reserved	'1 1111'	5	
if (MMT_package_id_flag) {			
MMT_package_id {			
MMT_package_id_length	N1	8	uimsbf
for (i=0;i<N1;i++) {			
MMT_package_id_byte		8	uimsbf
}			
}			
}			
if (CLI_id_flag) {			
CLI_id		8	uimsbf
}			
if (Mobile_info_descriptor_flag) {			
Mobile_info_descriptor()		var	
}			
average_rate		32	uimsbf
peak_rate		32	uimsbf
packet_loss_ratio		8	uimsbf
}			
}			

10.4.16.3 Semantics

MMT_package_id_flag – indicates whether MMT_package_id is present or not. When this flag is set to '1', MMT_package_id shall be present. When this flag is set to '0', MMT_package_id shall not be present.

CLI_id_flag – indicates whether CLI_id is present or not. When this flag is set to '1', CLI_id shall be present. When this flag is set to '0', CLI_id shall not be present.

Mobile_info_descriptor_flag – indicates whether Mobile_info_descriptor() is present or not. When this flag is set to '1', Mobile_info_descriptor() shall be present. When this flag is set to '0', Mobile_info_descriptor() shall not be present.

MMT_package_id – a unique identifier of the Package. It shall be present only when the value of MMT_package_id_flag is set to 1.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`.

`CLI_id` – an arbitrary integer number to identify NAM for the probing event reported by this message among the underlying network. It shall be present only when the value of `CLI_id_flag` is set to 1.

`Mobile_info_descriptor()` – contains information which is associated with an MMT receiving entity in a mobile network. Its syntax and semantics are given in 10.5.7. It shall be present only when the value of `Mobile_info_descriptor_flag` is set to 1.

`average_rate` – indicates the average bitrate of received MMTP packets measured during a probing event in units of kilobits per second. The value of this field is the average of the bitrate of each probing slot and the bitrate of each probing slot is calculated by dividing the total number of bits in MMTP packets received during the probing slot by the duration of it.

`peak_rate` – indicates the peack bitrate in a probing event in units of kilobits per second. The value of this field shall be set to as the maximum value over all bitrates for each probing slot.

`packet_loss_ratio` – indicates the ratio between the number of lost MMTP packets and the total number of transmitted packets within the probing event. The value of this field shall be calculated by taking the integer part after multiplying the loss fraction by 256.

10.4.17 Pacing Buffer Removal Rate (PRR) message

10.4.17.1 General

The Pacing Buffer Removal Message provides information on the drain rate of the MMTP pacing buffer. This message shall be signaled when the MMTP pacing buffer is used. If this message is signaled, the MMT receiving entity shall deliver the MMTP packets in the MMTP pacing buffer to the FEC decoding buffer in the MMTP HRBM at the rate specified in this message.

10.4.17.2 Syntax

The syntax for the PRR message is shown in Table 65.

Table 65 — PRR Message Syntax

Syntax	Values	No. of bits	Mnemonic
`Pacing_Buffer_Removal_Rate ( ){`			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
`message_payload {`			
pacing_buffer_removal_rate		**32**	**uimsbf**
`}`			
reserved	**'1111 1111'**	**8**	**uimsbf**
`}`			

10.4.17.3 Semantics

`message_id` - indicates the identifier of the Pacing Buffer Removal Rate message.

`version` - version of the Pacing Buffer Removal Rate messages. An MMT receiving entity can use this field to check the version of the received Pacing Buffer Removal Rate message.

length - length of the Pacing Buffer Removal Rate messages in bytes, counting from the first byte of the next field to the last byte of the Pacing Buffer Removal Rate message. The value '0' is not valid for this field.

pacing_buffer_removal_rate - provides the rate to deliver the MMTP packets from the MMTP pacing buffer to the FEC decoding buffer in bits per seconds.

10.4.18 Pacing Buffer Status Feedback (PSF) message

10.4.18.1 General

As the size of the MMTP pacing buffer and the amount of time for each MMTP packets are stored in the MMTP pacing buffer are decided by an MMT receiving entity, an MMT receiving entity shall send the pacing buffer status feedback messages when requested by the MMT sending entityMMT sedning entity and at the frequency requested by the MMT sedning entityMMT sending entity. An MMT receiving entity needs to keep track of the status of the MMTP pacing buffer level and decides the time for each MMTP packets to be sent to control the amount of excessive data stored at the MMTP pacing buffer.

NOTE The frequency is provided as part of the MC message.

10.4.18.2 Syntax

The syntax for the PSF message is shown in Table 66.

Table 66 — Syntax for PSF message

Syntax	Values	No. of bits	Mnemonic
PSF_message ( ) {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
num_sub_flows	N	**8**	**uimsbf**
for (i=0;i<N;i++) {			
packet_id		**16**	**uimsbf**
current_pacing_buffer_level		**32**	**uimsbf**
target_pacing_buffer_level		**32**	**uimsbf**
last_received		**32**	**uimsbf**
pacing_buffer_free_space		**32**	**uimsbf**
}			
}			
}			

10.4.18.3 Semantics

message_id - identifier of the PSF message. The length of this field is 16 bits.

version - version of the PSF message. The receiver can use this field to check the version of a received message. The length of this field is 8 bits.

length - length of the PSF message in bytes, counting from the first byte of next field to the last byte of the PSF message. The length of this field is 16 bits and the value '0' is not valid for this field.

current_pacing_buffer_level - informs an MMT sending entity about the amount of the MMTP packets currently buffered in the MMTP pacing buffer in bytes.

target_pacing_buffer_level - informs an MMT sending entity about the target amount of MMTP packets buffered in the MMTP pacing buffer in bytes. An MMTP sending entity shall adjust the bandwidth allocated to the delivery of MMTP packets to meet this level

last_received - informs the MMT sending entity about the last received MMTP packet sequence number for this particular MMTP sub-flow.

pacing_buffer_free_space - informs an MMT sending entity about the amount of the available free space of the MMTP pacing buffer in bytes.

10.4.19 Cell congestion information messages

10.4.19.1 General

An CCI signalling message delivers the information of network status which is measured by MANE or inter-network entities between the MMT sending entity and the MMT receiving entity. An CCI message contains the identification information of network entities and its related network status information which uses to provide the media aware delivery fuctions to avoid the media quality degradation by MMT sending entity. This information can be used by the MANE for QoS-managed delivery of Assets to report the MMT sending entity. Annex J provies an example use case this message can be used.

10.4.19.2 Syntax

The syntax for CCI message is shown in Table 67.

Table 67 — CCI message syntax

Syntax	Values	No. of bits	Mnemonic
CCI_message(){			
message_id		16	uimsbf
version		8	uimsbf
length		16	uimsbf
message_payload{			
numberOflist	N1	8	uimsbf
for(i=0;i<N1;i++){			
mobile_info_descriptor()			
levelOfnetworkstatus		4	uimsbf
reserved	'1111'	4	uimsbf
}			
reportingTime		32	uimsbf
validDuration		32	uimsbf
}			
}			

10.4.19.3 Semantics

message_id – indicates the identifier of the CCI message.

version – version of the CCI messages. An MMT sending entity can use this field to check the version of the received CCI message.

length – length of the CCI messages in bytes, counting from the first byte of the next field to the last byte of the CCI message. The value '0' is not valid for this field.

numberOflist – specifies the number of list contained in mobile_info_descriptor. This field shall not be set to '0'.

`mobile_info_descriptor` – indicates a descriptor defined in 10.5.7.

`levelOfnetworkstatus` – indicates the level of the network status which classifies the conditions of the underlying network in MANE.

`reportingTime` – is reporting time of current network status which is based on NTP timestamp. The reportingTime is expressed in milliseconds.

`validDuration` – indicates the valid period of this reporting information from the reportingTime in milliseconds. The value of this parameter within this CCI message is vaild until this duration.

10.4.20 MMT Transition Request (MTR) message

10.4.20.1 General

An MMT receiving entity shall send an MMT transition request message to MMT sending entity when it wants to change the delivery path of a Package. The MTR message provides identifiers of both the original delivery path and new delivery path. The MTR message defines the transition time using the information about the last MMTP packet, and the las MPU received by the MMT receiving entity. The information about the last sample can be used to indicate more precisely. The MMT sending entity shall send the Package through the original delivery path until the transition time, and after the transition time, it shall continue sending the Package through the new delivery path.

10.4.20.2 Syntax

The syntax for the MMT Transition Request message is shown in Table 68.

Table 68 — Syntax for MTR Message

Syntax	Value	No. of bits	Mnemonic
MTR_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
IP_version		**2**	**bslbf**
reserved	**'11 1111'**	**6**	**bslbf**
if (IP_version == 00) {			
origin_address		**32**	**uimsbf**
transition_ address		**32**	**uimsbf**
}			
else if (IP_version== 01) {			
origin_ address		**128**	**uimsbf**
transition_ address		**128**	**uimsbf**
}			
origin_port_number		**16**	**uimsbf**
transition_port_number		**16**	**uimsbf**
MMT_package_id {			
MMT_package_id_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			

Table 68 *(continued)*

Syntax	Value	No. of bits	Mnemonic
}			
number_of_assets	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
asset_id()			
packet_id		**16**	**uimsbf**
packet_sequence_number		**32**	**uimsbf**
mpu_sequence_number		**32**	**uimsbf**
mpu_presentation_time		**64**	**uimsbf**
sample_flag		**1**	**bslbf**
if (sample_flag == 1) {			
movie_fragment_sequence_number		**32**	**uimsbf**
sample_number		**32**	**uimsbf**
}			
}			
}			
}			

10.4.20.3 Semantics

`message_id` – indicates MTR message ID. The length of this field is 16 bits.

`version` – indicates the version of MTR messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the MTR message. The value "0" shall not be used for this field.

`IP_version` – indicates version of IP address of delivery path. The values of `IP_verison` are defined in Table 69.

Table 69 — Value of IP_version field

Value	Description
00	IPv4
01	IPv6
10 ~ 11	Reserved

`origin_ address` – indicates destination IP address of original delivery path.

NOTE The IP address of origin is used for identifying each MMT receiving entity. Therefore, it has to be the public IP address in case of NAT, so the MMT sending entity or MANE need not to know the private IP address of the network has.

`transition_ address` – indicates destination IP address of new delivery path for this MMT transition.

NOTE The IP address of transition is used for identifying each MMT receiving entity. Therefore, it has to be the public IP address in case of NAT, so the MMT sending entity or MANE need not to know the private IP address of the network has.

`origin_port_number` – indicates destination port number of the original delivery path.

`transition_port_number` – indicates destination port number of the new delivery path.

`MMT_package_id` – provides identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`number_of_assets` – indicates the number of Assets included in the Package.

`asset_id` – provides identifier of the Asset.

`packet_id` – provides identifier of the packet which belongs to each Asset.

`packet_sequence_number` – indicates the sequence number of the last packet received by the MMT receiving entity through the original delivery path.

`mpu_sequence_number` – indicates the sequence number of the last MPUreceived by the MMT receiving entity through the original delivery path. From the next MPU, the MMT sending entity shall start delivering through the new delivery path. If a transition is needed in the middle of delivery an MPU, then this field shall indicate the sequence number of the previous MPU which was delivered through the original delivery path.

`mpu_presentation_time` – indicates the presentation time of the MPU indicated by mpu_sequence_number in 64 bit NTP time stamp format.

`sample_flag` – indicates the existence of `sample_number`.

`movie_fragment_sequence_number` – indicates the sequence number of the movie fragment to which the sample indicated by `sample_number` belongs (see ISO/IEC 14496-12:2022, 8.8.5).

`sample_number` – indicates the sample number of the last sample received by the MMT receiving entity through the original delivery path (see ISO/IEC 14496-12:2022, 8.8.8).

10.4.21 MMT Transition Notification (MTN) message

10.4.21.1 General

An MMT sending entity shall send an MMT transition notification message to MMT receiving entity when it wants to change the delivery path of a Package. The MTN message provides identifiers of both the original delivery path and new delivery path. If the MMT transition was requested by an MTR message, then the MTN message shall be sent as a response of the request, containing the same delivery path identifiers and media information (e.g. Package, Assets, packet, sample related information) as confirmation. The MTN message defines the transition time using the delivery starting time of the new delivery path and the delivery end time of the original delivery path. When the MMT sending entity wants to hand over the responsibility of providing the transition time to another MMT sending entity, the MTN message shall contain the location information of the new MMT sending entity.

10.4.21.2 Syntax

The syntax for the MMT Transition Notification message is shown in Table 70.

Table 70 — Syntax for MTN Message

Syntax	Value	No. of bits	Mnemonic
`MTN_message () {`			
message_id		**16**	**uimsbf**

Table 70 *(continued)*

Syntax	Value	No. of bits	Mnemonic
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
transition_request_flag		**1**	**bslbf**
reserved	**'111 1111'**	**7**	**bslbf**
if (transition_request_flag == 1) {			
IP_version		**2**	**bslbf**
reserved	**'11 1111'**	**6**	**bslbf**
if (IP_version == 00) {			
origin_ address		**32**	**uimsbf**
transition_ address		**32**	**uimsbf**
}			
else if (IP_version == 01) {			
origin_ address		**128**	**uimsbf**
transition_ address		**128**	**uimsbf**
}			
origin_port_number		**16**	**uimsbf**
transition_port_number		**16**	**uimsbf**
transition_start_time		**32**	**uimsbf**
origin_end_time		**32**	**uimsbf**
}			
else if (transition_request_flag == 0) {			
IP_version		**2**	**bslbf**
reserved	**'11 1111'**	**6**	**bslbf**
if (IP_version == 00) {			
origin_ address		**32**	**uimsbf**
transition_ address		**32**	**uimsbf**
}			
else if (IP_version == 01) {			
origin_ address		**128**	**uimsbf**
transition_ address		**128**	**uimsbf**
}			
origin_port_number		**16**	**uimsbf**
transition_port_number		**16**	**uimsbf**
transition_scheduled_flag		**1**	**bslbf**
reserved	**'111 1111'**	**7**	**bslbf**
if (transition_scheduled_flag == 0) {			
MMT_general_location_info()			
}			
else if (transition_scheduled_flag == 1) {			
transition_start_time		**32**	**uimsbf**
origin_end_time		**32**	**uimsbf**

Table 70 *(continued)*

Syntax	Value	No. of bits	Mnemonic
}			
}			
MMT_package_id {			
MMT_package_id_length	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
number_of_assets	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
asset_id()			
packet_id		**16**	**uimsbf**
packet_sequence_number		**32**	**uimsbf**
mpu_sequence_number		**32**	**uimsbf**
mpu_presentation_time		**64**	**uimsbf**
reserved	**'1111 111'**	**7**	**bslbf**
sample_flag		**1**	**bslbf**
if (sample_flag == 1) {			
movie_fragment_sequence_number		**32**	**uimsbf**
sample_number		**32**	**uimsbf**
}			
}			
}			
}			

10.4.21.3 Semantics

`message_id` – indicates MTN message ID. The length of this field is 16 bits.

`version` – indicates the version of MTN messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the MTN message. The value "0" shall not be used for this field.

`transition_request_flag` – indicates whether this MMT transition is requested by MMT receiving entity or not. If this flag is set to '0', this MMT transition is not requested by MMT receiving entity. If this flag is set to '1', this MMT transition is requested by MMT receiving entity.

`IP_version` – indicates version of IP address of delivery path. The values of IP_version are defined in Table 69.

`origin_ address` – indicates destination IP address of original delivery path.

NOTE The IP address of origin is used for identifying each MMT receiving entity. Therefore, it has to be the public IP address in case of NAT, so the MMT sending entity or MANE need not to know the private IP address of the network has.

`transition_address` – indicates destination IP address of new delivery path for this MMT transition.

NOTE The IP address of transition is used for identifying each MMT receiving entity. Therefore, it has to be the public IP address in case of NAT, so the MMT sending entity or MANE need not to know the private IP address of the network has.

`origin_port_number` – indicates destination port number of the original delivery path.

`transition_port_number` – indicates destination port number of the new delivery path.

`transition_start_time` – indicates the time instance at which the Package delivery starts through the new delivery path, based on UTC. The format is the "short-format" as defined in clause 6 of IETF RFC5905, NTP version 4.

`origin_end_time` – indicates the time instance at which the Package delivery ends in the original delivery path, based on UTC. The format is the "short-format" as defined in clause 6 of IETF RFC5905, NTP version 4.

`transition_scheduled_flag` – indicates whether this MMT sending entity has the MMT transition schedule. If this flag is set to '0', additional information can be provided by `MMT_general_location_info` to access to the transition schedule. If this flag set to '1', schedule of MMT transition shall be provided by MMT sending entity in this MTN message.

`MMT_general_location_info` – provides the location where to access to transition schedule.

`MMT_package_id` – provides identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`number_of_assets` – indicates the number of Assets included in the Package.

`asset_id` – provides identifier of the Asset.

`packet_id` – provides identifier of the packet which belongs to each Asset.

`packet_sequence_number` – indicates the sequence number of the last packet received by the MMT receiving entity through the original delivery path.

`mpu_sequence_number` – indicates the sequence number of the last MPU received by the MMT receiving entity through the original delivery path. From the next MPU, the MMT sending entity shall start delivering through the new delivery path. If a transition is needed in the middle of delivery an MPU, then this field shall indicate the sequence number of the previous MPU which was delivered through the original delivery path.

`mpu_presentation_time` – indicates the presentation time of the MPU indicated by mpu_sequence_number in 64 bit NTP time stamp format.

`sample_flag` – indicates the existence of sample_number.

`movie_fragment_sequence_number` – indicates the sequence number of the movie fragment to which the sample indicated by sample_number belongs (see ISO/IEC 14496-12:2022, 8.8.5).

`sample_number` – indicates the sample number of the last sample received by the MMT receiving entity through the original delivery path (see ISO/IEC 14496-12:2022, 8.8.8).

10.4.22 Asset Change Request (ACR) message

10.4.22.1 General

An MMT sending entity shall send the Asset change request (ACR) message while delivering a Package, when it signals an Asset change event, such as splicing with 'replacement' or 'insertion', and other types of Asset change. The ACR message is sent to MANE (Media Aware Network Element) with descriptions of the target Assets to be changed, time information for changes, and location information of media data as the designated Assets which will be applied in the change of target Assets.

When MANE receives the ACR message, MANE shall update MP Table in PA message with corresponding data of the ACR message. The updated PA message shall be sent to MMT receiving entity with the changed Assets.

10.4.22.2 Syntax

The syntax for the Asset Change Request message is shown in Table 71.

Table 71 — Syntax for ACR Message

Syntax	Value	No. of bits	Mnemonic
ACR_message () {			
message_id		16	uimsbf
version		8	uimsbf
length		16	uimsbf
message_payload{			
target_MMT_package_id {			
MMT_package_id_length	N1	8	uimsbf
for (i=0; i<N1; i++) {			
MMT_package_id_byte		8	uimsbf
}			
}			
FEC_Clean_Type		8	uimsbf
number_of_assets	N2	8	uimsbf
for (i=0; i<N2; i++) {			
target_asset_id()			
target_asset_type		32	char
duration		32	uimsbf
change_type		8	uimsbf
change_indicate_mode		8	uimsbf
if (change_indicate_mode == 0x01) {			
UTC_timestamp		32	uimsbf
}			
else if (change_indicate_mode == 0x02) {			
mpu_sequence_number		32	uimsbf
}			
else if (change_indicate_mode == 0x03) {			
mpu_timestamp_descriptor()			
}			

Table 71 *(continued)*

Syntax	Value	No. of bits	Mnemonic
designated_MMT_general_location_info()			
}			
reserved	**'1111 111'**	**7**	**bslbf**
private_extension_flag		**1**	**bslbf**
if (private_extension_flag == 1) {			
private_extension ()			
}			
}			
}			

10.4.22.3 Semantics

`message_id` – indicates the message ID. The length of this field is 16 bits.

`version` – indicates the version of the messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the ACR message. The value "0" shall not be used for this field.

`target_MMT_package_id` – provides identifier of target Package to be changed.

`MMT_package_id_length` – the length in bytes of the `target_MMT_package_id`.

`MMT_package_id_byte` – a byte in the `target_MMT_package_id`.

`FEC_Clean_Type` - When set to 0, it indicates the AL-FEC processing is needed when performing the Asset change with this message. Otherwise, it indicates the AL-FEC processing is not needed when performing the Asset change with this message. The valid value of `FEC_Clean_Type` is as listed in Table 72. A change point is defined as the point before or after a relevant Asset. A change-in (or splice-in) point is defined as the change point before the first relevant Asset and a change-out (or splice-out) point is defined as the change point after the last relevant Asset.

Table 72 — value of FEC_Clean_Type

Type	Description
0x00	AL_FEC source blocks may cross the change point boundaries. The AL-FEC decoding and re-encoding shall be performed when performing the Asset change with this message.
0x01	AL_FEC source blocks do not cross the change point boundaries of the target Asset. The AL-FEC decoding and re-encoding is not needed when performing the Asset change with this message.
0x02	AL_FEC source blocks do not cross the change point boundaries of both the target Asset and the designated Asset. The AL-FEC decoding and re-encoding is not needed when performing the Asset change with this message.
others	Reserved

Note `FEC_Clean_Type` equal to 1 may be used for the Asset change type of insertion. `FEC_Clean_Type` equal to 2 may be used for the Asset change type of replacement.

`number_of_assets` – indicates the number of target Assets to be changed.

`target_asset_id` – provides identifier of the Asset which to be changed.

`target_asset_type` – provides the type of the Asset to be changed.

`duration` – indicates the change time duration in milliseconds starting at the time which is specified in the `change_indicate_mode` field. Note, accurate duration may depend on the designated Assets structure.

`change_type` – indicates the type of media change on the target Asset which is indicated by `target_asset_id`. The target Asset shall be changed with designated Asset, in the way of which this field describes. The designated Asset is located in `designated_MMT_general_location_info`. Valid values for this field are described in Table 73.

Table 73 — value of change_type

Type	Description
0x00	Reserved
0x01	Replacement - Replace the target Asset with the designated Asset.
0x02	Insertion - Insert before the target Asset with the designated Asset.
0x03	Overlay - Overlay the designated Asset on the target Asset
0x04~0xFF	Reserved

For a splicing event, a splice point is namely a change point in which the media data after that shall be decoded correctly with a clean presentation quality. Generally, a splicing event is to concatenate on the MMT system level two different MMT Assets and the resulting MMT packages are conformant to the MMT standard.

In the case of 0x01, the splice-in point and the splice-out point usually are different. The first target Asset after the splice-in point and the Asset after the splice-out point shall be a random access point for splicing. In the case of 0x2, the splice-in point and the splice-out point usually are the same. The target Asset after the splice-in point shall be a random access point for splicing.

A sending entity shall ensure a random access property of the target Assets and for a splicing event, it shall correspond the parameter fields in this message correctly with the Assets relating to the splice points. A MANE shall perform the relevant Asset change in terms of the ACR message by locating the closest splice point for the splice event,

In the case of 0x03, a MANE shall align with the change-in point and change-out point for the designated Assets in the overlay processing. The spatial location of the designated Assets on the display may be provided by PI document such as MPEG Composition Information specified in part 11 of this standard, or Layout Configuration Table as defined in sub-clause 10.3.12. The MMT sending entity should ensure that the MMT receiving entities are capable to process the designated Asset additionally for the overlay processing. A receiving entity may ignore and discard the designated Assets if it is not capable to do so.

`change_indicate_mode` – defines how to indicate the time at which the media change happens. Valid values for this field is described in Table 74.

Table 74 — value of change_indicate_mode

Type	Description
0x00	Reserved
0x01	Change at the time which is based on UTC time.
0x02	Change at the time which is based on mpu_sequence_number
0x03	Change at the time which is based on MPU_timestamp_descriptor
0x04~0xFF	Reserved

`UTC_timestamp` – indicates a UTC time in NTP format corresponding to the time at which the media change happens. The field is the 32 bits MSB from the full resolution NTP timestamp.

`mpu_sequence_number` – indicates the sequence number of the first MPU to be changed in this target Asset indicated by `target_asset_id`. The media change starts from this MPU during the time which is indicated by the `duration`.

`mpu_timestamp_descriptor` – provides presentation time of the first AU of MPU to be changed in this target Asset indicated by `target_asset_id`. The media change starts from this presentation time during the time which is indicated by the `duration`.

`designated_MMT_general_location_info` – provides the location of designated Asset to be used in this media change. A MANE receiving the ACR message may decide to provide its own location information for the designated Asset when performing Asset change with this message.

`private_extension_flag` – indicates whether a private extension is present. If this flag is set to '1', the private extension is present.

`private_extension` – is a syntax element group serving as a container for proprietary or application-specific extensions.

10.4.23 CMAF Presentation Description (CPD) Messages

10.4.23.1 General

The CPD message conveys information about the Presentation to the MMT receiving entity. Annex L shall be used for mapping of CMAF content to MPU.

10.4.23.2 Syntax

The CPD Message syntax is shown in Table 75.

Table 75 — Syntax for CPD Message

Syntax	Values	No. of bits	Mnemonic
CPD_message ( ){			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
payload {			
presentation_count	**N1**	**8**	**uimsbf**
for (i=0; i<N1;i++) {			
selection_set_count	**N2**	**8**	**uimsbf**
for (j=0; j<N2; j++) {			
switching_set_count	**N3**	**8**	**uimsbf**
for (k=0 ; k<N3;k++) {			
track_count	**N4**	**8**	**uimsbf**
for (l=0;l<N4;l++) {			
packet_id		**16**	**uimsbf**
bandwidth_requirement		**32**	**uimsbf**
}			
mime_type()			
}			
}			

Table 75 *(continued)*

Syntax	Values	No. of bits	Mnemonic
} } }			

10.4.23.3 Semantics

`message_id` – indicates the identification of the Mapping message. The length of this field is 16 bits.

`version` – indicates the version of the Mapping message. The length of this field is 8 bits.

`length` – indicates the length of the Mapping message in bytes, counting from the beginning of the next field to the last byte of the Mapping message. The length of this field is 32 bits.

`presentation_count` – indicates the number of CMAF presentations that will be served as part of this session. The list of presentations may be updated dynamically in the case of live sessions. This field is 8 bits.

`selection_set_count` – indicates the number of selection sets in the current presentation. The field is 8 bits.

`switching_set_count` – indicates the number of switching sets in the current selection set. This field is 8 bits.

`track_count` – indicates the number of CMAF tracks/MPUs that are contained in the current switching set. This field is 8 bits long.

`packet_id` – provides the packet_id that will identify the MMTP sub-flow that carries the CMAF track/MPU. This field is 16 bits.

`bandwidth_requirement` – provides the bandwidth requirement of the current CMAF track/MPU. The unit of this field in bits per second and the field is 32 bits.

`mime_type` – the MIME type descriptor is used to describe the MIME type of the current switching set. If the encodings of the CMAF tracks within the switching set differ, the MIME type of the track with the highest codec requirements shall be signaled.

10.4.24 Content Selection (CS) Message

10.4.24.1 General

The CS message is a message that is sent from the MMT receiving entity to the MMT sending entity (or MANE) to indicate the current MMT receiving entity's selection of the CMAF tracks that it wishes to receive. Annex L shall be used for mapping of CMAF content to MPU.

10.4.24.2 Syntax

The syntax of CS Message is shown in Table 76.

Table 76 — Syntax for CS Message

Syntax	Values	No. of bits	Mnemonic
`Content_Selection_message ( ) {`			

Table 76 *(continued)*

Syntax	Values	No. of bits	Mnemonic
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
payload {			
start_time		**64**	**uimsbf**
subflow_count	**N1**	**8**	**uimsbf**
for (i=0; i<N1;i++) {			
packet_id		**16**	**uimsbf**
}			
}			
}			

10.4.24.3 Semantics

`message_id` - indicates the identification of the Content Selection message. The length of this field is 16 bits.

`version` - indicates the version of the Content Selection message. The length of this field is 8 bits.

`length` - indicates the length of the Content Selection message in bytes, counting from the beginning of the next field to the last byte of the Content Selection message. The length of this field is 32 bits.

`start_time` – the presentation start time at which the MMT receiving entity requests the the sending entity to start delivering.

`subflow_count` – indicates the number of MMTP sub-flows that the MMT receiving entity is requesting in this selection message. This field is 8 bit long.

`packet_id` – provides the packet_id that the MMTP receiving entity is requesting. This field is 16 bits.

10.4.25 RTSP message

10.4.25.1 General

The RTSP message describes a unified way to send setup and control messages on an MMT communication flow. This method allows multiplexing signaling and data on a single port for graceful NAT traversal.

For setup and control, RTSP messages can be wrapped into an MMT signaling message. To make the solution NAT friendly, when no “MMT receiving entity_port” in RTSP 1.0 or “dest_addr” in RTSP 2.0 fields are present, the MMT sending entity shall respond and send the content to the same port that sent the SETUP request.

A basic RTSP SETUP request for MMT content may be:

```
SETUP mmt://mmt.mpeg.org/movie.mp4
CSeq: 1000
```

A basic MMT setup response may be:

```
RTSP/2.0 200 OK
          CSeq: 1000
          Date: 14 Mar 2015 11:26:53 GMT+2
          Session: 161803398
```

```
Transport: MMT;unicast;
```

The SETUP response will be followed by the MMT content. Similarly, other RTSP control messages can be sent during a session, such as PAUSE, PLAY and TEARDOWN.

The following MMT message will wrap RTSP requests and responses:

10.4.25.2 Syntax

The syntax of RTSP Message is shown in Table 77.

Table 77 — RTSP message syntax

Syntax	Values	No. of bits	Mnemonic
RTSP_message() {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length	**N1**	**16**	**uimsbf**
message_payload {			
for (i=0; i<N1; i++) {			
RTSP_byte		**8**	**uimsbf**
}			
}			
}			

10.4.25.3 Semantic

`message_id` - indicates RTSP message ID. The length of this field is 16 bits.

`version` - indicates the version of RTSP messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` - indicates the length of RTSP messages. The length of this field is 16 bits. It indicates the length of the RTSP message counted in bytes starting from the next field to the last byte of the RTSP message. The value '0' shall not be used.

`RTSP_byte` - a byte in the RTSP message.

10.4.26 VAST/VMAP Message

10.4.26.1 General

An MMT sending entity uses the VAST_VMAP message to signal ad opportunities (formatted according to the VMAP standard[2]) and/or ad content (formatted according to the VAST standard[1]) to the MMT receiving entity. Upon receiving a VAST_VMAP message, the MMT receiving entity shall perform the following:

— If the content is a VMAP message, it shall extract and pass the message to the application before the first AdBreak starts.

— If the content is a VAST message:

  — It should extract the references to the Ad content, such as the MediaFile references for Linear ads

  — If the reference is to content delivered using the GFD mode, it shall cache it for later usage.

— If the reference is to content delivered using the MPU mode, it shall identify the MMTP sub-flow that matches the packet_id referenced by the URL in the MediaFile reference.

10.4.26.2 Syntax

The syntax of the VAST_VMAP message is defined in Table 78.

Table 78 — VAST_VMAP message syntax

Syntax	Value	No. of bits	Mnemonic
VAST_VMAP_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload {			
message_type		**8**	**uimsbf**
if (message_type == 0x1) {			
VMAP_version		**16**	**uimsbf**
start_time		**64**	**uimsbf**
}			
if (message_type == 0x02) {			
VAST_version		**16**	**uimsbf**
}			
message {			
message_length	**N1**	**16**	**uimsbf**
for (i=0; i<N1; i++) {			
message_char		**8**	**uimsbf**
}			
}			
reserved	**'1111 111'**	**7**	**bslbf**
private_extension_flag		**1**	**bslbf**
if (private_extension_flag == 1) {			
private_extension ()			
}			
}			
}			

10.4.26.3 Semantics

`message_id` – indicates the message ID. The length of this field is 16 bits.

`version` – indicates the version of the messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the ACR message. The value "0" shall not be used for this field.

`message_type` – an identifier of the message type. Two message types are defined: VMAP (0x1) and VAST (0x2).

VMAP_version – indicates the version number of the VMAP specification that is used by the embedded VMAP message. The version is encoded by multiplying the value by 100, e.g. version 4.0 will be signaled as 400.

start_time – provides the time of the first Ad Break in the embedded VMAP message. The VMAP message shall be passed to the application for processing prior to this start_time. The time is provided as UTC wallclock time in form of a NTP timestamp.

VAST_version – indicates the version number of the VAST specification that is used by the embedded VAST message. The version is encoded by multiplying the value by 100, e.g. version 4.0 will be signaled as 400.

message_length – provides the length of the embedded VAST or VMAP message. The message shall be formatted in UTF-8 format.

message_char – contains one UTF-8 character of the embedded VAST or VMAP message.

10.4.27 Service List (SL) information message

10.4.27.1 General

Service list information messages contains information on any updates of MMT service lists. When the MMT receiving entity receive this message, it can acknowledge the existence of new MMT service list or any update on previous one. If the MMT receiving entity wants to update any MMT service list, it can request it with any specific channel types.

10.4.27.2 Syntax

The syntax of SL information message is defined in Table 79

Table 79 — Syntax of service list information message

Syntax	Value	No. of bits	Mnemonic
SL_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
num_of_list	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_svc_list_URL_length	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
MMT_svc_list_URL_byte		**8**	**uimsbf**
}			
event_type		**4**	**uimsbf**
reserved	**'1111'**	**4**	**bslbf**
If (event_type=='0x02'){			
num_of_channel	**N3**	**8**	**uimsbf**
for (i=0; i<N3; i++) {			
channel_id		**8**	**uimsbf**
}			
} else If (event_type=='0x03'){			

Table 79 *(continued)*

Syntax	Value	No. of bits	Mnemonic
MMT_package_id {			
MMT_package_id_length	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
change_time		**32**	**uimsbf**
change_delivery_channel_type		**4**	**uimsbf**
reserved	**'1111'**	**4**	**bslbf**
}			
}			
}			
}			

10.4.27.3 Semantics

`message_id` – indicates the message ID. The length of this field is 16 bits.

`version` – indicates the version of the messages. MMT receiving entity may check whether the version of the received message is new or not. The length of this field is 8 bits.

`length` – indicates the length of the messages in Bytes, counting from the beginning of the next field to the last byte of this message. The value "0" shall not be used for this field.

`num_of_list` - This value is a number of MMT service list to be acknowledged to MMT receiving entity

`MMT_svc_list_URL_length` – the length in Bytes of the `MMT_svc_list_URL` string, excluding the terminating null character.

`MMT_svc_list_URL_byte` – this field is a unique identifier of the MMT service list descriptor.

`event_type` – indicates the type of event as defined in Table 80

Table 80 — Value of event_type

Value	Description
0x0	Corresponding MMT service list is newly available
0x1	Corresponding MMT servlce list is totally not valid. Therefore don't refer it anymore.
0x2	Corresponding MMT servlce list has updates on the designated channels.
0x3	Corresponding MMT Package will have a change in deliveryChannelType information when the time indicated at change_time. Therefore, it is recommended to fetch the new MMT Service List at those time and follow the new information if MMT receiving entities' capability allows.
0x4~0xF	reserved

`num_of_channel` - This value is a number of the channel that has updates

`channel_id` - This value is a unique identifier of the channel.

`change_time` – indicates the time when corresponding to the time when the deliveryChannelType of corresponding channel will change. This field is a UTC time in NTP format and 32 bits long.

change_delivery_channel_type - indicates the new channel type into which the corresponding package will be changed.

10.5 Descriptors

10.5.1 CRI descriptor

10.5.1.1 General

This subclause describes the descriptors and information related to signalling tables.

A CRI descriptor can be used to specify the relationship between the NTP timestamp and the MPEG-2 STC for synchronization purpose. The value of clock reference used by the Asset which is derived from the MPEG-2 TS is specified in the clock_relation_id field.

CRI_descriptors are carried in a CRI table.

10.5.1.2 Syntax

The syntax of the CRI_descriptor() is defined in Table 81.

Table 81 — CRI_descriptor() syntax

Syntax	Value	No. of bits	Mnemonic
CRI_descriptor() {			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
clock_relation_id		**8**	**uimsbf**
reserved	"11 1111"	**6**	**uimsbf**
STC_sample		**42**	**uimsbf**
NTP_timestamp_sample		**64**	**uimsbf**
}			

10.5.1.3 Semantics

descriptor_tag – a tag value indicating the type of a descriptor.

descriptor_length – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

clock_relation_id – the identifier of a clock relation.

STC_sample – contains the MPEG-2 STC value that corresponds to the following NTP_timestamp_sample. The sample value of STC is in 42-bit format.

NTP_timestamp_sample – the sample value of NTP timestamp that corresponds to the preceding STC_sample.

10.5.2 MPU timestamp descriptor

10.5.2.1 General

This descriptor provides presentation time of the first AU of MPU in presentation order after application of any offset such as the one provided by "elst" box. When presentation information (PI) is present, this descriptor shall be ignored.

10.5.2.2 Syntax

The syntax of the `MPU_timestamp_descriptor()` is defined in Table 82.

Table 82 — MPU timestamp descriptor

Syntax	Value	No. of bits	Mnemonic
`MPU_timestamp_descriptor () {`			
`  descriptor_tag`		16	uimsbf
`  descriptor_length`	N*12	8	uimsbf
`  for (i=0; i<N; i++) {`			
`    mpu_sequence_number`		32	uimsbf
`    mpu_presentation_time`		64	uimsbf
`  }`			
`}`			

10.5.2.3 Semantics

`descriptor_tag` – a tag value indicating the type of a descriptor.

`descriptor_length` – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`mpu_sequence_number` – indicates the sequence number of the MPU presented at a time given the following `mpu_presentation_time`.

`mpu_presentation_time` – indicates the presentation time of the first AU in the designated MPU by the 64-bit NTP time stamp format.

10.5.3 Dependency descriptor

10.5.3.1 General

For each dependent Asset, a dependency descriptor as specified below shall be included in the MPT, within the syntax element `dependency_descriptors` specified in Table 53.

10.5.3.2 Syntax

The syntax of the `dependency_descriptor()` is defined in Table 83.

Table 83 — Syntax of dependency descriptor for layered media

Syntax	Value	No. of bits	Mnemonic
`dependency_descriptor() {`			
`  descriptor_tag`		16	uimsbf
`  descriptor_length`		16	uimsbf
`  num_dependencies`	N1	8	uimsbf
`  for ( i = 0 ; i < N1 ; i++ ) {`			
`    asset_id()`			
`  }`			
`}`			

10.5.3.3 Semantics

`descriptor_tag` – a tag value indicating the type of this descriptor.

`descriptor_length` – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`num_dependencies` – indicates the number of complementary Assets the dependent Asset associated with this descriptor depends on.

`asset_id` – indicates the `asset_id` of another Asset on which the dependent Asset associated with this descriptor depends. The order of the id-s provided in this descriptor is such that the concatenation of MPUs as defined in 6.4 leads to a valid MPU and follows the media dependency hierarchy.

10.5.4 GFDT descriptor

10.5.4.1 General

A GFDT descriptor contains one or more CodePoints describing association with a specific object and object delivery properties. An MPT may contain a GFD descriptor for each Asset if the MPU(s) consisting the Asset is/are delivered through GFD mode.

10.5.4.2 Syntax

The syntax of GFD descriptor is defined in Table 84.

Table 84 — GFDT descriptor syntax

Syntax	Value	No. of bits	Mnemonic
GFD_descriptor () {			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**32**	**uimsbf**
number_of_CodePoints	**N1**	**8**	**uimsbf**
for(i=0; i<N1; i++) {			
value		**8**	**uimsbf**
fileDeliveryMode		**2**	**bslbf**
constantTransferLength_flag		**1**	**bslbf**
outOfOrderSending_flag		**1**	**bslbf**
FileTemplate_flag		**1**	**bslbf**
startTOI_flag		**1**	**bslbf**
endTOI_flag		**1**	**bslbf**
EntityHeader_flag		**1**	**bslbf**
maximumTransferLength		**48**	**uimsbf**
if(FileTemplate_flag == 1) {			
FileTemplate_length	**N2**	**8**	**uimsbf**
for(j = 0; j < N2; j++)			
FileTemplate_byte		**8**	**uimsbf**
} else {			
File_length	**N3**	**16**	**uimsbf**
for(j = 0; j < N3; j++)			

Table 84 *(continued)*

Syntax	Value	No. of bits	Mnemonic
File_byte		**8**	**uimsbf**
}			
if(startTOI_flag == 1)			
startTOI			**uimsbf**
if(endTOI_flag == 1)			
endTOI			**uimsbf**
if(EntityHeader_flag == 1) {			
EntityHeader_length	**N4**	**16**	**uimsbf**
for(j = 0; j < N4; j++)			
EntityHeader_byte		**8**	**uimsbf**
}			
}			
}			

10.5.4.3 Semantics

`descriptor_tag` – a tag value indicating the type of a descriptor.

`descriptor_length` – specifies the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`number_of_CodePoints` – specifies the number of CodePoints contained in this GFD descriptor. This field shall not be set to 0.

`value` – specifies the value of the CodePoint. The value shall be between 1 and 255. The value 0 is reserved.

`fileDeliveryMode` – specifies the file delivery mode according to Table 18. If this field is "1", the delivered object is a regular file and if it is "2", the delivered object is an entity consisting of an entity header and the file.

`constantTransferLength_flag` – specifies if all objects delivered by this CodePoint have constant transfer length. If this flag is set to "1", all objects shall have the transfer length as specified in the maximumTransferLength field. The default value is 0.

`outOfOrderSending_flag` – specifies if an outOfOrderSending attribute is included in this GFD descriptor. If this flag is set to "1", the outOfOrderSending attibute is included in this GFD descriptor. The default value is 0.

`FileTemplate_flag` – specifies if a file template for this CodePoint is included in this GFD descriptor. If this flag is set to "1", `FileTemplate_length` and `FileTemplate_byte` are included in this GFD descriptor.

`startTOI_flag` – specifies if a startTOI is included in this GFD descriptor. If this flag is set to "1", a startTOI is included in this GFD descriptor.

`endTOI_flag` – specifies if an endTOI is included in this GFD descriptor. If this flag is set to "1", an endTOI is included in this GFD descriptor.

`EntityHeader_flag` – specifies if an entity header for this CodePoint is included in this GFD descriptor. If this flag is set to "1", `EntityHeader_length` and `EntityHeader_byte` are included in this GFD descriptor.

`maximumTransferLength` – specifies the maximum transfer length in bytes of any object delivered with the CodePoint set in value field.

`FileTemplate_length` – specifies the length in byte of the file template.

`FileTemplate_byte` – specifies a byte in the file template (e.g. UTF-8, null terminated string).

`startTOI` – specifies the TOI of the first object that is delivered. The length of startTOI is varied according to the length of the TOI field in the MMTP payload for GFD mode with the same CodePoint value set in the value of this GFD descriptor.

`endTOI` – specifies the TOI of the last object that is delivered. The length of endTOI is varied according to the length of the TOI field in the MMTP payload for GFD mode with the same CodePoint value set in the value of this GFD descriptor.

`EntityHeader_length` – specifies the length in byte of the entity-header.

`EntityHeader_byte` – specifies a byte in the entity-header. An entity header specifies a full entity header in the format as defined in IETF RFC 2616 sub-clause 7.1. The entity-header applies for all objects that are delivered with the value of this CodePoint.

10.5.5 SI descriptor

10.5.5.1 General

The SI descriptor is an asset descriptor that can be used by the MMT sender to provide information about the content encryption applied to the asset as part of DRM protection or Conditional Access. For content protection using common encryption, the contents of this descriptor are extracted from the "`pssh`" box that is located as part of the MPU metadata.

10.5.5.2 Syntax

The syntax of the SI descriptor is defined in Table 85.

Table 85 — SI descriptor syntax

Syntax	Value	No. of bits	Mnemonic
SI_descriptor() {			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
security_system_count	**N1**	**8**	**uimsbf**
reserved	**"111 1111"**	**7**	**uimsbf**
system_provider_url_flag		**1**	**bslbf**
if (system_provider_url_flag) {			
system_provider_url_length	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
system_provider_url_byte		**8**	**uimsbf**
}			
}			
for (i=0;i<N1;i++) {			
system_id		**16*8**	**uimsbf**
kid_count	**N3**	**16**	**uimsbf**
for(j=0;j<N3;j++) {			
KID		**16*8**	**uimsbf**
}			
data_size	**N4**	**32**	**uimsbf**

Table 85 *(continued)*

Syntax	Value	No. of bits	Mnemonic

for(j=0;j<N4;j++) {
data
}
}
}
		8	**uimsbf**

10.5.5.3 Semantics

`descriptor_tag` – a tag value indicating the type of a descriptor.

`descriptor_length` – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`security_system_count` – provides the number of DRM or CAS system information that can process and handle the content protection, access control and rights management.

`system_provider_url_flag` – indicates whether a URL location of a provider for the system is provided or not. If this flag is set to "1", a system provider URL is provided. This URL can be used for downloading and installing the system, when needed.

`system_provider_url_length` – provides the length of a system provider URL.

`system_provider_url_byte` – specifies a byte in a system provider URL.

`system_id` – provides the UUID that uniquely identifies the DRM system.

`KID_count` – specifies the number of KID entries in the descriptor.

`KID` – identifies a key that the data field applies to.

`data_size` – specifies the size in bytes of the data field.

`data` – carries DRM system specific data.

10.5.6 MMT Service Descriptor

10.5.6.1 General

The MMTServiceList element provides information on MMT services that will be provided with MMT protocol from a specific service provider. A MMTServiceList element is comprised of multiple MMTChannel elements, which is a logical group of MMT packages. The MMT package corresponds to a specific program such as, news, sports game and a movie. The MMTPackage element has MMT_Package_id to distinguish from other MMT packages. When a specific MMT Package is selected, corresponding MMTP session will be initiated through its own session information from either of SDP file (SDPURI), in-lined SDP (SDP) or location of MMT PA message (MMT_PA_URL). In case the channel is made with linear contents, such as live contents with time duration, each MMTPackage element will have time information through startTime and endTime element.

The `descriptorURL` is the location of corresponding MMTServiceList. Multiple `descriptorURL` can be acquired and stored at the MMT receiving entity. For example, user can choose their preferred MMT Service Listprovider and input URL to their MMT receiving entity by themselves or it can be retrieved from service list (ex. EPG) of broadcast channel as another option to provide service list information. However, the way to acquire `descriptorURL` is out of scope of this specification. The MMT receiving entity shall send request to `descriptorURL`s to acquire the MMTServiceList. When the corresponding service list are received, MMT receiving entity should aggregate those multiple MMTServiceList and present to user to provide MMT service list that the user can select. `channelType` element provides type

of corresponding channel. It means which kind of transport physical channels are required to consume services described in this channel. In aggregating service list, MMT receiving entity should consider the channel type based on the current channel availability status. In case only broadband channel is available, MMT receiving entity should aggregate channel which has `channelType` . Session setup process shall follow process specified in Annex H.

10.5.6.2 Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
version="1.0">
<xs:element name="MMTServiceList" type="MMTServiceListType"/>

<xs:complexType name="MMTServiceListType">
 <xs:sequence>
 <xs:element name="MMTChannel" type="MMTChannelType"/>
 <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
 <xs:attribute name="version" type="xs:decimal" use="required"/>
 <xs:attribute name="descriptorProvider" type="xs:string"/>
 <xs:attribute name="updatePeriod" type="xs:nonNegativeInteger" minOccurs="0"/>
 <xs:anyAttribute namespace="##any"/>
</xs:complexType>

<xs:complexType name="MMTChannelType">
 <xs:sequence>
 <xs:choice maxOccurs="unbounded">
 <xs:element name="name" type="xs:string"/>
 <xs:element name="link" type="xs:anyURI"/>
  <xs:element name="channelDescription" type="xs:string"/>
  <xs:element name="releaseDate" type="dateTime" minOccurs="0"/>
<xs:element name="lastBuildDate" type="dateTime" minOccurs="0"/>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
</xs:choice>
<xs:element name="MMTPackage" type="MMTPackageType" maxOccurs="unbounded"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="access_network_descriptor" type="xs:decimal" use="required"/>
 <xs:attribute name="channelID" type="xs:decimal" use="required"/><xs:anyAttribute
namespace="##any"/>
</xs:complexType>

<xs:complexType name="MMTPackageType">
 <xs:sequence>
 <xs:choice maxOccurs="unbounded">
   <xs:element name="title" minOccurs="0"/>
   <xs:element name="packageDescription" minOccurs="0"/>
   <xs:element name="MMT_Package_id" type="xs:string"/>
   <xs:element name="startTime" type="dateTime" minOccurs="0"/>
   <xs:element name="endTime" type="dateTime" minOccurs="0"/>
   <xs:element name="SDPURI" type="xs:anyURI"/>
   <xs:element name="SDP" type="SDPType"/>
```

```
   <xs:element name="MMT_signaling_flow_URI" type="xs:anyURI "/>
   <xs:SimpleType name="SDPType">
      <!--- Note: the InlinedSDP below must be embedded in a CDATA section -->
      <restriction base="string"/>
   </xs:SimpleType>
   <xs:element name="PI_URI" type="xs:anyURI"/>
 <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:choice>
 </xs:sequence>
 <xs:anyAttribute namespace="##any"/>
</xs:complexType>
</xs:schema>
```

10.5.6.3 Semantics

`MMTServiceList` – The element provides information on MMT services that will be provided with MMTP from a specific service provider. The HTTP URL syntax is as follows: request_URI = service_URI "?" query_string,

where

— `service_URI` : the URI of the MMT service list server that provides MMT service list elements. This may be the same with descriptorURL, based on its implementation;

— `query_string` : descriptorURL *("&" deliveryChannelType) ("&" channel_range);

— `descriptorURL` : the URL of the MMT service list element for which the MMT receiving entity is requesting the service list element in the query part of the request;

— `deliveryChannelType` : the required channel type of the packages listed in MMT service list which the MMT receiving entity is requesting in the query part of the request;

— `channel_range` : a range of MMT service channel_id; channel_range = (channelA ["-" channelZ]) / (channelA [";"]); channelA = 1×DIGIT; the channel, or the first of a range of channels; channelZ = 1×DIGIT; the last channel of a range of channels.

`version` - A version of `MMTServiceList`

`descriptorProvider` - MMT service descriptor information provider.

`updatePeriod` – indicates the validity period of the updated descriptor information from the source in milliseconds.

`MMTChannel` - A logical entity which is a group of MMT packages

`name` - The name of the channel.

`link` - The HTML page URL about the corresponding channel.

`channelDescription` - Description on the corresponding channel

`releaseDate` - The release date for the channel information

`lastBuildDate` - The last time the program content of the channel changed.

`MMTPackage` – A specific program which corresponds to a MMT package

`access_network_descriptor` – The type of corresponding delivery channel. It means which kind of delivery channels are required to consume services described in this channel.

`channel_id` - This value is a unique identifier of the channel.

`title` - The name of the program with corresponding MMT Package

`packageDescription` - Description on the corresponding package

`MMT_Package_id` - This value is a unique identifier of the Package. This value has to be set to uniquely identify each package within a MMT Service List.

`startTime` - The start time of a corresponding MMT package if it is a scheduled content, e.g. linear contents. If it is a non-linear content, startTime information should be omitted.

`endTime` - The end time of a corresponding MMT package if it is a scheduled content, e.g. linear contents. If it is a non-linear content, endTime information should be omitted.

`SDPURI` - URI referring to a content referencing SDP file containing information that the MMT receiving entity needs in order to be able to receive and consume the content of the MMT Package. This content referencing SDP file is transported in a HTTP.

`SDP` – In-lined content referencing SDP file containing information that the MMT receiving entity needs in order to be able to receive and consume the content of the MMT Package encoded as UTF-8.

`MMT_Signaling_flow_URI` - URI referring to a flow of MMT signalling message. It can retrieve such as, MMT PA (Package Access) message or PA table for the corresponding MMT Package.

`PI_URI` - URI referring to a PI document containing information that the MMT receiving entity needs in order to be able to receive and consume the content of the MMT Package.

10.5.7 Mobile information descriptor

10.5.7.1 General

Mobile information descriptor contains information which is associated with MMT receiving entity in case it is a mobile device. When MMT receiving entity is a cellular mobile device, it can have mobile-specific attributes including MSISDN or Cell ID which are also defined in 3GPP.

10.5.7.2 Syntax

The syntax of mobile information descriptor is defined in Table 86

Table 86 — Syntax of mobile information descriptor

Syntax	Value	No. of bits	Mnemonic
mobile_info_descriptor() {			
descriptor_tag	**'0x0004'**	**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
MSISDN_flag		**1**	**boolean**
IMSI_flag		**1**	**boolean**
current_cell_id_flag		**1**	**boolean**
reserved	**'1 1111'**	**5**	**uimsbf**
if(MSISDN_flag == 1){			
MSISDN		**60**	**uimsbf**
reserved	**'1111'**	**4**	**uimsbf**
}			
if(IMSI_flag == 1){			
IMSI		**60**	**uimsbf**
reserved	**'1111'**	**4**	**uimsbf**

Table 86 *(continued)*

Syntax	Value	No. of bits	Mnemonic
}			
if(current_cell_id_flag == 1){			
current_cell_id		**60**	**uimsbf**
reserved	**'1111'**	**4**	**uimsbf**
}			
}			

10.5.7.3 Semantics

`descriptor_tag` – a tag value indicating the type of this descriptor. The value of this field is 0x0004 in Table 86.

`descriptor_length` – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`MSISDN_flag` – indicates whether MSISDN is included or not. If it is set to '1', the descriptor is included.

`IMSI_flag` – indicates whether IMSI is included or not. If it is set to '1', the descriptor is included.

`current_cell_id_flag` – indicates whether current_cell_ID is included or not. If it is set to '1', the descriptor is included.

`MSISDN` – indicates MSISDN (Mobile Subscriber International Subscriber Directory Number) number of MMT receiving entity, and it follows format as defined in ITU-T specification E.164. The length of this field is 15 decimal digits, which is coded into 60 binary digits where each decimal digit is assigned 4 binary bits.

`IMSI` – indicates IMSI (International Mobile Subscriber Identity) number of MMT receiving entity, and it follows format as defined in 3GPP TS 23.003. The length of this field is 15 decimal digits, , which is coded into 60 binary digits where each decimal digit is assigned 4 binary bits.

`current_cell_id` – indicates current cell ID (CGI for 2G/3G and eCGI for 4G) as defined in 3GPP TS 36.331. It is assigned a 15 decimal digit code which corresponds to totally 60 bits where 4bits are assigned for each 1 decimal digit.

10.5.8 Media quality descriptor

10.5.8.1 General

Media quality descriptor contains information on playback quality of media at MMT receiving entity side.

NOTE How a specific parameter is measured is out of the scope.

10.5.8.2 Syntax

The syntax of media quality descriptor is defined in Table 87

Table 87 — Syntax of media quality descriptor

Syntax	Value	No. of bits	Mnemonic
media_quality_descriptor() {			
descriptor_tag	**'0x0005'**	**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**

Table 87 *(continued)*

Syntax	Value	No. of bits	Mnemonic
num_time_sample	**N1**		
for (i=0;i<N1;i++){			
timestamp		**32**	**uimsbf**
video_quality{			
freeze		**8**	**uimsbf**
black		**8**	**uimsbf**
mb_error		**8**	**uimsbf**
}			
audio_quality{			
decoding_error		**8**	**uimsbf**
}			
}			
rebuffering_duration		**32**	**uimsbf**
}			

10.5.8.3 Semantics

descriptor_tag – a tag value indicating the type of this descriptor. The value of this field is 0x0005 in Table 87.

descriptor_length – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

num_time_sample – indicates the count of time instances (timestamp) at which the media quality is measured.

timestamp – indicates the time instance at which the media quality is measured based on UTC. The format is the "short-format" as defined in IETF RFC 5905, NTP version 4, Clause 6. The unit is a second.

freeze – indicates total count of consecutive playback video frames which is identical and measured from current timestamp to next timestamp. It counts all the frames which is consecutive and same even though its first head frame is different.

black – indicates total count of black playback video frames which is measured from current timestamp to next timestamp. It counts all the wholly black frames.

mb_error – indicates total count of video frames which has macro block error measured from current timestamp to next timestamp.

decoding_error – indicates total count of audio frame decoding error measured from current timestamp to next timestamp.

rebuffering_duration – indicates total duration of abnormal playback measured during the period specified in MC message in milliseconds. The length of this field is 32 bits.

10.5.9 MPU Presentation Region Descriptor

10.5.9.1 General

This descriptor provides information on the position of presenting the MPU by using the layout number and region number configured by the Layout Configuration Table. This is delivered in the descriptor area in the MP Table for each asset. Examples of layout configuration are provided in Annex K.

10.5.9.2 Syntax

Table 88 shows the syntax of the MPU Presentation Region Descriptor.

Table 88 — MPU Presentation Region Descriptor

Syntax	Values	No. of bits	Mnemonic
MPU_Presentation_Region_Descriptor (){			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**8**	**uimsbf**
for (i=0; i<N; i++) {			
mpu_sequence_number		**32**	**uimsbf**
layout_number		**8**	**uimsbf**
region_number		**8**	**uimsbf**
length_of_reserved	**M**	**8**	**uimsbf**
for (j=0; j<M; j++) {			
reserved_future_use		**8**	**bslbf**
}			
}			
}			

10.5.9.3 Semantic

`mpu_sequence_number` – This field specifies the MPU sequence number of which presentation region is specified in this descriptor.

`layout_number` – This field specifies the layout number in which the MPU is presented. The layout number '0' means the default layout.

`region_number` – This field specifies the region number in which the MPU is presented. The region number '0' means the default region.

`length_of_reserved` – This field specifies the length of the following reserved field in byte.

10.5.10 Asset Group Descriptor

10.5.10.1 General

This descriptor provides the group and priority within a group of assets. This descriptor enables grouping of multiple assets and switching assets within the group. This is delivered in the descriptor area in the MP Table for each asset.

10.5.10.2 Syntax

Table 89 shows the syntax of the Asset Group Descriptor.

Table 89 — Asset Group Descriptor

Syntax	Values	No. of bits	Mnemonic
Asset_Group_Descriptor (){			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**8**	**uimsbf**
group_identification		**8**	**uimsbf**
selection_level		**8**	**uimsbf**

Table 89 *(continued)*

Syntax	Values	No. of bits	Mnemonic
}			

10.5.10.3 Semantic

`group_identification` – This field specifies the group identification which identifies a group of audio/video and other assets.

`selection_level` – This field specifies the selection level within each group. The selection level '0' means the default asset to be presented. When the default asset cannot be chosen, the asset of which selection level is the smaller number is the candidate of alternative asset to be chosen and presented.

10.5.11 Access Network Descriptor

10.5.11.1 General

The delivery type descriptor is a descriptor to indicate specific delivery type for a service or interface that is being carried on it or interfaces supported by MMT receiving entity.

10.5.11.2 Syntax

Table 90 shows the syntax of the Asset Group Descriptor.

Table 90 — Access Network Descriptor

Syntax	Value	No. of bits	Mnemonic
`access_network_descriptor() {`			
descriptor_tag	'0x0004'	**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
access_network_type		**4**	**uimsbf**
reserved	'1111'	**4**	**uimsbf**
}			

10.5.11.3 Semantics

`descriptor_tag` - a tag value indicating the type of a descriptor.

`descriptor_length` – indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`access_network_type` – the type of corresponding delivery type shown in Table 91.

Table 91 — access_network_type

Access Network Type Value	Description
0x0	*reserved*
0x1	Channels which can receive only (e.g. Broadcast channel)
0x2	Channels which can receive and request (e.g. Broadband channel)
0x3	Channels having at least one Broadcast channel and one Broadband channel respectively at the same time.
0x4 ~ 0xFF	*reserved for future use*

10.5.12 Subtitle Change descriptor

10.5.12.1 General

When an MMT Package is composed of multiple subtitle Assets, the subtitle Assets can be selectively delivered to MMT receiving entity. Each subtitle can be generated in various languages and the time information (such as begin time and end time of a certain subtitle data) of the subtitles for the same period of media can be different because of the lingual characteristics. The selected subtitle Asset can be changed to another subtitle Asset during its delivery and its presentation. If the subtitle Asset changed before it is completely presented and another subtitle Asset is already passed, the passed Assets can be presented while waiting for the next subtitle data. The Subtitle Change descriptor provides MPU presentation time and durations of the passed MPUs of both subtitle Assets. MMT sending entity shall send the Subtitle Change descriptor to MMT receiving entity, so the MMT receiving entity can decide which subtitle Asset to replay and the duration of replay before the presentation time of next MPU of the new subtitle Asset.

10.5.12.2 Subtitle change configuration

When a multimedia content contains more than one subtitle stream of different languages, the length of subtitle data is different even for the same content because of their lingual characteristics. In this case the presentation duration can be also different. For example, Korean subtitle has 30 letters while English subtitle has 20 letters in a certain scene, and the start time (or begin time) and also end time of the subtitle data is different.

While a multimedia data which has more than one subtitle stream is playing, if the subtitle is changed for some reason (e.g. by user's decision), because of the difference of length of subtitles, the subtitle data cannot be played appropriately.

If subtitle is changing before it is completely played in a scene, and also if the other subtitle unfortunately already finished, the user cannot watch any of them completely and has no choice except waiting the next paragraph.

It can be one of the factors that cause degradation of the user experience for the users expecting multi-lingual subtitle multimedia content.

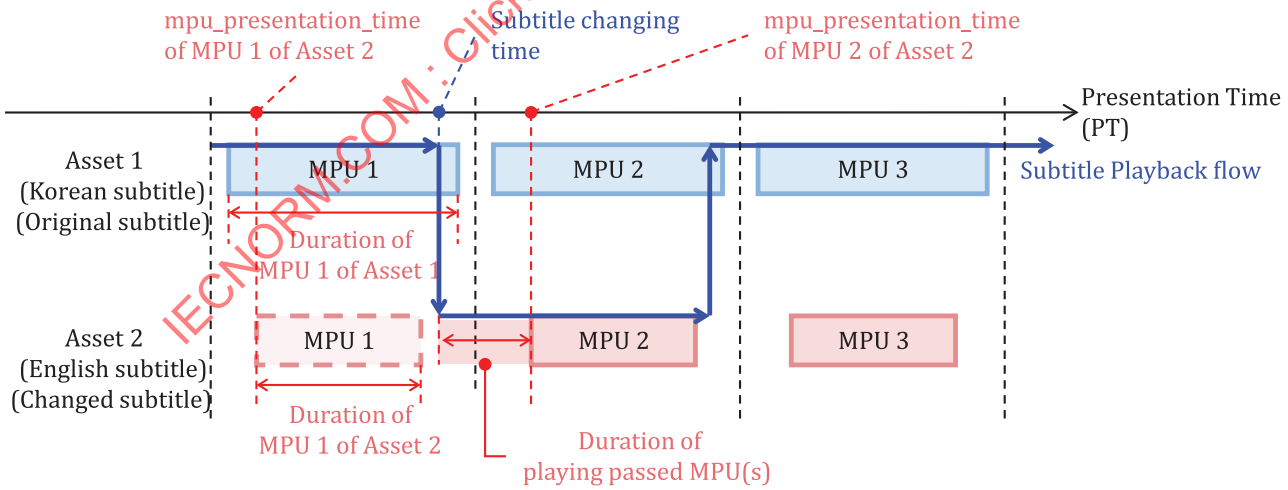


Figure 21 — Subtitle change during MMT delivery

Figure 21 depicts an example of subtitle change process in presentation timeline. The original subtitle Asset is Asset 1 (Korean subtitle) and the changed subtitle Asset is Asset 2 (English subtitle). Subtitle is changing from Asset 1 to Asset 2 while Asset 1 is being served. Each MPU can contain a single subtitle paragraph for a certain video scene.

As shown in Figure 21, if subtitle is changed before the original subtitle (MPU 1 of Asset 1) is finished, and the changed subtitle (MPU 1 of Asset 2) is already passed in the same scene, none of them cannot be presented completely. Any subtitle data won't be displayed and will wait until the presentation time of next subtitle MPU (MPU 2 of Asset 2).

In this case, by presenting the past MPU(s) while waiting for the presentation of next MPU, user can keep the context of subtitle. Utilizing the space of MPU 1 and MPU 2 of Asset 2, MPU 1 of Asset 1 or MPU 1 of Asset, or both MPU 1 of Asset 1 and Asset 2 are presented.

For this operation, MMT sending entity should send subtitle change descriptor so that MMT receiving entity can decide the presentation duration of passed MPU(s), and which MPU(s) to be presented. The descriptor should mainly provide the following values below:

1. presentation start time of MPU 1 of Asset 2
2. presentation start time of MPU 2 of Asset 2
3. presentation duration of MPU 1 of Asset 1
4. presentation duration of MPU 1 of Asset 2

The detailed explanation with syntax and semantics of subtitle change descriptor is provided in the following sub-clauses.

10.5.12.3 Syntax

The syntax of subtitle change descriptor is defined in Table 92.

Table 92 — Syntax for Subtitle Change descriptor

Syntax	Value	No. of bits	Mnemonic
Subtitle_Change_descriptor () {			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
MPUPT_new_passed		**64**	**uimsbf**
MPUPT_new_next		**64**	**uimsbf**
duration_original_passed		**32**	**uimsbf**
duration_new_passed		**32**	**uimsbf**
}			

10.5.12.4 Semantics

`descriptor_tag` - a tag value indicating the type of a descriptor.

`descriptor_length` - indicates the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`MPUPT_new_passed` - indicates the mpu_presentation_time of the passed MPU in changed subtitle Asset.

`MPUPT_new_next` - indicates the mpu_presentation_time of the next MPU in changed subtitle Asset.

`duration_original_passed` - indicates the presentation time duration of the passed MPU in original subtitle Asset. The duration_original_passed is expressed in milliseconds.

`duration_new_passed` - indicates the presentation time duration of the passed MPU in changed subtitle Asset. The duration_new_passed is expressed in milliseconds.

10.5.12.5 Subtitle Change operation

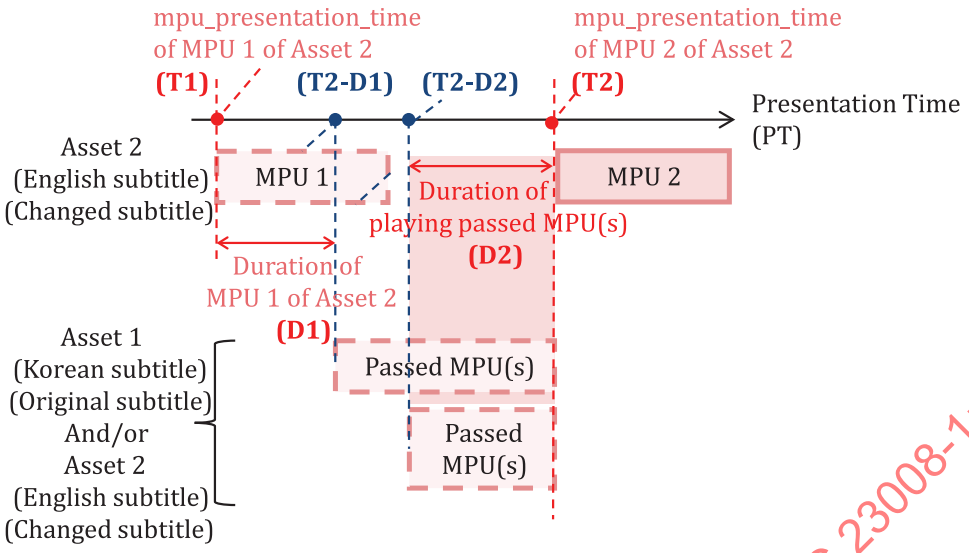


Figure 22 — Subtitle change configuration

MMT receiving entity can change metadata of subtitle MPUs using the Subtitle Change descriptor and transfer appropriate media data and metadata to its render as depicted in Figure 22.

The configuration operates as below:

1. Receive Subtitle Change descriptor
   A. presentation start time of MPU 1 of Asset 2 (MPUPT_new_passed)
   B. presentation start time of MPU 2 of Asset 2 (MPUPT_new_next)
   C. presentation duration of MPU 1 of Asset 1 (duration_original_passed)
   D. presentation duration of MPU 1 of Asset 2 (duration_new_passed)
2. Calculate the gap period of MPU 1 and MPU 2 of Asset 2 using A, B, and D.
3. Decide which past MPU(s) to present until the presentation of next MPU and the duration of presenting the past MPU(s) by considering the period, MPU metadata of passed MPU of both Asset 1 and Asset 2, capability of decoder and render.
4. Decide replay mode of passed MPU(s)
   A. Normal replay mode: replay only the time of duration decided in 3. calculated back from the present end time of the passed MPU(s).
   B. Trick replay mode: replay total duration of the passed MPU(s) in the duration decided in 3. (The replay speed can be the duration of passed MPU ÷ the duarion of replay)
   C. Private mode: replay in any other way that MMT receiving entity decides.
5. Adjust the mpu_presentation_time of passed MPU(s) as decided replay mode in 4.
   A. Normal replay mode: Calculate back from MPUPT_new_next for the time of duration of passed MPU(s).

B. Trick replay mode: Calculate back from MPUPT_new_next for the time of duration decided in 3.

6. For trick replay mode, the duration of media sample of the MPU(s) should be adjusted. The sample duration is defined in moov box in MPU metadata.

7. Deliver the information decided in 2~6. and related media data to the renderer.

When the duration of presenting the past MPU(s) is determined (Step 3. above), the ratio of gap period of MPU 1 and MPU 2 of Asset 2 and the duration of presenting the past MPU(s) itself can be considered.

The operation described in this clause can be done by MMT sending entity. In that case, MMT receiving entity will just do the typical MPU data reproduction and media presentation.

10.5.13 AT descriptor

10.5.13.1 General

`AT_descriptor()` contains information about time period during which asset can be available from the server side. A MPT may contain an `AT_descriptor()` for each Asset when available time is provided by one of the locations of this asset.

10.5.13.2 Syntax

The syntax of AT descriptor is defined in Table 93.

Table 93 — AT descriptor syntax

Syntax	Value	No. of bits	Mnemonic
`AT_descriptor() {`			
`  descriptor_tag`		16	uimsbf
`  descriptor_length`		32	uimsbf
`  available_time_count`	N1	8	uimsbf
`  for (i=0; i<N1; i++) {`			uimsbf
`    location_index`		8	uimsbf
`    available_begin`		32	uimsbf
`    available_end`		32	uimsbf
`  }`			
`}`			

10.5.13.3 Semantics

`descriptor_tag` – a tag value indicating the type of a descriptor.

`descriptor_length` – specifies the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`available_time_count` – indicates the number of locations that provide available time information. 'N1' is the value of this count.

`location_index` – indicates the index of the available time information inside the descriptor corresponding to the location information in `MMT_general_location_info()`.

`available_begin` – describe the earliest time in UTC when the content can be available. The unit of this field is second.

`available_end` – describe the latest time in UTC when the content can be available. The unit of this field is second. Based on the value of those two attributes, there are different situations to obtain the range of accessible time window, as it is shown in Table 94.

Table 94 — The value of available time attributes and correspoding situations

'available_begin' value	'available_end' value	Description
all '0'	all '1'	The content is always available
UTC	all '1'	The content is always available from 'available_begin' time
UTC1	UTC2	The content is available between 'available_begin' and 'available_end'

10.5.13.4 Procedure

— If the available time of the content is not ready on the server side, the `AT_descriptor()` would not be included in MPT asset_descriptors.

— If the content is prepared with available time information. The descriptor would be delivered along with the MPT.

  — For the new client, it will read the `AT_descriptor()` and get the value of attributes 'location_type', 'available_begin' and 'available_end'. Then based on the location_index, the client may get the location information in `MMT_general_location_info()` in accordance with available time information. Thus the client is able to get the asset content during available time at the corresponding location.

  — For the legacy client, once they receive the `AT_descriptor()` with descriptor_tag=0x8000, it will ignore this descriptor as the tag is defined in the reserved fields. Since there is no available time provided, the client may not get the data if it tries to request before the content is available.

10.6 Syntax element groups

10.6.1 MMT_general_location_info

10.6.1.1 General

An `MMT_general_location_info` syntax element group is used to provide location information for the payload of the described item.

10.6.1.2 Syntax

The syntax of the `MMT_general_location_info` is defined in Table Table 95.

Table 95 — MMT_general_location_info syntax

Syntax	Value	No. of bits	Mnemonic
`MMT_general_location_info() {`			
`  location_type`		8	uimsbf
`  if (location_type == 0x00) {`			
`  packet_ id`		16	uimsbf
`  } else if (location_type == 0x01) {`			
`  ipv4_src_addr`		32	uimsbf
`  ipv4_dst_addr`		32	uimsbf
`  dst_port`		16	uimsbf
`  packet_ id`		16	uimsbf

Table 95 *(continued)*

Syntax	Value	No. of bits	Mnemonic
} else if (location_type == 0x02) {			
ipv6_src_addr		**128**	**uimsbf**
ipv6_dst_addr		**128**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_ id		**16**	**uimsbf**
} else if (location_type == 0x03) {			
network_id		**16**	**uimsbf**
MPEG_2_transport_stream_id		**16**	**uimsbf**
reserved	"111"	**3**	**bslbf**
MPEG_2_PID		**13**	**uimsbf**
} else if (location_type == 0x04) {			
ipv6_src_addr		**128**	**uimbsf**
ipv6_dst_addr		**128**	**uimbsf**
dst_port		**16**	**uimbsf**
reserved	"111"	**3**	**bslbf**
MPEG_2_PID		**13**	**uimbsf**
} else if (location_type == '0x05') {			
URL_length	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
URL_byte		**8**	**char**
}			
} else if (location_type == '0x06') {			
length	N2	**16**	**uimsbf**
for (i=0;i<N2;i++) {			
byte		**8**	**uimsbf**
}			
} else if (location_type == '0x07') {			
}			
} else if (location_type == '0x08') {			
message_id		**16**	**uimsbf**
} else if (location_type == '0x09') {			
packet_id		**16**	**uimsbf**
message_id		**16**	**uimsbf**
} else if (location_type == '0x0A') {			
ipv4_src_addr		**32**	**uimsbf**
ipv4_dst_addr		**32**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_id		**16**	**uimsbf**
message_id		**16**	**uimsbf**
} else if (location_type == '0x0B') {			
ipv6_src_addr		**128**	**uimsbf**
ipv6_dst_addr		**128**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_id		**16**	**uimsbf**

Table 95 *(continued)*

Syntax	Value	No. of bits	Mnemonic
message_id		**16**	**uimsbf**
} else if(location_type == '0x0C') {			
ipv4_src_addr		**32**	**uimbsf**
ipv4_dst_addr		**32**	**uimbsf**
dst_port		**16**	**uimbsf**
reserved	"111"	**3**	**bslbf**
MPEG_2_PID		**13**	**uimbsf**
}			
}			

10.6.1.3 Semantics

`location_type` – this field indicates the type of the location information as defined in Table 96.

Table 96 — Value of location_type

Value	Description
0x00	An Asset in the same MMTP packet flow as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x01	MMTP packet flow over UDP/IP (version 4)
0x02	MMTP packet flow over UDP/IP (version 6)
0x03	An elementary stream within an MPEG-2 TS in a broadcast network.
0x04	An elementary stream (ES) in an MPEG-2 TS over the IP broadcast network
0x05	URL
0x06	Reserved for private location information
0x07	The same signalling message as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x08	A signalling message delivered in the same data path as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x09	A signalling message delivered in a data path in the same UDP/IP flow as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x0A	A signalling message delivered in a data path in a UDP/IP (version 4) flow
0x0B	A signalling message delivered in a data path in a UDP/IP (version 6) flow
0x0C	An elementary stream (ES) in an MPEG-2 TS over the IP v4 broadcast network
0x0D~0x9F	Reserved for ISO use
0xA0~0xFF	Reserved for private use

`packet_id` – `packet_id` in the MMTP packet header.

`ipv4_src_addr` – IP version 4 source address of an IP application data flow.

`ipv4_dst_addr` – IP version 4 destination address of an IP application data flow.

`dst_port` – destination port number of an IP application data flow.

`ipv6_src_addr` – IP version 6 source address of an IP application data flow.

`ipv6_dst_addr` – IP version 6 destination address of an IP application data flow.

`network_id` – broadcast network identifier that carries the MPEG-2 TS. This field is specific to the broadcast system in use.

`MPEG_2_transport_stream_id` – MPEG-2 TS identifier.

`MPEG_2_PID` – PID of the MPEG-2 TS packet carrying the ES.

`URL_length` – length in bytes of a URL. The terminating null ("0x00") shall not be counted.

`URL_byte` – a byte data in a URL. The terminating null ("0x00") shall not be included.

`byte_offset` – a byte offset from the first byte of a file.

`length` – the length of the byte range in bytes.

`message_id` – MMT signalling message identifier (see Table 62).

10.6.2 asset_id

10.6.2.1 General

An `asset_id` syntax element group is used to provide an Asset identifier. If the "`mmpu`" box is present, the values of this syntax element group shall be identical to the Asset identifier of that box. If not present, the assignment of Asset identification is outside the scope of this document.

10.6.2.2 Syntax

The syntax of the `asset_id` is defined in Table 97.

Table 97 — asset_id syntax

Syntax	Value	No. of bits	Mnemonic
`asset_id(){`			
`  asset_id_scheme`		32	uimsbf
`  asset_id_length`	N1	32	uimsbf
`  for (j=0; j<N1; j++) {`			
`  asset_id_byte`		8	uimsbf
`  }`			
`}`			

10.6.2.3 Semantics

`asset_id_scheme` – provides the Asset ID scheme as defined in Table 7.

`asset_id_length` – provides the length in bytes of the `asset_id`.

`asset_id_byte` – specifies a byte in the `asset_id`.

10.6.3 Identifier mapping

10.6.3.1 General

This syntax group element provides a mapping of the content identifier and an MMTP packet sub-flow. The MMTP packet sub-flow is the subset of the packets of an MMTP packet flow that share the same `packet_id`. The content identifier may be provided in different forms, such as an asset identifier, a URL or a pattern.

10.6.3.2 Syntax

The syntax of the `Identifier_mapping()` is defined in Table 98.

Table 98 — Identifier_mapping

Syntax	Value	No. of bits	Mnemonic
Identifier_mapping () {			
identifier_type		**8**	**uimsbf**
if (identifier_type == 0x00) {			
asset_id()			
} else if (identifier_type == 0x01) {			
URL_count	N1	16	**uimsbf**
for(i=0;i<N1;i++) {			
URL_length[i]	N2	16	**uimsbf**
for(j=0;j<N2;j++){			
URL_byte[i]		8	**uimsbf**
}			
}			
} else if (identifier_type == 0x02) {			
regex_length	N3	16	**uimsbf**
for(i=0;i<N3;i++){			
regex_byte		8	**uimsbf**
}			
} else if (identifier_type == 0x03) {			
representation_id_length	N4	16	**uimsbf**
for (i=0;i<N4;++) {			
representation_id_byte		8	**uimsbf**
}			
} else {			
private_length	N5	16	**uimsbf**
for(i=0;i<N5;i++){			
private_byte		8	**uimsbf**
}			
}			
}			

10.6.3.3 Semantics

`identifier_type` – provides the type of the identifier that is used for the mapping to the `packet_id`. The list of available identifier types is defined in Table 99.

Table 99 — Identifier types for the identifier mapping

Value	Description
0x00	The identifier of the content is provided as an Asset id as defined in 10.6.2.
0x01	The identifier of the content is provided as a list of URLs that are related together and share the same `packet_id` mapping. An example are DASH segments of the same representation.
0x02	The identifier of the content is provided as RegEx string that is used to match one or more URLs that identify files with the same `packet_id` mapping.
0x03	The identifier of the content is provided as a DASH Representation@id as defined in ISO/IEC 23009-1 to identify all DASH segments of the same representation.
0x04 ~ 0xFF	This value range is reserved for private identifiers.

`URL_count` – the URL is a list of URLs and this value indicates the number of URLs provided in this list.

`URL_length` – the length in bytes of the following URL.

`URL_byte` – a byte of the URL that is formatted as an UTF-8 character string.

`regex_length` – the identifier is provided as a regular expression that matches a set of URLs and this field indicates the length of the RegEx string.

`regex_byte` – a byte of the RegEx string that is provided as a UTF-8 string.

`representation_id_length` – the identifier is provided as a DASH Representation@id and this value indicates the length of the Representation@id string.

`representation_id_byte` – a byte of the Representation@id string.

`private_length` – the identifier is provided as private data and this field provides the length of the private identifier in bytes.

`private_byte` – a byte of the private identifier.

10.6.4 MIME type

10.6.4.1 General

This syntax element provides a MIME type as a string.

10.6.4.2 Syntax

The syntax of the `mime_type()` element is defined in Table 100.

Table 100 — mime_type ()

Syntax	Value	No. of bits	Mnemonic
`mime_type () {`			
`  mime_type_length`	N1	8	uimsbf
`  for(i=0;i<N1;i++) {`			
`    mime_type_byte`		8	uimsbf
`  }`			
`}`			

10.6.4.3 Semantics

`mime_type_length` – the length of the MIME type string.

`mime_type_byte` – a byte of the MIME type string, which is provided as a UTF-8 string.

10.7 ID and tags values

The values of the message identifier (`message_id`) are defined in Table 101.

Table 101 — Message identifier (message_id) values

Value	Description
0x0000	PA message

Table 101 *(continued)*

Value	Description
0x0001 ~ 0x0010	MPI messages For a Package, 16 contiguous values are allocated to MPI messages. If the value equals 16 (0x0010), the MPI message carries complete PI. If the value equals N where N = 1 ~ 15, the MPI message carries Subset-(N-1) PI.
0x0011 ~ 0x0020	MPT messages For a package, 16 contiguous values are allocated to MPT messages. If the value equals 32 (0x0020), the MPT message carries complete MPT. If the value equals N where N = 1 ~ 15, the MPT message carries Subset-(N-1) MPT.
0x0021 ~ 0x01FF	Reserved
0x0200	CRI message
0x0201	DCI message
0x0202	SSWR message
0x0203	AL_FEC message
0x0204	HRBM message
0x0205	MC message
0x0206	AC message
0x0207	AF message
0x0208	RQF message
0x0209	ADC message
0x020A	HRBM Removal message
0x020B	LS message
0x020C	LR message
0x020D	NAMF message
0x020E	LDC message
0x020F	NK message
0x0210	MRI message
0x0211	CR message
0x0212	DRI message
0x0213	DSI message
0x0214	BQP message
0x0215	BPR message
0x0216	PRR message
0x0217	PSF message
0x0218	CCI message
0x0219	MTR message
0x021A	MTN message
0x021B	ACR message
0x021C	CPD message
0x021D	CS message
0x021E	RTSP message
0x021F	VSAT/VMAP message
0x0220	SL message
0x0221 ~0x6FFF	Reserved for ISO use (16-bit length message)

Table 101 *(continued)*

Value	Description
0x7000 ~0x7FFF	Reserved for ISO use (32-bit length message)
0x8000 ~ 0xFFFF	Reserved for private use

The values of the table identifier (`table_id`) are defined in Table 102.

Table 102 — Table identifier (table_id) values

Value	Description
0x00	PA table
0x01	Main PI (Subset 0 MPI table)
0x02 ~ 0x0F	Subset 1 MPI table ~ Subset 14 MPI table
0x10	Complete MPI table
0x11 ~ 0x1F	Subset 0 MP table (`SUBSET_0_MPT_TABLE_ID`) ~ Subset 14 MP table
0x20	Complete MP table
0x21	CRI table
0x22	DCI table
0x23	SI table
0x24	LC table
0x25 ~0x7F	Reserved for ISO use (16-bit length table)
0x80 ~ 0xFF	Reserved for private use

The values of descriptor tag are defined in Table 103.

Table 103 — Descriptor tag values

Value	Description
0x0000	CRI descriptor
0x0001	MPU timestamp descriptor
0x0002	Dependency descriptor
0x0003	GFDT descriptor
0x0004	SI descriptor
0x0005	MMT service descriptor
0x0006	Mobile information descriptor
0x0007	Media quality descriptor
0x0008	MPU presentation region descriptor
0x0009	Asset group descriptor
0x000A	Access network descriptor
0x000B	Subtitle change descriptor
0x000C	AT descriptor
0x000D ~ 0x3FFF	Reserved for ISO use (8-bit length descriptor)
0x4000 ~ 0x6FFF	Reserved for ISO use (16-bit length descriptor)
0x7000 ~ 0x7FFF	Reserved for ISO use (32-bit length descriptor)
0x8000 ~0xFFFF	Reserved for private use

11 Hypothetical receiver buffer model (HRBM)

11.1 General

The HRBM is used to ensure an effective MMT operation under a fixed end-to-end delay and limited memory requirements for buffering of incoming MMTP packets. The hypothetical receiver buffer model shall be used by the MMT sending entity to emulate the behaviour of the receiving entity. It is described in the following subclauses in more detail.

An MMT sending entity shall run the hypothetical receiver buffer model to ensure that any processing it performs on the packet stream is within the reception constraints in the MMT receiving entity. The sending entity shall determine the required buffering delay and the required buffer size and shall signal this information to the receiving entities.

At the receiving entity, several buffers are involved in the reconstruction of an MPU from the MMTP packets received. Some delivery operations, such as FEC coding, interleaving and retransmission, introduce delay and jitter that requires buffering at the receiving entity. The hypothetical receiver buffer model defines operations of the buffers at the receiving entity to ensure that at any time the buffer occupancy is within the buffer size requirement. The buffers, whose operation is defined, are depicted in Figure 23 and are described in detail in 11.2, 11.3, and 11.4. The MMT HRBM is applied per each MMTP sub-flow, i.e. for all MMTP packets of an MMTP flow that share the same `packet_id`. The actual set of buffers is determined based on the configuration of the delivery session for that particular MMTP sub-flow.

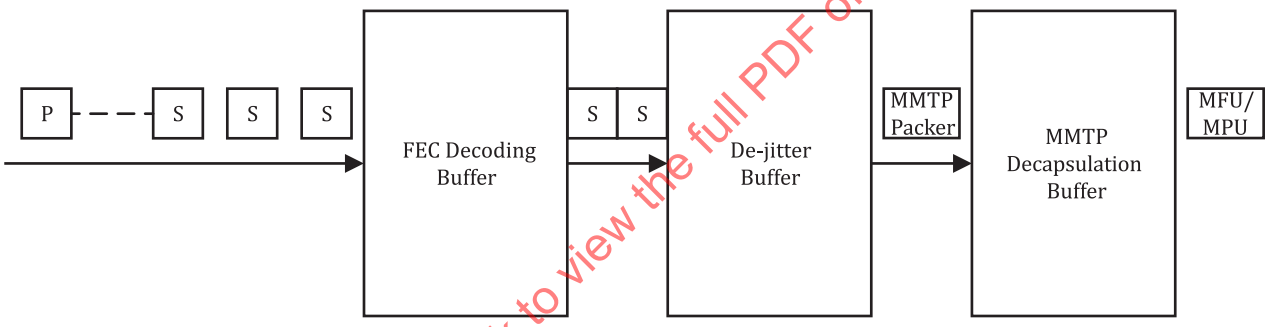


Figure 23 — MMT protocol hypothetical receiver model

11.2 FEC decoding buffer

The FEC decoding buffer is optional and is only required when FEC protection is applied to the MMTP packet sub-flow (e.g. for packets of a particular asset). FEC decoding is typical for many applications, where lower layer transmission may not be sufficient to recover messages from channel errors or when network congestion may cause packet drops or excessive delays. To perform FEC decoding, a buffer is required where incoming packets are stored until sufficient source and repair symbol data are available to perform the FEC decoding.

In this hypothetical receiver buffer model, the FEC decoding buffer is optional and shall operate as follows.

— The FEC decoding buffer is initially empty.

— Incoming packet `i` with transmission timestamp `ts` arrives to the FEC decoding buffer. If `buffer_occupancy + packet_size < max_buffer_size`, accept the packet or otherwise mark the packet as being non-compliant .

— If FEC is applied to packet `i`,

  — determine the source block `j` to which packet `i` belongs;

— determine the insertion time `ts` of the first packet of source block `j`;

— at time `ts+protection_window_time`, move all packets of source block `j` to the de-jitter buffer;

— discard repair packets.

The `protection_window_time` is the required buffering time that must elapse since the reception of the first packet of a source block, until FEC decoding is attempted. This time is typically calculated based on the FEC block size.

11.3 De-jitter buffer

The de-jitter buffer ensures that MMTP packets experience a fixed transmission delay from source to the output of the MMT protocol stack, assuming a maximum transmission delay. Data units that experience a transmission delay larger than the maximum transmission delay might be discarded by the MMT receiving entity.

The de-jitter buffer shall operate as follows.

— The de-jitter buffer is initially empty.

— Accept the MMTP packet as it arrives.

— Remove the MMTP packet at time `ts+Δ`, where `ts` is the timestamp of the MMTP packet and Δ is the signaled fixed end-to-end delay.

After the de-jittering is applied, all MMTP packets that arrived correctly or were recovered through FEC/retransmissions will have experienced the same end-to-end delay.

NOTE Annex A shows jitter calculation in MMTP.

11.4 MMTP packet decapsulation buffer

The MMTP packet decapsulation buffer is used to perform MMTP packet processing before delivering the payload to the upper layers. The output of the MMTP packet processing may either be the MFU payload (in low-delay operation), a movie fragment or a complete MPU. The decapsulation (removal of the MMTP packet and payload headers) and any required de-fragmentation/de-aggregation of the packets are then performed as part of the MMTP processing. This procedure may require some buffering delay, denoted as decapsulation delay, to reconstruct the data to be passed to the application layer. However, the decapsulation delay is not considered as part of the transmission delay and the availability of the data for consumption by the upper layers will be guaranteed by the entity delivering the MPU, regardless of the decapsulation delay.

The MMTP packet decapsulation buffer shall work as follows.

— The MMTP packet decapsulation buffer is initially empty.

— The MMTP packet is inserted into the MMTP packet decapsulation buffer immediately after the de-jittering is performed.

— For MMTP packets carrying aggregated payload, remove the packet and payload header and extract each single data unit.

— For MMTP packets carrying fragmented payload, the packet is kept in the buffer until all corresponding fragments are received correctly or until a packet is received that does not belong to the same fragmented data unit.

— Depending on the operation mode of the client, if a complete MPU, a movie fragment or a single MFU is recovered, forward the reconstructed data to the upper layer.

11.5 Usage of HRBM

Based on the hypothetical receiver buffer model, an MMT sending entity is able to determine the transmission schedule, the buffer size and the buffering delay Δ, so that no packets are dropped due to buffer overflow, assuming a maximum delivery delay in the target path. The MMT sending entity shall guarantee that packets that experience a transmission delay below a set threshold will be delivered to the upper layer after a constant delay and without causing the MMT receiving entity buffer to underflow or overflow.

11.6 Estimation of end-to-end delay and buffer requirement

An MMT sending entity shall estimate the maximum expected and tolerable transmission delay in the transmission path down to an MMT receiving entity. If FEC is in use, the MMT sending entity shall add the FEC buffering delay that covers for the time needed to assemble a source block since FEC decoding is required to recover lost MMTP packets. Finally, any delays that might be incurred by fragmentation of packets (and other operations) shall be added. The resulting estimation of the MMTP packet delivery delay shall then be signalled to the MMT receiving entities as the fixed end-to-end transmission delay.

fixed end-to-end = *maximum transmission delay* + *FEC buffering time*

In order to estimate the resulting buffer requirement, the MMT sending entity shall use the fixed end-to-end delay and subtract the minimum transmission delay for the path down to the MMT receiving entity and then estimate the buffer size requirement as the product of the maximum bitrate of the MMTP packet stream and the calculated buffered data duration.

buffer size = (*maximum delay* – *minimum delay*) × *maximum bitrate*

11.7 HRBM signalling

After determining the required buffer size and the fixed end-to-end delay for the system, an MMT sending entity shall communicate this information to the MMT receiving entity. This is done using the HRBM signalling message as specified in 10.4.2. The MMT sending entity continuously runs the hypothetical receiver buffer model to verify that the selected end-to-end delay and buffer size are aligned and do not cause buffer under-runs or overruns. At the MMT receiving entity side, the MMT receiving entity shall perform the buffering as instructed, so that each data unit experiences the signalled fixed end-to-end delay Δ before its payload is delivered to the application layer. Under the assumption that clocks are synchronized, the output time is then calculated based on the transmission timestamp and the signalled fixed end-to-end delay.

11.8 HRBM with pacing buffer

As the buffering of the excessive media data needs to be stored before the processing of packets for any further operation, the proposed HRBM adds an additional buffer in front of the buffers of the MMTP HRBM. Operation of the MMTP HRBM including MMTP pacing buffer is as follows;

— The value of the fixed end-to-end delay of the MMTP HRBM is set to the value of delay contributed by the FEC operation

— Each MMTP packets shall be stored in the pacing buffer for a specific time decided by an MMTP receiving entity before any further processing. The amount of time for storing each MMTP packets are decided by an MMTP receiving entity to maintain the amount of the MMTP packets stored in the MMTP pacing buffer to the value of target_pacing_buffer_level of the PSF message and delivery rate of the packets to the FEC decoding buffer to the value of the pacing_buffer_removal_rate of the PRR message.

— The value of the timestamp field of each MMTP packets being delivered to the FEC decoding buffer shall be increased by the amount of the time such packet actually stored in the MMTP pacing buffer.

— Rest of operation of the MMTP HRBM is same as the case without using the MMTP pacing buffer.

12 Cross layer interface (CLI)

12.1 General

Cross layer interface provides the means within the single MMT entity to support QoS by exchanging QoS-related information between the application layer and underlying layers including the MAC/PHY layer. The application layer provides information about media characteristics as top-down QoS information while underlying layers provide bottom-up QoS information such as network channel condition.

CLI provides the unified interface between the application layer and various network layers including IEEE 802.11 WiFi, IEEE 802.16 WiMAX, 3G, 4G LTE, etc. Common network parameters of popular network standards are abstracted as the NAM for static and dynamic QoS control of real-time media application through any network.

12.2 Cross layer information

12.2.1 General

MMT defines an interface for exchanging cross layer information between the application layer and the underlying network layer. The interface allows for top-down as well as bottom-up flow of cross layer information. The cross layer information provides QoS information that may be used by the involved functions to optimize the overall delivery of the media data. MMT entities may support this interface for cross layer information.

12.2.2 Top-down QoS information

The application layer provides top-down QoS information about media characteristics to underlying layers. There are two kinds of top-down information such as Asset level information (e.g. ADC) and packet level information. Asset information is used for capability exchange and/or (re)allocation of resources in underlying layers. Packet level top-down information is written in the appropriate field of every packet for underlying layers to indentify QoS level to support.

12.2.3 Bottom-up QoS information

The underlying layers provide bottom-up QoS information to the application layer about time-varying network condition which enables faster and more accurate QoS control in the application layer. Bottom-up information is represented as an abstracted fashion to support heterogeneous network environments. These parameters are measured in the underlying layers and read by the application layer, periodically or on request of the MMT application.

12.2.4 Network abstraction for media (NAM)

12.2.4.1 General

Network abstraction for media (NAM) is used for an interface between application layers and underlying layers. NAM provides unified representation of network QoS parameters for assuring to communicate with legacy and future standards of underlying layers.

If an MMT receiving entity supports CLI, it shall support the generation of all CLI parameters.

12.2.4.2 Absolute NAM

Absolute NAM is raw QoS value measured in each appropriate unit. For example, bitrate is represented in the unit of bits per second while jitter is in the unit of second.

12.2.4.3 Relative NAM

Relative NAM represents the ratio of the expected NAM value to the current NAM value so that it is unit-less and informs tendency of change.

12.2.5 Syntax

The CLI information is exchanged using a Network Abstraction for Media (NAM) or a relative NAM.

The syntax of the absolute parameters for NAM is defined in Table 104.

Table 104 — Absolute NAM

Syntax	Size (bits)	Mnemonic
NAM {		
CLI_id	8	unsigned integer
available_bitrate	32	float
buffer_fullness	32	float
peak_bitrate	32	float
average_bitrate_period	16	unsigned integer
current_delay	32	float
SDU_size	32	unsigned integer
SDU _loss_ratio	8	unsigned integer
generation_time	32	unsigned integer
BER	32	float
}		

The syntax of the relative parameters for NAM is defined in Table 105.

Table 105 — Relative NAM

Syntax	Size (bits)	Mnemonic
relative_NAM {		
CLI_id	8	unsigned integer
relative_bitrate	8	float
relative_buffer_fullness	8	float
relative_peak_bitrate	8	Float
average_bitrate_period	16	unsigned integer
current_delay	32	float
generation_time	32	float
BER	32	float
}		

12.2.6 Semantics

`CLI_id` – is an arbitrary integer number to identify the underlying MMT delivery network for the MMT protocol session.

NOTE Identification of MMT protocol session is outside the scope of this document.

`available_bitrate` – is the instantaneous bitrate at measured `generation_time` that the scheduler of the underlying network expects to be available for the MMTP session which carries the MMTP packet.

The available_bitrate is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

buffer_fullness – signals the buffer level of the generating function. The buffer is used to absorb any excess data caused by the data rates above the available_bitrate. The buffer_fullness is expressed in bytes.

peak_bitrate – is maximum allowable bitrate at measured generation_time that the underlying network is able to handle temporarily as input from to the MMTP session which carries the MMTP packet. The peak_bitrate is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

average_bitrate_period – provides the period of time over which the average bitrate of the input of MMT procotol session that carries the MMT packet shall be calculated. The average_bitrate_period is provided in units of milliseconds.

current_delay – is measured transport time for delay between the last hop network entity and receiving entity. The current_delay is expressed in milliseconds.

SDU_size – specifies the length of the service data unit (SDU) in which the underlying network carries the MMTP packet over the MMTP session. It is expressed in bits. Overhead for the protocols of the underlying network is not included.

SDU_loss_ratio – is a fraction of SDUs lost or detected as erroneous. Loss ratio of MMTP packets can be calculated as a function of SDU_loss_ratio and SDU_size. The SDU_loss_ratio is expressed in percentile.

generation_time – is generation time of the current NAM which is based on the NTP timestamp. The generation_time is expressed in milliseconds.

relative_bitrate – the available_bitrate change ratio (%) between the current NAM and the previous NAM parameter.

relative_buffer_fullness – the remaining buffer_fullness change ratio (%) between the current NAM and the previous NAM parameter.

relative_peak_bitrate – the peak_bitrate change ratio (%) between the current NAM and the previous NAM parameter.

BER – bit error rate is the last measured BER at the PHY or MAC layer. For BER from the PHY layer, the value shall be presented as a positive value. For BER from the MAC layer, this value shall be presented as a negative value of which the absolute value is to be used.

13 MMTP Session Setup and Control over Unicast

13.1 Session Description for MMTP

The SDP defines a widely used format for describing media sessions and is defined in.[13] This section defines the SDP parameters that can be used for describing MMTP sessions.

The required SDP parameters for describing an MMTP session are:

— The destination information, consisting of the destination IP address and port number

— An indication that the session is an MMTP session

— The version number of the MMTP protocol used in the current MMTP Session

The optional SDP parameters are:

— The start and end time of the MMTP session

— Bandwidth specification

— Description of the object flows of the MMTP sessiona

An MMTP session consists of one or more object flows, each of which is identified by a unique packet identifier. The packets that belong to the same object flow use the same packet identifier and build a sub-flow of the MMTP flow.

The description of the SDP parameters is provided in the following sections.

13.1.1 MMTP Protocol Identifier

The ABNF syntax for the "m=" line as specified by[13] is defined as follows:

media-field = "m=" media SP port {"/" integer] SP proto 1*(SP fmt) CRLF

proto = "UDP/MMTP" | “UDP/TLS/MMTP”

The "MMTP/UDP" protocol identifier specifies that the session being described will use the MMTP protocol on top of a UDP connection. The "fmt" list may be used to describe the object flows that are delivered as part of the MMTP session.

13.1.2 Object Flow Semantics

An MMTP session consists of an MMTP flow that is delivered during a designated period of time that is indicated by the session start and end time or through external means. Each MMTP sub-flow carries objects of an object flow. An object flow is composed of a set of related objects that are meant to be consumed together by the receiver. An object flow may carry objects of one of the following types:

— Media objects that are formatted as fragmented ISOBMFF files and that are meant to be consumed sequentially. In MMT terms, MPUs would constitute the objects of such an object flow, which is named as Asset.

— The object flow consists of generic objects that are meant to be consumed simultaneously or in sequence.

— The object flow consists of objects that carry signaling messages that are relevant to the consumption of the MMTP flow.

— The object flow carries FEC repair packets.

The SDP descriptors enables providing basic descriptive information about the object flow. The related SDP parameters are described in the following section.

13.1.3 Object Flow Descriptors

The "fmt" field in the media line is used to identify the relevant object flows of an MMTP session and to indicate its type. In addition, a textual description of the object flow may be provided separately. The ABNF syntax of the object flow description is defined as follows:

of-descriptor = "a=of:" fmt SP of-identifier SP type-indicator CRLF

*of-identifier = "flowid=" 1*DIGIT*

type-indicator = "type=" type ["/" subtype]

type = "MPU" / "GF" / "Signaling" / "FEC"

The flowid is set to the packet_id of the MMTP sub-flow that carries the objects of this object flow. The type of the object flow corresponds to one of the object flow types described in Section 12.2. The subtype field is a type specific optional field that can be used to indicate a more specific type of the

object flow. As an example, a subtype "MP" of type "signaling" may be used to indicate that the object flow carries (possibly among others) the MP tables of this MMTP session.

13.1.4 SDP Syntax Examples

This section gives examples of the use of SDP attributes to describe an MMTP session.

v=0

o=user 6431641313 1 IN IP4 10.10.52.13

s=An MMTP session

t=1411639200 1427277600

*a=source-filter: incl IN IP4 * 10.10.52.13*

m=application 12345 MMTP/UDP 100 101 102 103 104

a=of:100 flowid=0 Signaling/PA

a=of:101 flowid=7623 MPU

a=of:102 flowid=7624 MPU

a=of:103 flowid=7625 GF

a=of:104 flowid=7626 FEC

The SDP fragment shows an example SDP instance for an MMTP session. The MMTP session declares a set of 5 object flows. The first object flow, which has the packet_id 0, is the flow that carries signaling messages and in particular PA messages that describe the MMT services. Object flows with packet_id 7623 and 7624 carry media data of 2 different assets using the MPU mode. Object flow with packet_id 7625 carries generic files that are carried using the GFD mode. Finally, object flow with packet_id 7626 carries FEC repair data for one or more other object flows. The description of the FEC protected object flows is provided through the FEC signaling message.

13.2 RTSP

13.2.1 General

The RTSP is an application level protocol for starting and controlling delivery sessions of real-time media data. RTSP can control either a single or multiple time-synchronized streams of continuous media. RTSP relies on SDP for describing these streams.

RTSP may be used to setup and control MMTP sessions. If RTSP is supported then:

— MMT sending and receiving entities shall implement the DESCRIBE, SETUP, PLAY, PAUSE, and TEARDOWN methods

— MMT sending and receiving entities should implement the SET_PARAMETER method

— MMT sending and receiving entities shall support SDP as the session description format and shall include that as part of the Accept header field

— The SETUP request shall include the “MMT receiving entity_port” parameter and the response should include the “MMT sending entity_port” parameter. The former shall indicate the UDP port on which the MMT receiving entity wishes to receive the MMTP flow and the latter shall indicate the UDP port on which the MMT sending entity wants to receive feedback messages.

— The MMT sending and receiving entities shall support the “utc” and the “mmt” range formats

13.2.2 MMT Range Format

The mmt range format may be used to select the starting and ending range of media data to stream using the packet_id and the MPU sequence number. The MMT sending entity shall deliver media starting from the indicated MPU. It shall also deliver the media from other sub-flows maintaining synchronization, i.e. starting from MPU with the highest start time less than the selected MPU start time.

The mmt range has the following ABNF syntax:

mmt-range = (mmt-pos '-" [mmt-pos]) | ("-" mmt-pos)

mmt-pos = packet_id ":" mpu_sequence_number

13.2.3 MMT sub-flow Parameter

The MMT sub-flow parameter is used to select the sub-flows that the MMT receiving entity wishes to receive from the MMT sending entity. It is used both in the SETUP request as part of the Transport header field or is send as part of the SET_PARAMETER message body.

The MMT sub-flow parameter has the following ABNF syntax when used in the Transport header field:

MMT_subflow_parameter= "MMT_subflows" "=" 1(packet_id ",")*

It has the following ABNF syntax when used with the SET_PARAMETER method:

MMT_subflow_parameter= "MMT_subflows" ":" 1(packet_id ",")*

When present, the MMT sending entity should only include the MMT sub-flows that are requested by the receiver, unless these sub-flows are carrying signaling messages. Sub-flows carrying signaling messages cannot be turned off/removed from the transmission.

13.3 WebSockets for MMTP

13.3.1 General

WebSockets[27] may be used to setup and control an MMT session/presentation. When WebSockets are used for session control, media delivery is multiplexed together with the control protocol as part of the same connection.

An example use case of MMT over WebSocket is described in Annex I.

13.3.2 Upgrade to MMT over WebSocket

The MMT sub-protocol shall be identified by the name "mmt" in the handshake request. A MMT receiving entity wishing to use WebSockets for controlling an MMT session shall include the keyword "mmt" as part of the *Sec-WebSocket-Protocol* header field together with the HTTP 1.1 protocol upgrade request.

The following is an example of a WebSocket handshake in which the MMT receiving entity requests an upgrade to MMT sub-protocol over WebSockets from the MMT sending entity:

```
GET / HTTP/1.1
Host: mmt.mpeg.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dFhmILNhbYCsXSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: mmt
Sec-WebSocket-Version: 13
```

The response from the MMT sending entity for a successful upgrade may look like this:

HTTP/1.1 101
Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: p3sPMLciToaR9kXGzhzYRbL+xOo=
Sec-WebSocket-Protocol: mmt

13.3.3 Framing in the MMT sub-protocol

13.3.3.1 Encoding

After a successful upgrade of the protocol to MMT over WebSockets, the MMT receiving entity and MMT sending entity are able to exchange MMT data frames over the established WebSocket connection. The MMT sub-protocol shall use the 'binary' format (opcode 'binary' or any 'continuation' frames thereof) for all messages exchanged over the WebSocket connection.

13.3.3.2 Frame Format and Semantics

The MMT sub-protocol defines a frame structure that is used to frame the payload for transmission over binary WebSocket frames. The MMT sub-protocol frame consists of a frame header as depicted in Figure 24 and frame payload. The frame header shall be provided as WebSocket frame Extension Data, which shall be present and of which the size can be determined as 4+4*EXT_LENGTH bytes as given by the MMT sub-protocol frame header.

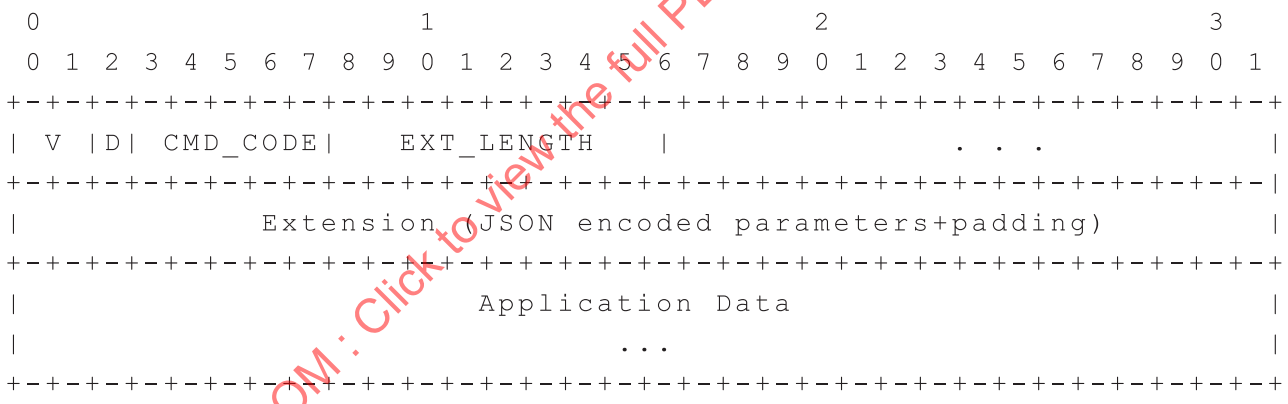


Figure 24 — MMT sub-protocol frame header

The MMT sub-protocol frame header is defined as follows:

`V (2 bits)` - The version number of the MMT sub-protocol. The current version number shall be set to '00'.

`D (1 bit)` - The data bit indicates if the content is data or control. If the data flag is set then the payload shall be the MMTP packet and CMD_CODE should be set to 0x1F.

`CMD_CODE (5 bits)` - Indicates the control command that is sent in this request/response. The following commands are currently defined. The command codes are defined in the table below.

Table 106 — MMT WebSocket Sub-protocol Commands

Command Code	Value	Description
DESCRIBE	1	A request by the MMT receiving entity to the MMT Sending entity to request the description of the MMT Package. The extension header shall contain the "url" parameter that provides the URL of the MMT package that is requested.

Table 106 *(continued)*

Command Code	Value	Description
START	2	Request the start of the transmission of the MMTP flow. The parameters that are included in the extension header are the URL that identifies the MMT Package provided by the parameter "url". Optionally, the request may also include the range parameter "range" and a selection of the sub-flows "packet_id". This message goes from the MMT receiving entity to the MMT sending entity.
STOP	3	Stop the transmission of the MMTP sub-flows that are identified by the packet_id. The packet ids are comma separated.
ACK	4	An acknowledgment by the MMT sending entity to the MMT receiving entity that confirms the acceptance of the previous request.
FEEDBACK	5	The MMT receiving entity periodically transmits feedback to the MMT sending entity to inform about the current buffer level and the current playback position. The payload shall be the NAMF message.

`EXT_LENGTH (8 bits)` - Provides the length in 4 bytes of the extension data that precedes the application data. The extension header must be a JSON encoding of additional information fields that apply to the current frame. To align with 4 byte boundaries, padding 0 bytes may be added after the extension header.

13.3.4 Sub-protocol Registration

Websockets[27] requires that sub-protocols be registered with the IANA. The registry requires the following information:

Subprotocol-Identifier: shall be set to "mmt"

Subprotocol Common Name: shall be set to "MMT"

Subprotocol Definition: shall refer to the specification of the WebSocket MMT subprotocol as defined in this section.

13.4 Multipath Support in MMTP

13.4.1 General

Multiple transport connections using multiple network interfaces can be established between the MMT sender and the MMT receiver so data can be sent faster and in a reliable fashion to the destination. In addition, MMT application can be made link aware i.e. the application can have visibility into performance of each of the multiple network paths and make informed decisions as to what different types of payload (MMT asset types) can be transferred on which network path (interface). With this capability, MMT application can dynamically change the network path if it finds out that one or more of the possible network paths are performing bad.

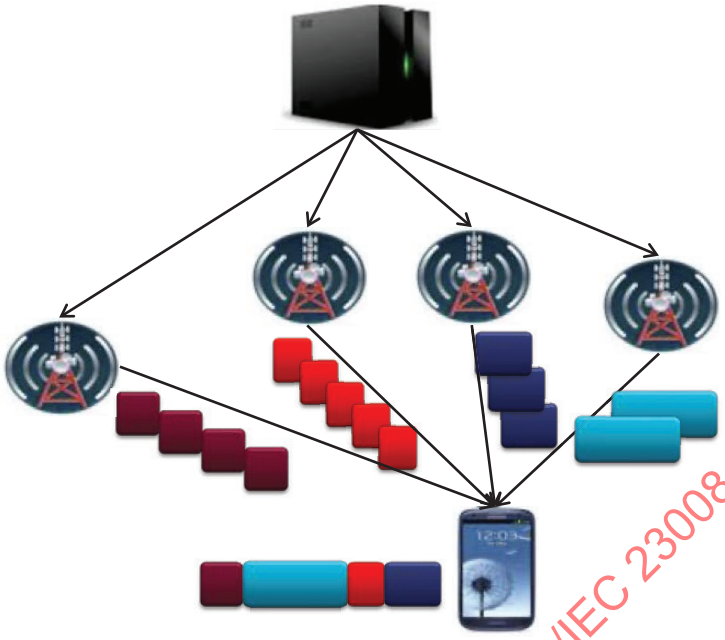


Figure 25 — Multipath Delivery in MMT

Figure 25 shows an architecture diagram for multipath delivery in MMT. Multiple transport connections can be setup between the MMT sender and MMT receiver using different network interfaces available on the MMT sender and the MMT receiver. Different packet flows of the multimedia data stream can be transferred on different transport connections thus using multiple network interfaces (paths) available for data delivery. Upon receiving packet flows on different network paths, MMT receiver can compute per network quality statistics (QoS, QoE etc.) and give this information as feedback to the MMT sender. The MMT sender can use per network path feedback to make informed decisions for sending subsequent packets of different packet flows. If one network path performs poorly during the course of the session, a different network path can be chosen for subsequent packet delivery. As a result, poor performance of one network path does not impact the overall quality of media delivery, and therefore has no adverse impact on the end user experience.

For supporting multipath delivery in MMT, control protocols (signaling protocols) for setting up MMT sessions have to be enhanced to support multipath setup capabilities. MMT sessions can be setup using signaling protocols such as RTSP and HTTP. This section presents how two communicating parties can negotiate multipath capabilities and setup a MMT multipath session. In addition, it also focuses on MMT feedback message enhancements for supporting per network path feedback.

13.4.2 Multipath Negotiation

MMT sender (MMT sending entity) and MMT receiver (MMT receiving entity) should have capabilities to negotiate for multipath MMT setup. This negotiation is performed during the MMT session setup using signaling protocols such as RTSP and HTTP.

13.4.2.1 Multipath Feature Discovery

If RTSP is used as the signaling protocol, the MMT receiving entity and MMT sending entity can mutually discover multipath capability support by sending OPTIONS request to each other. As OPTIONS request can be generated both by the MMT sending entity and the MMT receiving entity, the OPTIONS request becomes a good candidate method for enquiring about multipath feature support. We propose to register a new option tag called “multipath” with IANA to be used in the Require header of an OPTIONS request as shown below

Require: multipath

For an OPTIONS request from one party with option tag “multipath”

1. If the other party supports multipath feature, it shall respond with an affirmative response.
2. If the other party does not support multipath feature, it shall respond with Unsupported header.

13.4.2.2 Session Setup with Multipath

For setting up a multipath MMT session, the MMT receiving entity and MMT sending entity need a mechanism for exchanging the body description (using SDP with RTSP and HTTP) and a mechanism for correlating flows on different network paths.

13.4.2.2.1 Body Descriptions

For a session description request from a MMT receiving entity, the MMT sending entity should send a standard SDP if it does not intend to use multipath session. However, if the MMT sending entity intends to use a multipath session, the MMT sending entity should send an SDP with a new attribute called “multipath” with syntax as shown below.

a=multipath:<value>

where <value> gives the maximum number of distinct interfaces supported by the MMT sending entity for multipath delivery

In the SDP, the MMT sending entity should include media descriptions for each of the multipath interfaces as given by the multipath attribute. For synchronization of different media streams, SDP attributes such as “group” and “mid” (media stream identification) shall be used. These attributes along with media specific connection information tells the MMT receiving entity that the MMT sending entity supports multipath with interface grouping as specified with “group” and “mid” attribute semantics and that the MMT sending entity would use multiple interfaces to receive packet data in the format indicated by the payload type values in the respective media descriptions.

Once the MMT receiving entity receives SDP from the MMT sending entity, based on the multipath attribute, the MMT receiving entity can setup multiple connections to the MMT sending entity – one each to each of different network end points specified in the SDP. Further, the MMT receiving entity can request to receive data from the MMT sending entity on different interfaces of its own if it intends to use multipath delivery.

1. If RTSP is used as the signaling protocol for session setup and if the MMT receiving entity intends to use multipath delivery, the MMT receiving entity should send multiple SETUP requests to the MMT sending entity with different transport header values to receive data on different network interfaces.
2. If HTTP and WebSockets is used for setting up unicast MMT sessions and if the MMT receiving entity intends to use multipath delivery, the MMT receiving entity should send multiple HTTP upgrade requests to the MMT sending entity to receive data on different network interfaces.

When a MMT receiving entity makes multiple SETUP requests (RTSP) or multiple WebSockets requests (HTTP) for multipath capabilities, the MMT sending entity should always be able to deliver the data to different network interfaces of the MMT receiving entity as requested in the corresponding signaling messages. Using RTSP and HTTP signaling methods described above and the new SDP attributes and option tags described in this contribution before, the MMT receiving entity and MMT sending entity can setup multiple network paths for data delivery.

13.4.2.2.2 Correlating Multiple Flows

To help the MMT receiving entity and MMT sending entity setup multipath delivery, there is a need to correlate flows on different network interfaces. To correlate flows on different network paths, the MMT receiving entity should include a new header field called “MultipathId” in the corresponding signaling request message. This new header field should be included in all the session setup messages

that the MMT receiving entity sends to the MMT sending entity and have the same value in all the setup request messages. When the MMT sending entity receives multiple setup messages with the same value for "MultipathId" header, the MMT sending entity should recognize that the MMT receiving entity is intending to use multipath delivery and that the MMT sending entity should be able to send data to those multiple network interfaces of the MMT receiving entity.

Similarly, if the MMT sending entity intends to use multipath, the MMT sending entity should include "MultipathId" header of its own in all the response messages to the MMT receiving entity. The values of "MultipathId" header in all response messages from the MMT sending entity should be identical. When the MMT receiving entity sees that the MMT sending entity is sending responses with "MultipathId" header, the MMT receiving entity should recognize that the MMT sending entity is intending to use multipath delivery feature and that all flows that result because of the corresponding signaling messages belong to the same MMT session.

13.4.3 Session Modification with Multipath

It is possible that MMT receiving entity and MMT sending entity intend to enable or disable support for multipath during the session.

a) Add multipath support during the session

The MMT receiving entity and MMT sending entity open a transport connection (TCP or UDP) for data delivery using RTSP or HTTP signaling protocol without using multipath capabilities.

i. MMT receiving entity proposal for multipath: During the session if the MMT receiving entity likes to use multipath delivery feature, the MMT receiving entity can check the multipath capabilities of the MMT sending entity as described in section 1. If the MMT receiving entity and MMT sending entity decide to use multipath delivery, the MMT receiving entity and MMT sending entity should setup multiple network paths as described in section 1 MMT sending entity proposal for multipath: If during the session the MMT sending entity likes to use multipath delivery feature, it can check multipath capabilities of the MMT receiving entity as described in section 1 The MMT sending entity should then send a new message body (SDP with multipath attribute) with updated media description information to specify multipath capabilities. If the MMT receiving entity elects to use multipath, it can setup multiple connections as described in section 1

b) Modify multipath support by adding or removing interfaces

The MMT receiving entity and MMT sending entity can add or drop network paths if they are already using multipath delivery.

i. MMT receiving entity: The MMT receiving entity can drop a network path by sending termination messages to the corresponding signaling setup messages. For such termination messages for a network path, the MMT sending entity should send an affirmative response and continue using the remaining network paths for data delivery.

ii. MMT sending entity: The MMT sending entity can add or drop network paths by announcing a new SDP and ask the MMT receiving entityMMT receiving entity to setup connections based on modified SDP as described in section 1

c) Drop multipath support during the session

The MMT receiving entity and MMT sending entity can drop multipath support during the session. The MMT receiving entity can do so by 1) terminating all network paths and setting up one path for data delivery or 2) terminating all but one paths and modifying the remaining path to carry required payloads. The MMT sending entity can drop multipath support by sending a modified media description (using SDP without the "multipath" attribute). For such multipath dropping requests from either of the parties, the other party should recognize that the former is intending to drop multipath feature support for the session from the former's end.

The support for multipath is mutually exclusive between the MMT receiving entity and MMT sending entity i.e. the MMT receiving entity and MMT sending entity can indulge in multipath irrespective of other party's preference for multipath delivery. If a party does not have multiple interfaces or would not like to use multipath delivery feature, it can ignore to include the "MultipathId" header field in its outgoing messages. However, if the other party includes the "MultipathId" header, it should be able to identify that the other party is requesting for multipath support and therefore should be able to stream packet data to multiple interfaces of the other party.

13.4.4 Feedback Message Enhancements

The MMT sender and MMT receiver setup multiple network paths as described in section 13.2 to support multipath delivery. When MMT receiver starts receiving data on those multiple paths, it can keep track of the quality information (e.g., QoS, QoE statistics) for each network path. This quality information for each network path can be sent back to the MMT sender so the MMT sender has better visibility into performance of each network path. With this information, the MMT sender can take necessary actions e.g., re-transmit lost packets on a given network path, transport packet flows on a different network path etc. This specification describes a method for receiver feedback to the sender using the ARQ (signaling and feedback) messages and Reception Quality Feedback. However, in the current specification, these messages carry feedback information for one network path on which all the packet flows are transported. Therefore, there is a clear need for enhancing the receiver feedback messages to account for multiple network paths between the sender and the receiver. In particular, the following enhancements have to be supported:

1. The receiver should be able to compute quality information on a path-by-path basis.
2. The receiver should be able to send ARQ feedback messages consisting of multiple packet flows. In addition, the receiver should provide path identifier information for each flow. When the sender receives updated ARQ feedback messages (consisting of packet flows and network path identifiers), it should be able to perform corrective actions on a path-by-path basis.
3. The receiver should be able to send Reception Quality Feedback for multiple network paths. In addition, the receiver should provide path identifier information and flow identifiers for each flow. When the sender receives the reception quality feedback information, it should be able to recognize quality information for each different network path and take corrective actions on a path-by-path basis.

14 CDN Support

14.1 General

Many of the CDNs nowadays support the HTTP/1.1 protocol exclusively. Several of them have recently started to add support for the HTTP/2 protocol. TLS and SSL are also widely supported to enable serving content securely over an encrypted channel. However, little or no CDNs offer support for other protocols and especially those based on UDP.

When using MMT, content can be streamed using the MMTP protocol over UDP or over TCP. A viable approach is to use WebSockets to migrate an HTTP session into an MMTP over TCP session (see Annex I). The WebSockets protocol serves as an enabler to assure this migration and convert an HTTP session into a plain TCP connection.

In order to integrate MMT into existing CDNs, a solution based on load balancing provides a viable alternative. Connections are established to a load balancer, which provides the front end for incoming traffic requests. Load balancers may use different techniques such as DNS to route the traffic through the most appropriate computing instance, usually based on availability and proximity to the requesting client. The serving computing instance may get its content from regular CDN edge nodes and convert the traffic to MMTP flows.

Figure 26 depicts such an instantiation.

Signalling messages may be used to support the conversion between cached objects and MMT assets served over MMTP flows.

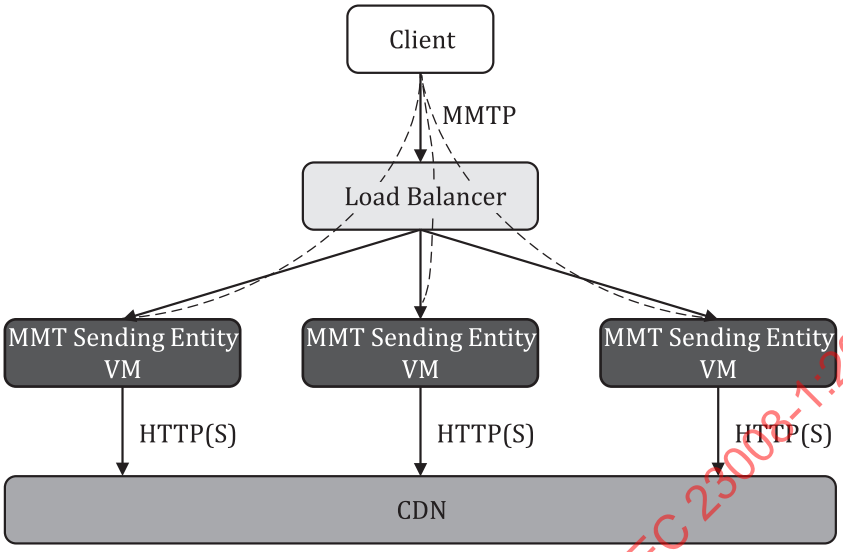


Figure 26 — MMT integration in CDNs

14.2 DNS Resolution of MMT URLs

14.2.1 General

When the MMT receiving entity consumes the MMT based media streaming service over the Internet, MMT receiving entity should do the media source URLs resolution process. In order to that MMT receiving entity should request the DNS query message to DNS or MANE DNS. DNS or MANE DNS should response the both IP address of media source MMT and MMT-specific media source information (e.g. MMT_Package_id, Asset_id, MPU_sequence_number) by using DNS response message. In this clause, we describe the DNS message for supporting MMT URL resolution.

14.2.2 DNS query message for MMT URLs

When the MMT receiving entity wants to receive the contents from the origin media source provider server, MMT receiving entity should request media source URL resolution to DNS or MANE DNS. By using DNS query message, MMT receiving entity can send the media source URL information to MANE DNS. DNS query message is composed of three entities: QNAME, QTYPE, and QCLASS. QNAME indicates the domain name part of a uniform resource locator (URL) which called a fully qualified domain name (FQDN). QTYPE indicates the type of the query, i.e. the type of RR which should be returned in responses. QCLASS indicates the class field such as IN for internet.

14.2.3 DNS response message for MMT URLs

The DNS response message is used to deliver the media source URL resolution result to MMT receiving entity. By using resource records, MANE DNS or DNS answering the query message from the MMT receiving entity. In order to enable DNS resolution of the URLs of media source that are potentially delivered over MMTP, a set of DNS Resource Records need to be used.

Service (SRV) resource records (RR) enable to specify the location of the servers for a specific service, protocol, and DNS domain.

The MMT SRV RR follows the format specified as follows:

_MMT(Service).proto.name TTL class SRV Priority Weight Port target

The _Service field specifies the name of the desired service, such as http or telnet. Some services are defined as defined in Assigned numbers or locally. The _Proto field specifies symbolic name of the desired protocol, such as TCP or UDP. The Proto is case insensitive. The Name field specifies the domain name to which the resource record refers. The TTL (Time to Live) field specifies the time interval that the resource record may be cached before the source of the information should again be consulted. The class field identifies the type of resource record. SRV records occur in the IN Class. The Priority field specifies the priority of the origin media source location or server. Clients attempt to contact the host with the lowest priority. The Weight field is a load balancing mechanism. When the priority field is the same for two or more records in the same domain, clients should try records with higher weights more often, unless the clients support some other load balancing mechanism. The Port field shows the port of the service on origin media source location or server. The target field shows the fully qualified domain name for the host supporting the service. The target also contains the origin media source location of media source for MMT streaming service that carries the resource as part of the domain name of the internet that is serving the content over MMTP.

The following is an example of an SRV RR for the MMT streaming service:

_mmt._udp.example.com 14400 IN SRV 1 100 53 a123.origin.example.com.

Uniform Resource Identifier (URI) RR is an alternative to the SRV RR whereby the returned URI strings may be used directly by the requesting application unlike the SRV RR in which the useable URI must be assembled from the search and results information. Similar to the SRV RR it returns both weight and priority values which can be used to select an appropriate URI from multiple results. However, unlike the SRV no port number is returned in the URI RR(s) since this information is contained (where appropriate) in the URI string.

The MMT URI RR follows the format specified as follows:

Owner-name TTL Class URI Priority Weight Target

The service information is encoded in owner name.

The following is an example of an URI RR for the MMT streaming service:

_mmt._udp IN URI 0 10 "mmt://a123.origin.example.com://53/package1/asset1/1.mpu"

The response also contains an Address (A) or IPv6 address (AAAA) record for the target and it corresponds to the IP address of the media source location.

The A RR is specified to follow the following format:

owner-name ttl class rr ipv4

The following is an example of a corresponding A RR:

a123.origin.example.com 14400 IN A 10.100.20.1

Additionally, in media resource resolution result, a TXT resource records is returned to the MMT receiving entity, containing MMT specific information. The following parameters are defined:

The TXT RR is specified to follow the following format:

Owner class ttl TXT "<attribute name>=<attribute value>"

An example of such a TXT RR is given as follows:

a123.origin.example.com IN 14400 TXT "IP_address_range=10.20.0.1~10.20.0.100"

a123.origin.example.com IN 14400 TXT "mmt_package_id=1"

a123.origin.example.com IN 14400 TXT "asset_id=1"

a123.origin.example.com IN 14400 TXT "Starting_number_of_mpu_sequence =1"

a123.origin.example.com IN 14400 TXT "End_number_of_mpu_sequence =10"

In TXT RR, IP_address_range provides IP address ranges information of client. mmt_package_id provides unique identifier of the Package to MMT receiving entity. asset_id provides identifier of Asset which is included in the current MMT_package.

starting_number_of_mpu_sequence indicates the first MPU sequence number of each asset and

end_number_of_mpu_sequence indicates the last MPU sequence number of each asset.

14.2.4 DNS update message of MMT URLs

When the MMT receiving entity requests a media source URL resolution to a DNS or a MANE DNS, then the (MANE) DNS should resolve the media source URL referenced by the media resource identification information. If the (MANE) DNS does not have the media resource identification information which is requested by the MMT receiving entity, then the media resource identification information should be updated using the DARI proxy or the Name Server which provides actual media resource identification information. By using DNS update request message, the DNS or MANE DNS can request the updated media resource identification information from a DARI proxy or Name Server. After the DARI proxy or Name Server receives the DNS update request message, it should check the request and send the updated media resource identification information by using DNS update response message.

14.2.5 Media Resource Update (MRU) Message

When the MANE DNS received the updated media resource identification information which is given by MMT sending entity or MANE (i.e. DARI proxy), MANE DNS should send the updated media resource identification information to MMT receiving entity. By using Media Resource Update (MRU) message, MANE DNS can notify the updated media resource identification information to MMT receiving entity. The syntax for MRU message is shown in Table 107.

14.2.5.1 Syntax

Syntax of the MRU message is shown in Table 107.

Table 107 — MRU message syntax

Syntax	Value	No. of bits	Mnemonic
MRU_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
ip_addr_flag		**1**	**boolean**
application_flag		**1**	**boolean**
client_flag		**1**	**boolean**
target_client{			
if(ip_addr_flag){			
if (ip_addr_version == 00) {			
start_ipv4_src_addr		**32**	**uimsbf**
end_ipv4_src_addr		**32**	**uimsbf**
}else if (ip_addr_version == 01) {			
start_ipv6_src_addr		**128**	**uimsbf**
end_ipv6_src_addr		**128**	**Uimsbf**

Table 107 *(continued)*

Syntax	Value	No. of bits	Mnemonic

}
}
if(application_flag == 1){
application_id
		32	**Uimsbf**

}
if(client_flag == 1){
identifier
		16	**Uimsbf**

} } ***Number_of_URLs***	**N1**	**32**	**uimsbf**
for (i=0; i<N1; i++){ ***mmt_URL()*** ***request_type***		**3**	**uimsbf**

if ((request_type == "001b") ‖
(request_type == "000b")) {
MMT_general_location_info()
}			

if ((request_type == "010b") ‖
(request_type == "000b")) {
access_network_descriptor()
}
		4	**uimsbf**

if ((request_type == "011b") ‖
(request_type == "000b")) {
media__resource_info{
MMT_package_id{
		N1	

MMT_package_id_length for (i=0; i<N1; i++) {		**8**	**uimsbf**
MMT_package_id_byte } } ***number_of_assets***		**8**	**uimsbf**

for (i=0; i<N2; i++) {
asset_id()
start _mpu_sequence
end _mpu_sequence
}
		N2
		32
		32
			uimsbf
			uimsbf

} } ***reserved***		**8**	**uimsbf**
} } }			

14.2.5.2 Semantics

`message_id` - indicates MRU message ID. The length of this field is 16 bits.

`version` - indicates the version of MRU messages. The length of this field is 8 bits.

`length` - indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the ECC message. The value "0" shall not be used for this field

`ip_addr_flag` – indicates whether IP address is included or not. If it is set to '1', the IP address is included.

`application_flag` – indicates whether application id is included or not. If it is set to '1', the application id is included.

`client_flag` – indicates whether client id is included or not. If it is set to '1', the client id is included.

`start_ipv4_src_addr` - This field is the IPv4 address of the sender of the packet. It is the start IPv4 address of IPv4 address range.The length of this field is 32 bits.

`end_ipv4_src_addr` - This field is the IPv4 address of the sender of the packet. It is the start IPv4 address of IPv4 address range.The length of this field is 32 bits.

`start_ ipv6_src_addr` - This field is the IPv6 address of the sender of the packet. It is the start IPv6 address of IPv6 address range.The length of this field is 128 bits.

`start_ ipv6_src_addr` - This field is the IPv6 address of the sender of the packet. It is the end IPv6 address of IPv6 address range.The length of this field is 128 bits.

`application_id` – indicates a unique application ID which is uniquely identifies application on the device and in MMT.

`identifier` – indicates a 16-bit identification field generated by the device that creates the DNS query.

`number_of_URLs` - provides the number of `MMT_URLs` which should be updated.

`mmt_url` – indicates the media source URL which is request by MMT receiving entity.

`request_type` – indicates the type of request information in the media resource identification according to Table 108.

Table 108 — value of request_type field

Value	Description
000b	all information (media_resource_location, access_network_descriptor(), media_resource_info)
001b	media resource location
010b	access_network_descriptor()
011b	media_resource_info

`MMT_general_location_info()` - provides the location information of Asset. General location reference information for Asset is used. Only the value of `location_type` between '0x00' and '0x06' shall be used for an Asset location.

`access_network_descriptor()` - indicates type of transmission access channel of media content delivery. Its specific value is decided by `access_network_type` value defined in Table 91

`MMT_package_id` – this field is a unique identifier of the Package that is being served.

`MMT_package_id_length` – the length in bytes of the `MMT_package_id` string, excluding the

terminating null character.

`MMT_package_id_byte` – a byte in the `MMT_package_id`. When the `MMT_package_id_byte` is string, the terminating null character is not included in the string.

`asset_id` – indicates the asset_id of another Asset on which the dependent Asset associated with this descriptor depends. The order of the id-s provided in this descriptor is such that the concatenation of MPUs as defined in clause 6.3 leads to a valid MPU and follows the media dependency hierarchy.

`start _mpu_sequence` - This field is the start number of mpu sequence in asset which is referenced by mmt_URLs. The length of this field is 32 bits.

`end _mpu_sequence` - This field is the end number of mpu sequence in asset which is referenced by mmt_URLs. The length of this field is 32 bits.

14.2.6 MRI Request (MRIR) message

Media resource identification information, which includes the policy to be adopted by the MMT sending entity and MMT receiving entity in the event of dynamic media resource transmission access/session control, shall be transmitted at the beginning or in the middle of a session as the media resource identification information (MRI) message from the transmitting MMT sending entity to the MMT receiving entity. If the MANE DNS or DARI proxy wants to receive or update the media resource identification information, it should request the media resource identification information by using MRI request (MRIR) message. The syntax for MRIR message is shown in Table 109.

14.2.6.1 Syntax

Syntax of the MRIR message is shown in Table 109.

Table 109 — MRIR message syntax

Syntax	Value	No. of bits	Mnemonic
MRIR_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
message_payload{			
ip_addr_flag		**1**	**boolean**
request_flag		**1**	**boolean**
if(ip_addr_flag){			
if (ip_addr_version == 00) {			
ipv4_src_addr		**32**	**uimsbf**
}else if (ip_addr_version == 01) {			
ipv6_src_addr		**128**	**uimsbf**
}			
}			
number of URLs	**N1**	**32**	**uimsbf**
for (i=0; i<N1; i++){			
mmt_URL()			
request_type		**3**	**uimsbf**
}			
reserved		**8**	**uimsbf**
}			
}			

14.2.6.2 Semantics

`message_id` - indicates MRIR message ID. The length of this field is 16 bits.

`version` - indicates the version of MRIR messages. The length of this field is 8 bits.

`length` - indicates the length of the messages in bytes, counting from the beginning of the next field to the last byte of the MRIR message. The value "0" shall not be used for this field

`ip_addr_flag` – indicates whether IP address is included or not. If it is set to '1', the IP address is included.

`request_flag` – indicates whether MRI request message for request media resource identification information or update media resource identification information. If it is set to '0', the request new information. When the request for update media resource identification information, it is set to '1'.

`ipv4_src_addr` - This field is the IPv4 address of the sender of the packet. The length of this field is 32 bits.

`ipv6_src_addr` - This field is the IPv6 address of the sender of the packet. The length of this field is 128 bits.

`mmt_url` – indicates the media source URL which is requested by MMT an receiving entity.

`request_type` - indicates type of request information in media resource identification in Table 110.

Table 110 — value of request_type field

Value	Description
000b	all information
001b	media resource location
010b	access_network_descriptor()
011b	Media resource information

14.3 MANE

14.3.1 Definition

MMT-aware network entity (MANE) is a network entity which aware of media types and its characteristics that is carried through themselves by the tools MMT technology provides as depicted in Figure 27. MANE can acquire this media-related information, such as from header of MMTP packets where media is being carried on and MMTP signalling messages retrieved, etc. And those information enables media-aware operations.

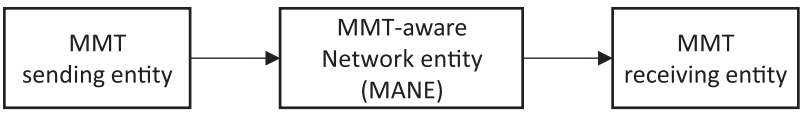


Figure 27 — MANE at network

14.3.2 Interface between MMT sending entity and MANE

The interface between MMT sending entity and MANE refers to the management used by the MMT sending entity to create MMTP session and push content into the MANE for delivery of MMT Package for flexible distribution. The MMTP session may in particular be linear TV programs for which it is possible to have a directly ingest TV content over the IP-based network with same manner of MMT funcationality.

In this case, the interface between MMT sending entity and the MANE is RESTful connection and ingests a set of Assets that are hierarchically organized as follow:

— MMTP session is defined by the profile of MMT Package delivery for distribution related with characteristic of delivery network and scheduling information. Associated configuration parameters are Package description including Package identification, delivery characteristics, delivery and signalling methods and needed MMT receiving entity's capabilities. A MMTP session that may have one or more MMTP flows.

— MMTP flows defines what Asset is going to be distributed, how and when it is distributed to specific delivery network condition. Associated parameters are therefore MMTP flow identification, type of Asset, descriptor of Asset, timing information for presentation , mapping information between Asset and packet flow, AL-FEC and measurement configuration related information. One or more flows can be attached to a MMTP session thus allowing in practice a MMTP session having several delivery networks for Package distribution. However, there can only be one MMTP flow within one single MMTP session at a given time, since there is only one PA message from MMTP sending entity.

— For each of the MMTP session and MMTP flow procedure, the MMT sending entity shall be able to retrieve the state through a MMTP session creation message. The MANE interface supports forwarding of MMTP over UDP based delivery of MMT package.

— Typically, a MMTP session or a MMTP flow is created using a MMTP session creation message, MMTP flow creation message upon the flow, updated with a MMTP session update message, fetched using a MMTP session query message, and deleted using the MMTP session termination message.

The MANE interface defines the following procedures for the MMTP session for media delivery and signalling messages:

— Authentication and authorization procedure to enable MMT sending entity and MANE to cross-authenticate and to enable provisioning of MMTP session at the MANE.

— MMTP session creation by the MMT sending entity and according to the grants given to the MMT sending entity representative based on the used credentials such as token based mechanism.

— MMTP flow creation by the MMT sending entity to configure a MMTP session and enable the MANE to perform MMTP session procedures as described in current specification (mobile MMT).

— Modification and termination of a MMTP session or delivery of MMTP flow.

— MMTP session reporting and feedback procedures from MANE to MMT sending entity to collect the information of specific MMTP session for presentation metrics and delivery metrics.

The procedures are depicted by Figure 28:

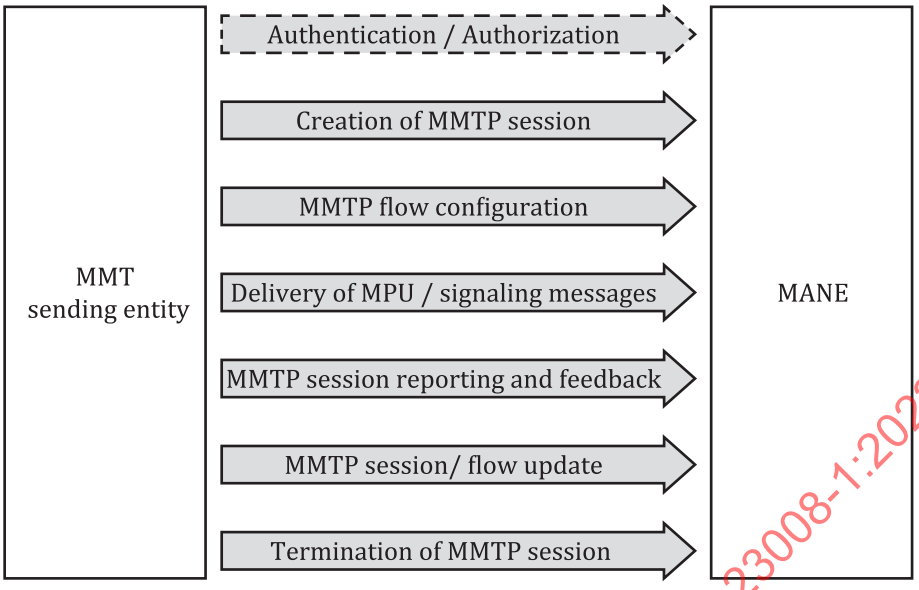


Figure 28 — General procedures between MMT sending entity and MANE

14.3.3 Authentication / Authorization

The intention of the authentication procedure is to reliably identify both MMT entities of the communication channel each other. Based on the authorization information, the MANE checks for every transaction whether the authenticated MMTP sending entity is authorized to execute a procedure. As a result of the successful authentication and authorization procedure, the MANE provides an authorization token to the MMT sending entity. The authorization token must be provided for all subsequent transactions between MANE and MMT sending entity. Figure 29 depicts the authentication process as follows:

— The MANE and the MMT sending entity exchange authenticated Certificates that certifies who they are based on Service Level Agreement (SLA) step.

— The MMT sending entity then either match SSO or provide a pre-exchanged authorization key to initiate a connection to the MANE.

— Based on the assigned specific function of the network node agreed between the Mobile Network Operator (MNO) and the MMT sending entity, the MANE shall respond with a token specifying the entitlements the MMT sending entity end point used.

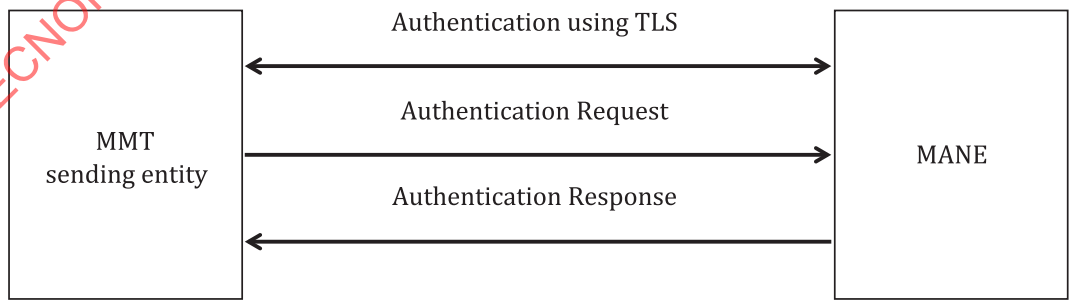


Figure 29 — Authentification procedure of MMT Sending entity and MANE

Note that the MMT sending entity may connect several times to the MANE, each connection having different delegate based on the functions assigned to the requesting MANE.

Security procedures are mandatory. No MMTP delivery of MPU or MMT signalling message shall be exchanged without a valid token. It is assumed that the implementation follows the following assumptions:

— TLS technology is used to secure transport of both MMTP delivery of MPU and MMT signalling message.

— Every transaction requires that MMT entities are authenticated

Finally, the MMT sending entity may fetch its authorization entitlements using a query to the MANE.

14.3.4 Creation of a MMTP session

A MMTP session is a delivery of MMT Package as defined in.[1] Each MMTP session may contain one or multiple MMTP flows. Each MMTP session is time bound by start time and stop time, uses a specific delivery method, and is associated with a target service area which can be used to derive the MMTP session area by logistic information. The stop time may be absent in case of limitless sessions. Figure 30 depicts session creation procedure as follows:

— The authenticated and authorized MMT sending entity creates a new MMTP session by providing Package ID, service class of Package delivery as well as expected MMT receiving entity capabilities. The MMT sending entity may select whether the Contents Provider or the service provider distributes service announcement by providing a description of MMT packages used for operator-driven service announcement.

— Upon successful MMTP session creation by the MANE, the MANE shall provide a unique MMTP session ID, which the MMTP sending entity shall use for subsequent MMTP flow creations and/or MMTP session updates & termination.

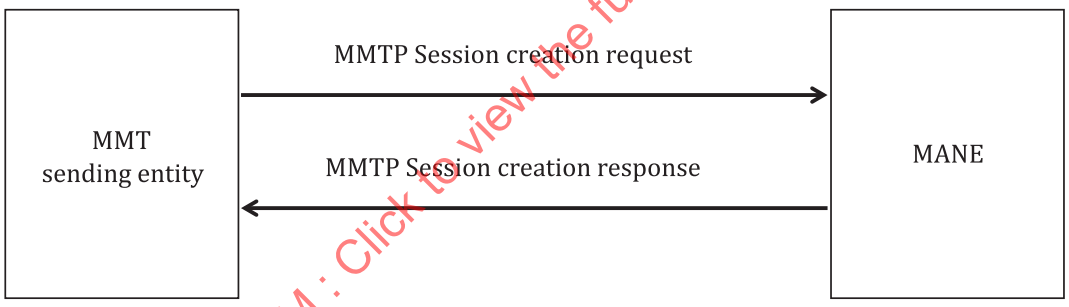


Figure 30 — Session creation procedure between MMT sending entity and MANE

Initially, the description of MMT Package does not contain any delivery network informatoin. The delivery methods will be set during session creation by MMT sending entity or MANE.

14.3.5 Creation of MMTP flow

Once a MMT sending entity has created a MMTP session, it may add one or multiple MMTP flows to this MMTP session. Each flow is mapped with a group of sub-flows of Assets each of which is identified by Asset Identifier, uses a specific packet identification method and is associated with QoS information for delivery of Asset. Figure 31 depicts MMTP flow creation procedure as follows:

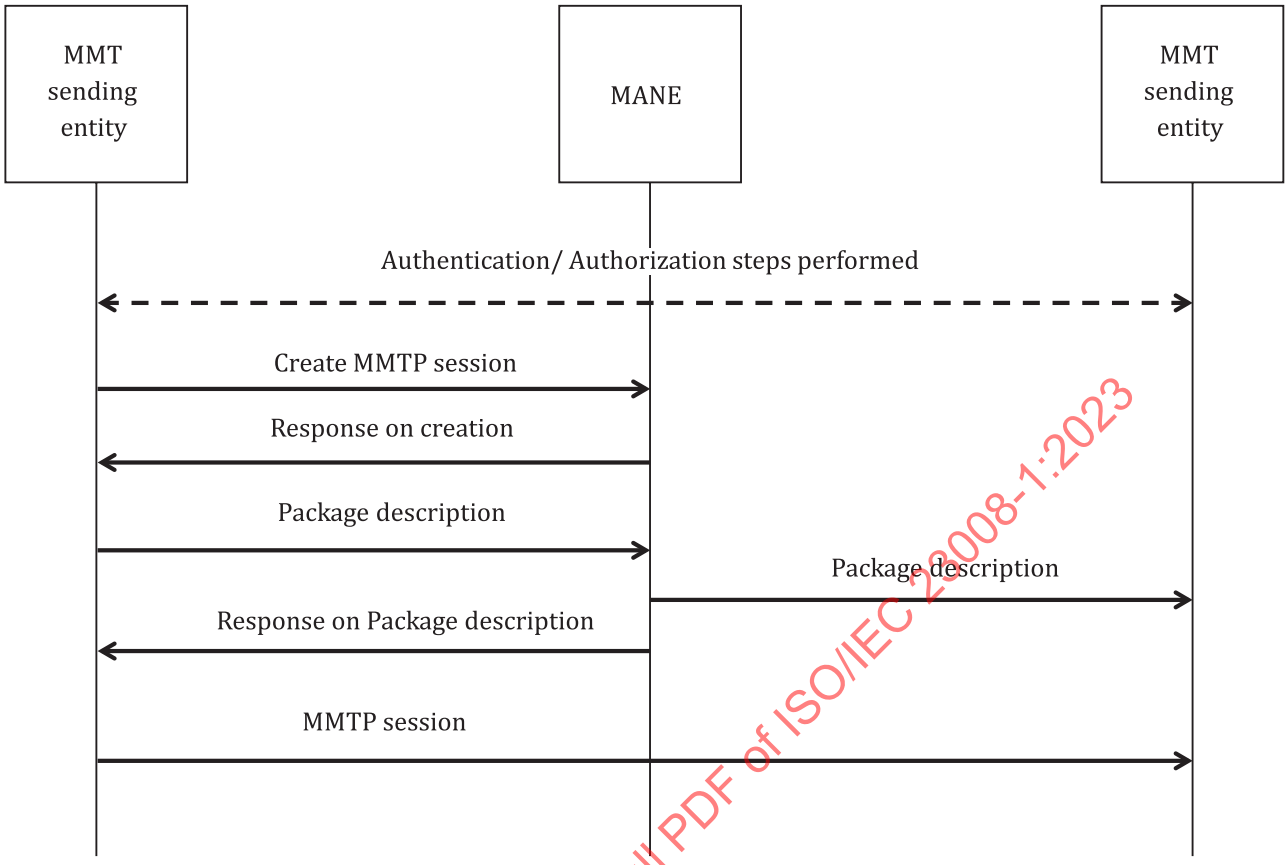


Figure 31 — MMTP flow creation procedure between MMT sending entity and MANE

— The authenticated and authorized MMT sending entity creates a MMTP flow within a MMTP session already created. This triggers the MANE to allocate many parameters from an MMTP session creation message which specifies delivery of Package, including a PA message, MMTP packet_id, source IP Address for multicast, UDP Port and QoS information, etc. The MMT sending entity may select continuous push or on-request pull for MMT Package ingestion for each flow. The unique MMTP session ID, Asset Delivery Characteristic (ADC) which includes bitrate of media (excluding any FEC repair data), scheduling information of MMTP package (start time, stop time) and AL-FEC configuration message are provided as input.

Note AL-FEC message or configuration may be provided from MMT sending entity.

— After MMTP flow creation, the MANE returns a flow ID, which uniquely identifies the created flow. Additionally, in case of a MMTP live flow, the push MMT-URL is added to the response.

— Once the flow is created, all information for signalling message is available and the MANE is ready to start announcing the MMTP session. The MANE starts providing Package-relatedsignalling message automatically as soon as all the necessary MMTP session information have been provided (e.g., PA message for MMT Package). MMTP session signalling message is automatically updated following MMTP flow updates.

Note A MMTP session may hold one or multiple flows of different types of Asset delivery. (e.g., MPU mode, GFD mode for MMT Package delivery).

— At MMTP flow start time, the IP-based connection is activated between MANE and MMT receiving entity and the MANE issues the required necessary information for MMT signalling messages to the MMT receiving entity.

— The MMT content starts flowing through the MMTP flows

— At MMTP flow stop time, the IP-based connection is terminated.

Note In case of regional services, i.e. that delivers region-specific content, a MMTP session can be cloned so that all flows of the MMTP session use the same delivery methods parameters.

The IP-based connection is automatically provisioned by the MANE and is active between start and stop time independently whether the MMT sending entity is sending media stream and its related signaling message. The MANE automatically terminates the IP-based connection at stop time. The MMT sending entity may proactively terminate the MMTP flow before the stop time.

14.3.6 MMTP session update

A part or all possible information may be changed since a MMTP session is created. In that case, it results to need of dynamic modification of MMT signaling message by MMT sending entity. Figure 32 depicts MMTP session update procedure as follows:

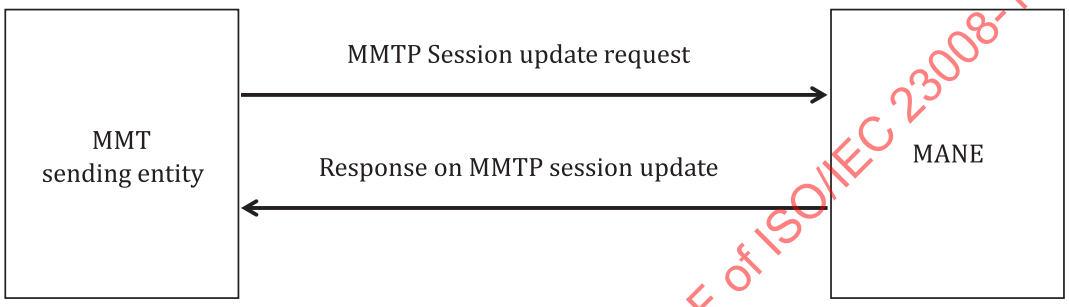


Figure 32 — Procedure of MMTP session update between MMT sending entity and MANE

— The authenticated and authorized MMT sending entity updates an existing MMTP session providing the unique MMTP session ID, and updated parameters. As long as no flows are added to the MMTP session, all information of MMTP session may be modified. Once a MMTP flow is created, only the MMT receiving entity capabilities and Measurement Configuration messages are allowed to be modified.

— The MANE shall provide an update status of MMTP session to MMT sending entity.

Note that the MMT sending entity should first request the MMTP session information using an Measurement Configuration message before performing an any update as the MANE expects the Update MMTP session message to contain all MMTP session information.

14.3.7 MMTP flow update

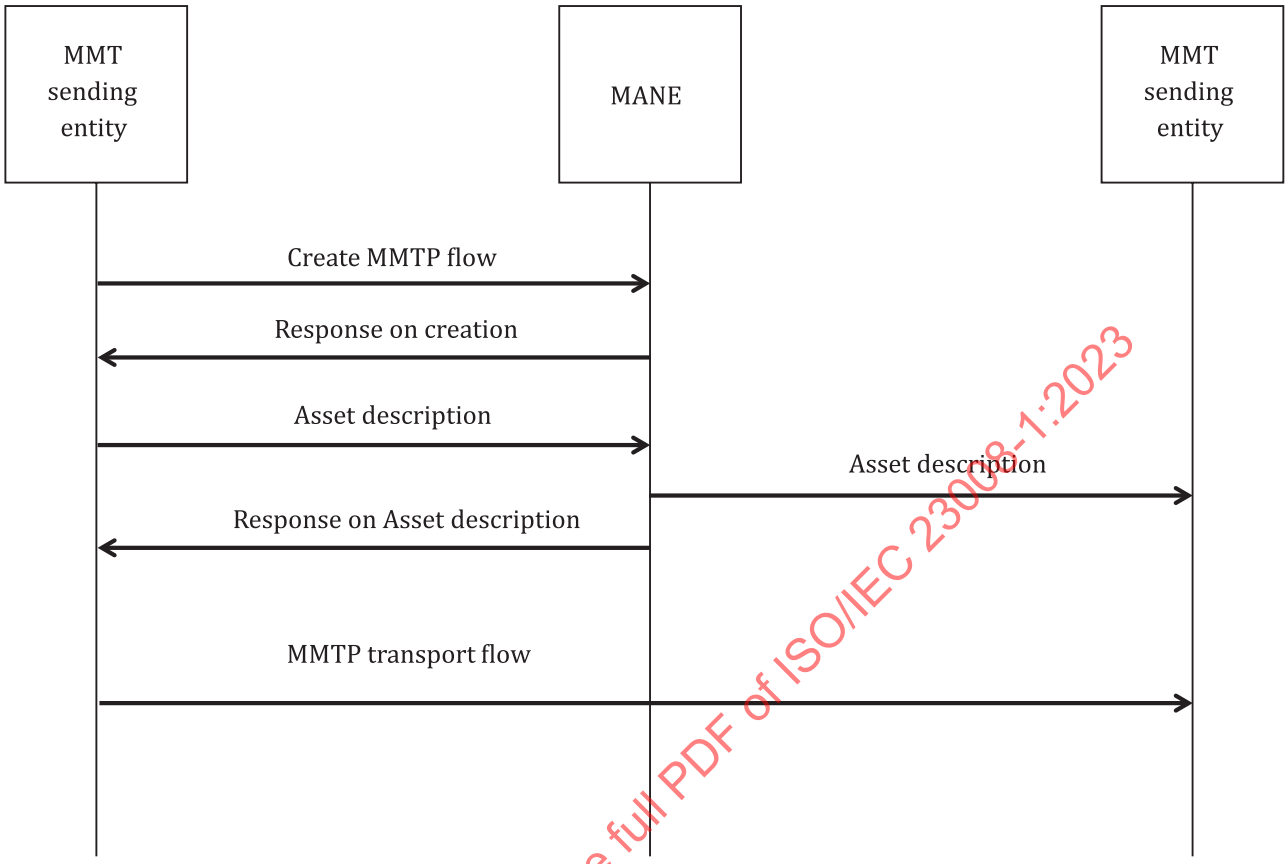


Figure 33 — Procedure of MMTP flow update between MMT sending entity and MANE

Figure 33 depicts MMTP flow update procedure as follows:

— The authenticated and authorized MMTP sending entity may update an existing MMTP flow. Before the MMTP flow is started, all information of MMTP flow can be updated. Once the MMTP flow is started, only the following information are allowed tobe changed:

  — MMTP flow schedule (start, stop),

  — MMTP flow AL-FEC information,

  — number of MMTP flow (add, delete).

— The MANE shall provide an any update status of MMTP flow to MMT sending entity.

— New MMT signalling message may be initiated or updated upon MMTP flow update.

Note that the MMT sending entity should first request the MMTP flow information using Measurement Configuration before performing an update as the MANE expects the update message to contain all information of MMTP flow.

14.3.8 Termination of MMTP session

Figure 34 depicts MMTP session termination procedure as follows:

— The authenticated and authorized MMT sending entity may terminate an existing MMTP session. After that, any MMTP flow is terminated andMMTP session provisioning information is removed from the MANE.

— The MANE shall provide a deletion status and update package information to MMT receiving entity.

— Deleted MMTP session is removed from the MMT signaling message

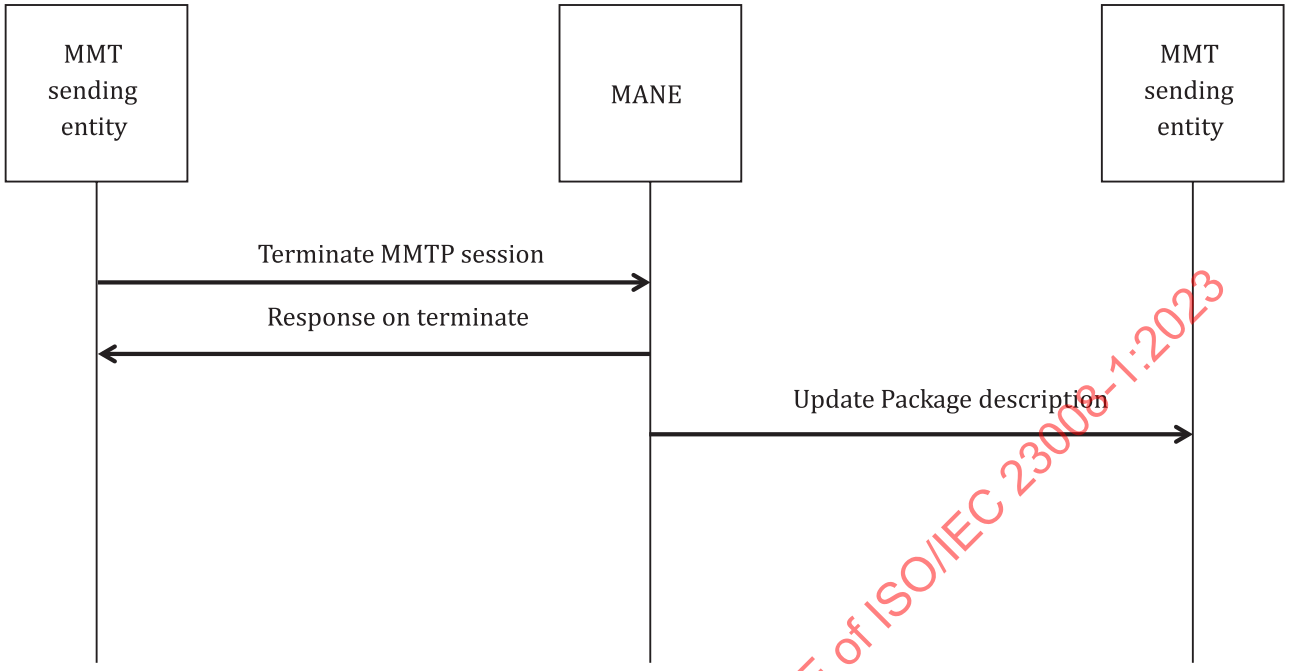


Figure 34 — Procedure of MMTP session termination between MMT sending entity and MANE

14.3.9 Termination of MMTP flow

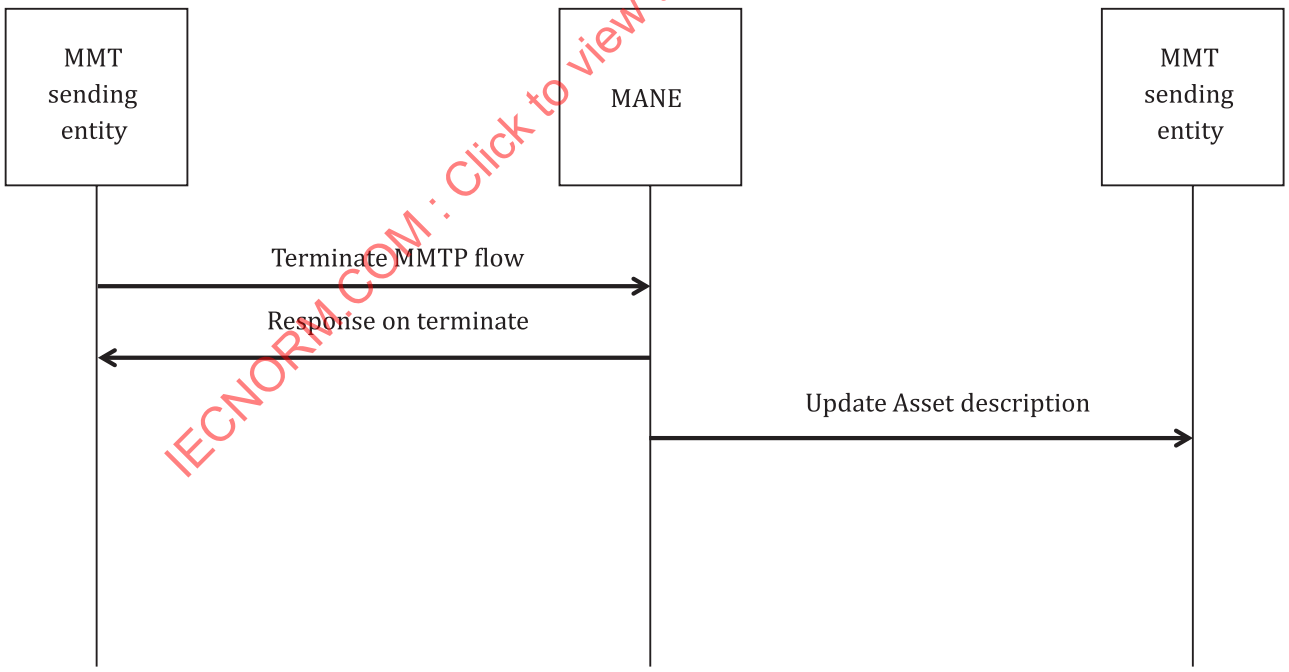


Figure 35 — Procedure of MMTP flow termination between MMT sending entity and MANE

Figure 35 depicts MMTP flow termination procedure as follows:

— The authenticated and authorized MMT sending entity may delete an existing MMTP flow. If the MMTP flow is deleted, connection for signaling message and delivery of MMT Package are

terminated. Then, MMTP flow provisioning information is removed from the MANEand allocated resources in MANE are released.

— The MANE shall provide a deletion status and update package information to MMT receiving entity.

— Deleted MMTP flow is removed from the MMT signaling message for MMT receiving entity

14.3.10 MMTP session query to MANE

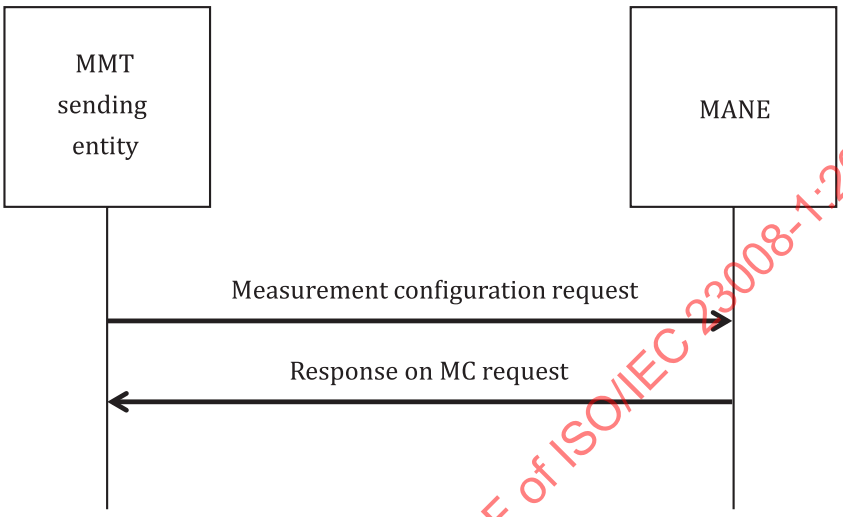


Figure 36 — Procedure of MMTP session query to MANE

Figure 36 depicts MMTP session query procedure as follows:

— The authenticated and authorized MMT sending entity may request to the MANE for specific information by sending measurement configuration message.

— The MANE shall provide the information response upon receiving the measurement configuration.

14.3.11 MMTP session measurement feedback

Figure 37 depicts MMTP session measurement feedback procedure as follows:

— The authenticated and authorized MMT sending entity may request to register for feedback on the MANE. The MANE shall provide the scope of the feedback (for all MMTP session, or for a specific MMTP session or MMTP flow) and the URL to be used by the MANE to report the feedback.

— The MANE shall response proper registration request.

— If a measured condition in the measurement configuration defined by the MMT sending entity occurs, the MANE shall send an feedback message to the MMT sending entity or report to 3rd party server with the payload of the message containing the feedback message.

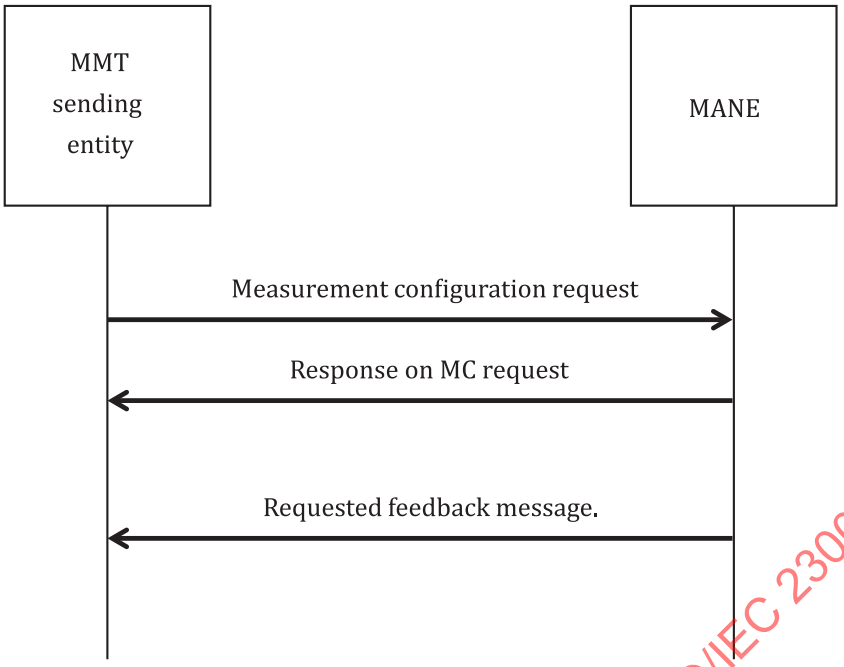


Figure 37 — Procedure on MMTP session measurement feedback

15 FCAST support in MMT

15.1 General

FCAST[28] can be considered as a protocol extension to deliver objects on top of other transport protocols such as ALC/LCT and NORM. Depending on the underlying protocol, FCAST can offer a different range of reliability. With NORM for instance, full reliability can be achieved using an acknowledged delivery mode. However, this comes at the cost of scalability. FCAST is intented to be a simple alternative to a fully-fledged file delivery protocol like FLUTE by sacrificing some flexibility.

FCAST delivers compound objects, including both metadata and actual data of an object together. This simplifies the reception procedure considerably. The object metadata is provided in the form of HTTP 1.1 headers, but other formats can be used and signalled as well.

It comes with an integrity check mechanism using a checksum that can verify correctness of the header or of the full compound object. To avoid large overhead, the object header may be compressed using Gzip.

FCAST comes with special support for object delivery carousels. An FCAST session can carry an object carousel, where a fixed set of files is sent repeatedly over the session. The carousel is described in a dedicated compound object and identified by a carousel instance ID (CID).

MMT comes with support for generic file delivery using the GFD payload format. In terms of functionality, GFD and FCAST are equivalent. However, for deployments that prefer to leverage their existing file delivery procedures instead of migrating them to GFD, the co-existence of MMT with FCAST integrated for a combined delivery solution would offer a good alternative.

15.2 MMT signalling of resources delivered using FCAST

15.2.1 General

Support for FCAST as an alternative to in-band GFD mode delivery is provided through signalling of media resources in MMT and indicating that they are delivered using FCAST. A new location type is introduced to support referencing FCAST transport objects as defined in Table 111.

Note that only FCAST based on ALC/LCT is supported. NORM is not supported in conjunction with MMT.

15.2.2 Syntax of FCAST location type

Table 111 — FCAST location type syntax

Syntax	Value	No. of bits	Mnemonic
…			
if (location_type == '0x0D') {			
ipv4_address_type		1	b
toi_flag		1	b
url_flag		1	b
carousel_flag		1	b
reserved	'1111'	4	uimsbf
if (ipv4_address_type == '1b') {			
dst_ipv4_addr		32	uimsbf
} else {			
dst_ipv6_addr		128	uimsbf
}			
port_number		16	uimsbf
if (toi_flag) {			
S		1	b
H		1	b
O		1	b
reserved	'1 1111'	5	uimsbf
TSI		32*S+16*H	uimsbf
TOI		32*O+16*H	uimsbf
}			
if (url_flag) {			
URL_length	N1	32	uimsbf
for (i=0; i<N1; i++) {			
URL_byte		8	uimsbf
}			
}			
if (carousel_flag) {			
CID			
}		32	uimsbf

15.2.3 Semantics

The semantics of the fields are as follows:

`ipv4_address_type`: the type of the IP address of the destination;

`toi_flag`: a flag that indicates if the TOI of the referenced object is known and provided here;

`url_flag`: a flag that indicates if the URL of the referenced object is provided;

`carousel_flag`: a flag that indicates that the object is delivered as part of a carousel and the identifier of that carousel is given here;

`dst_ipv4_addr`: the IPv4 address of the destination on which the FCAST session is running;

`dst_ipv6_addr`: the IPv6 address of the destination on which the FCAST session is running;

`S`: a flag that indicates the number of 4-bytes in the TSI;

`H`: a flag that indicates the number of 2-bytes in the TSI and TOI;

`O`: a flag that indicates the number of 4-bytes in the TOI;

`URL_length`: the length of the URL of the FCAST object;

`URL_byte`: a byte from the URL of the FCAST object, encoded in UTF-8 format;

`CID`: numeric value that identifies a carousel instance.

15.3 FCAST over MMTP

15.3.1 MMTP packet header for FCAST

Among the syntax elements of the MMTP packet header, a new value for the `type` field is needed to support FCAST. To indicate FCAST payload type, a new value of the “reserved for ISO use” code space shall be assigned for FCAST in the “data type and definition of data unit” table.

15.3.2 MMTP payload header for FCAST mode

The payload packets sent using MMTP shall include an FCAST payload header and an FCAST payload as shown in Figure 38. A FCAST payload can be a whole compound object or a member of a partition of a compound object.

An imminent end of FCAST carousel session can be signalled by the FCAST payload header. The actual termination of an FCAST carousel session is indicated by removing the corresponding asset entry from the MPT.

An imminent removal of a compound object out of a carousel instance can be signalled by the FCAST payload header. The actual removal of a compound object is indicated by removing the corresponding TOI from the CID.

To allow compound object multiplexing, a payload sequence number shall be provided in the FCAST payload header. The payload sequence number can be used to sort out payloads that belong to a TOI. Also, when a receiver joins an FCAST session in the middle of a long object, it can help early download of the object by downloading its latter parts from the current cycle and downloading the rest in the next cycle.

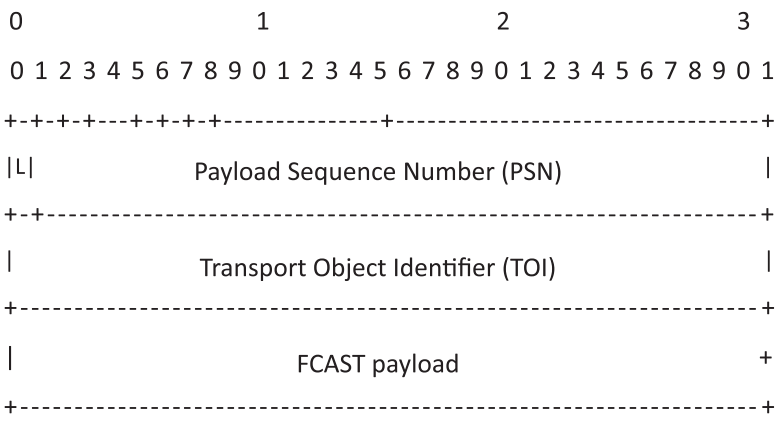


Figure 38 — MMTP payload header for FCAST mode

As shown in Figure 38, the FCAST payload header has a fixed size of 8 bytes.

`L` (`1 bit`) – indicates that this packet contains the last byte of the object.

`Payload Sequence Number (PSN: 31 bits)` – is a number incremented per FCAST payload with the same TOI. It starts from 0 and a wrap-around is not allowed. For a long compound object that requires a payload sequence number beyond its maximum value, a partitioning method shall be used. FCAST protocol provides a means to partition a long compound object into multiple compound objects.

`Transport Object Identifier (TOI: 32 bits)` – indicates that this packet contains the last byte of the object.

15.3.3 Signalling in MPT for FCAST

A FCAST session corresponds to an FCAST carousel asset in an MMT package. The MPT contains the signalling information for all the FCAST carousel assets in an MMT package. A FCAST carousel asset is identified by its `asset_id` in MPT. To indicate the `asset_type` of an FCAST carousel asset, a new four character code (4CC) 'fcst' shall be used.

A FCAST descriptor defined in subclause 15.3.4 shall be included in the `asset_descriptors` loop in MPT. An FCAST descriptor carries the signalling for the current FCAST carousel instance.

15.3.4 FCAST descriptor

The syntax of the `FCAST_descriptor()` is defined in Table 112 and the semantics are provided below the table.

Table 112 — FCAST descriptor

Syntax	Value	No. of bits	Mnemonic
`FCAST_descriptor () {`			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**8**	**uimsbf**
descriptor_version		**3**	**bslbf**
static_carousel_flag		**1**	**bslbf**
fixed_payload_size_flag		**1**	**bslbf**
cycle_independent_payload_flag		**1**	**bslbf**
use_CID_flag		**1**	**bslbf**
use_metadata_collection_flag		**1**	**bslbf**
`if (use_CID_flag)`			

Table 112 *(continued)*

Syntax	Value	No. of bits	Mnemonic
CID_TOI		**32**	**uimsbf**
`if (use_CID_flag)`			
metadata_collection_TOI		**32**	**uimsbf**
`}`			

`descriptor_tag` – a tag value indicating the type of a descriptor. (A new value shall be assigned for this descriptor.)

`descriptor_length` – specifies the length in bytes counting from the next byte after this field to the last byte of the descriptor.

`descriptor_version` – specifies the version of this FCAST descriptor. This field shall be incremented whenever the FCAST descriptor is updated. This field is a wrap-around counter. If this field has a value different from that of the previously saved FCAST descriptor, receivers shall replace it with this FCAST descriptor.

`static_carousel_flag` – indicates a static carousel. If this flag is set to 1, the FCAST carousel is static.

`fixed_payload_size_flag` – If this flag is set to 1, all MMTP packets with a same TOI in the FCAST header has equal amount of FCAST payload except for the last MMTP packet.

`cycle_independent_payload_flag` – If this flag is set to 1, the way a compound object is partitioned into the FCAST payload fields of MMTP packets is the same throughout the FCAST carousel. In this case, the MMTP packets with a same TOI or an equivalent TOI and with a same payload sequence number (PSN) have the same part of a compound object as FCAST payload.

`use_CID_flag` – If this flag is set to 1, a CID is delivered in the current FCAST instance.

`use_metadata_collection_flag` – If this flag is set to 1, a metadata collection object is delivered as a regular compound object in the current FCAST instance. A metadata collection object contains all or part of object metadata for all the objects delivered in the current FCAST instance.

`CID_TOI` – specifies the TOI of the CID compound object delivered in the current FCAST instance.

`metadata_collection_TOI` – specifies the TOI of the metadata collection compound object delivered in the current FCAST instance.

15.3.5 Metadata collection object

If not compressed, a metadata collection object is a text file encoded in UTF-8.

A metadata collection object contains a list of `{(1stCID, TOI), object metadata}` where `1stCID` is the CID of the first FCAST carousel instance to which the object is added and `TOI` is the TOI value of the object in the FCAST carousel instance that has `1stCID` as its CID. For static carousel, `1stCID` shall not be provided.

The `object metadata` provided in a metadata collection object may be only a part of the original object metadata.

For each entry `{(1stCID, TOI), object metadata}` in a metadata collection object, the following format shall be used:

CID: the CID value in a decimal number<CR-LF>

TOI: the TOI value in a decimal number<CR-LF>

A part or all of the object metadata in HTTP/1.1 metainformation format

Annex A
(informative)

Jitter calculation in MMTP

A.1 General

MMT specifies the timing model to be used for MMTP packets delivery. Delivery of timed media data according to their temporal requirements is an important feature supported by the MMT protocol. Preservation of timing relationships among packets in a single MMTP packet flow or between packets from different MMTP packet flows is another important feature of MMT. The delivery timing model provides also the necessary information to calculate jitter and the amount of delay introduced by the underlying delivery network during the delivery of a Package.

A.2 Network jitter calculation

Network jitter calculation is essential for jitter compensation that may be required by some services using the MMT protocol. Network jitter calculation method used in this document is adopted from IETF RFC 3550. Formula (A.1) is used to calculate the difference in packet spacing, $D_{MMT}(i,j)$, for a pair of MMTP packets i and j at the receiving entity:

$$D_{MMT}(i,j)=(T_{A,j}-T_{A,i})-(T_{D,j}-T_{D,i})=(T_{A,j}-T_{D,j})-(T_{A,i}-T_{D,i}) \quad \text{(A.1)}$$

where $T_{D,i}$ and $T_{D,j}$ denote the delivery time instance of two MMTP packets i and j, respectively, carried in an MMTP packet header. The $T_{A,i}$ and $T_{A,j}$ are the time instances at which MMTP packets i and j have arrived at the MMT receiving entity, respectively.

The inter-arrival jitter, $J_{MMT}(i)$, which is defined to be the mean deviation of the difference in packet spacing is calculated continuously according to the following formula every time an MMTP packet i is received:

$$J_{MMT}(i)=J_{MMT}(i-1)+[D_{MMT}(i-1,i)-J_{MMT}(i-1)]/16 \quad \text{(A.2)}$$

Annex B
(normative)

XML syntax and MIME type for signalling message

B.1 XML syntax

The following provides the XML syntax for the signalling messages that are defined in this document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="urn:mpeg:MMT:schema:Signalling:2013"
targetNamespace="urn:mpeg:MMT:schema:Signalling:2013" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:annotation>
      <xs:appinfo>MMT Signalling</xs:appinfo>
      <xs:documentation xml:lang="en">This schema defines the syntax for MMT Signalling
messaging that </xs:documentation>
   </xs:annotation>
   <xs:element name="Message">
      <xs:complexType>
         <xs:choice minOccurs="1" maxOccurs="1">
            <xs:element name="PA_message">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="table" type="Table" minOccurs="1"
maxOccurs="unbounded"/>
                     <xs:any namespace="##other" processContents="skip"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="MPI_message">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="MPI_table" type="MPIT"/>
                     <xs:element name="MP_table" type="MPT" minOccurs="0"/>
                     <xs:any namespace="##other" processContents="skip"/>
                  </xs:sequence>
                  <xs:attribute name="associated_MP_table_flag" type="xs:boolean"
use="optional" default="false"/>
                  <xs:anyAttribute namespace="##other" processContents="skip"/>
               </xs:complexType>
            </xs:element>
            <xs:element name="MPT_message">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="MP_table" type="MPT"/>
                     <xs:any namespace="##other" processContents="skip"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="CRI_message">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="CRI_table" type="CRIT"/>
                     <xs:any namespace="##other" processContents="skip"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="DCI_message">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="DCI_table" type="DCIT"/>
```

```
                        <xs:any namespace="##other" processContents="skip"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element name="AL_FEC_message">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="fec_flow">
                           <xs:complexType>
                              <xs:sequence>
                                 <xs:element name="FECConfiguration">
                                    <xs:complexType>
                                       <xs:attribute name="repair_flow_id"
type="xs:integer"/>
                                       <xs:attribute name="maximum_k_for_repair_flow"
type="xs:integer"/>
                                       <xs:attribute name="maximum_p_for_repair_flow"
type="xs:integer"/>
                                       <xs:attribute name="protection_window_time"
type="xs:integer"/>
                                       <xs:attribute name="protection_window_size"
type="xs:integer"/>
                                       <xs:attribute name="num_of_layer_for_LAFEC"
type="xs:integer"/>
                                       <xs:attribute name="fec_code_id_for_repair_flow"
type="xs:integer"/>
                                       <xs:anyAttribute namespace="##other"
processContents="skip"/>
                                    </xs:complexType>
                                 </xs:element>
                                 <xs:any namespace="##other" processContents="skip"/>
                              </xs:sequence>
                              <xs:attribute name="fec_flow_id" type="xs:integer"/>
                              <xs:attribute name="source_flow_id" type="xs:integer"/>
                              <xs:attribute name="packet_ids" type="ListT" />
                              <xs:attribute name="fec_coding_structure" type="xs:integer"/>
                              <xs:attribute name="ssbg_mode" type="xs:integer"/>
                              <xs:attribute name="ffsrpts_flag" type="xs:boolean"/>
                              <xs:attribute name="length_of_repair_symbol"
type="xs:integer"/>
                              <xs:anyAttribute namespace="##other" processContents="skip"/>
                           </xs:complexType>
                        </xs:element>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element name="HRBM_message">
                  <xs:complexType>
                     <xs:attribute name="max_buffer_size" type="xs:integer"/>
                     <xs:attribute name="fixed_end_to_end_delay" type="xs:integer"/>
                     <xs:attribute name="max_transmission_delay" type="xs:integer"/>
                     <xs:anyAttribute namespace="##other" processContents="skip"/>
                  </xs:complexType>
               </xs:element>
               <xs:element name="ADC_message">
                  <xs:complexType>
                     <xs:attribute name="ADC_level_flag" type="xs:boolean"/>
                     <xs:attribute name="MPU_sequence_number " type="xs:integer"
minOccurs="0"/>
                     <xs:attribute name="packet_id" type="xs:integer"/>

                     <xs:attribute name="loss_tolerance" type="xs:integer"/>
                     <xs:attribute name="jitter_sensitivity" type="xs:integer"/>
                     <xs:attribute name="class_of_service" type="xs:boolean"/>
                     <xs:attribute name="bidirection_indicator" type="xs:boolean"/>

                     <xs:attribute name="flow_label" type="xs:integer"/>
                     <xs:attribute name="sustainable_rate" type="xs:integer" minOccurs="0"/>
                     <xs:attribute name="buffer_size" type="xs:integer" minOccurs="0"/>
                     <xs:attribute name="peak_rate" type="xs:integer"/>
                     <xs:attribute name="max_MFU_size" type="xs:integer"/>
```

```
                  <xs:attribute name="mfu_period" type="xs:integer"/>
<xs:complexType name="mmt:OperationPointCharacteristics">
      <xs:attribute name="sampleGroupIndex" type="xs:integer"/>
      <xs:attribute name="operationPointSpatialQuality" type="xs:integer"/>
      <xs:attribute name="operationPointTemporalQuality" type="xs:integer"/>
      <xs:attribute name="operationPointBitrate" type="xs:integer"/>
      <xs:anyAttribute processContents="lax">
</xs:complexType>
                  <xs:anyAttribute namespace="##other" processContents="skip"/>
               </xs:complexType>
            </xs:element>
         </xs:choice>
         <xs:attribute name="message_id" type="xs:integer"/>
         <xs:attribute name="version" type="xs:integer"/>
         <xs:anyAttribute namespace="##other" processContents="skip"/>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="Table">
      <xs:attribute name="table_id" type="xs:integer"/>
      <xs:attribute name="version" type="xs:integer"/>
      <xs:anyAttribute namespace="##other" processContents="skip"/>
   </xs:complexType>

   <xs:complexType name="PAT">
      <xs:complexContent>
         <xs:extension base="Table">
            <xs:sequence>
               <xs:element name="location" maxOccurs="unbounded">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="MMT_general_location_info" type="MMT_general_
location_info_type" maxOccurs="2"/>
                        <xs:any namespace="##other" processContents="skip"/>
                     </xs:sequence>
                     <xs:attribute name="alternative_location_flag" type="xs:boolean"/>
                     <xs:attribute name="signalling_information_table_id"
type="xs:integer"/>
                     <xs:attribute name="signalling_information_table_version"
type="xs:integer"/>
                     <xs:anyAttribute namespace="##other" processContents="skip"/>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="MMT_general_location_info_type">
      <xs:choice>
         <xs:element name="asset">
            <xs:complexType>
               <xs:attribute name="packet_id" type="xs:integer"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="flow">
            <xs:complexType>
               <xs:attribute name="ipv4_src_addr" type="xs:string"/>
               <xs:attribute name="ipv4_dst_addr" type="xs:string"/>
               <xs:attribute name="dst_port" type="xs:integer"/>
               <xs:attribute name="packet_id" type="xs:integer"/>
               <xs:anyAttribute namespace="##other" processContents="skip"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="flow_ipv6">
            <xs:complexType>
               <xs:attribute name="ipv6_src_addr" type="xs:string"/>
               <xs:attribute name="ipv6_dst_addr" type="xs:string"/>
               <xs:attribute name="dst_port" type="xs:integer"/>
               <xs:attribute name="packet_id" type="xs:integer"/>
               <xs:anyAttribute namespace="##other" processContents="skip"/>
            </xs:complexType>
         </xs:element>
```

```
      <xs:element name="M2TS">
         <xs:complexType>
            <xs:attribute name="network_id" type="xs:integer"/>
            <xs:attribute name="MPEG_2_transport_stream_id" type="xs:integer"/>
            <xs:attribute name="MPEG_2_PID" type="xs:integer"/>
            <xs:anyAttribute namespace="##other" processContents="skip"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="M2ES">
         <xs:complexType>
            <xs:attribute name="ipv6_src_addr" type="xs:string"/>
            <xs:attribute name="ipv6_dst_addr" type="xs:string"/>
            <xs:attribute name="dst_port" type="xs:integer"/>
            <xs:attribute name="MPEG_2_PID" type="xs:integer"/>
            <xs:anyAttribute namespace="##other" processContents="skip"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:complexType>

<xs:complexType name="MPIT">
   <xs:complexContent>
      <xs:extension base="Table">
         <xs:sequence>
            <xs:element name="PI_fragment">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="fragment">
                        <xs:complexType>
                           <xs:simpleContent>
                              <xs:extension base="xs:string">
                                 <xs:attribute name="mime_type" type="xs:string"/>
                              </xs:extension>
                           </xs:simpleContent>
                        </xs:complexType>
                     </xs:element>
                     <xs:any namespace="##other" processContents="skip"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="skip"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="MPT">
   <xs:complexContent>
      <xs:extension base="Table">
         <xs:sequence>
            <xs:element name="asset" type="AssetT"/>
            <xs:element name="descriptor" type="DescriptorT"/>
            <xs:any namespace="##other" processContents="skip"/>
         </xs:sequence>
         <xs:attribute name="MP_table_mode" type="xs:integer"/>
         <xs:attribute name="MMT_package_id" type="xs:string"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="CRIT">
   <xs:complexContent>
      <xs:extension base="Table">
         <xs:sequence>
            <xs:element name="CRI_descriptor" type="CRIDT"/>
            <xs:any namespace="##other" processContents="skip"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

```
   <xs:complexType name="DCIT">
      <xs:complexContent>
         <xs:extension base="Table">
            <xs:sequence>
               <xs:element name="asset">
                  <xs:complexType>
                     <xs:choice>
                        <xs:element name="video">
                           <xs:complexType>
                              <xs:attribute name="video_average_bitrate"
type="xs:integer"/>
                              <xs:attribute name="video_maximum_bitrate"
type="xs:integer"/>
                              <xs:attribute name="horizontal_resolution"
type="xs:integer"/>
                              <xs:attribute name="vertical_resolution" type="xs:integer"/>
                              <xs:attribute name="temporal_resolution" type="xs:integer"/>
                              <xs:attribute name="video_minimum_buffer_size"
type="xs:integer"/>
                              <xs:anyAttribute namespace="##other"
processContents="skip"/>
                           </xs:complexType>
                        </xs:element>
                        <xs:element name="audio">
                           <xs:complexType>
                              <xs:attribute name="audio_average_bitrate"
type="xs:integer"/>
                              <xs:attribute name="audio_maximum_bitrate"
type="xs:integer"/>
                              <xs:attribute name="audio_minimum_buffer_size"
type="xs:integer"/>
                              <xs:anyAttribute namespace="##other"
processContents="skip"/>
                           </xs:complexType>
                        </xs:element>
                        <xs:element name="download">
                           <xs:complexType>
                              <xs:attribute name="required_storage" type="xs:integer"/>
                              <xs:anyAttribute namespace="##other"
processContents="skip"/>
                           </xs:complexType>
                        </xs:element>
                     </xs:choice>
                     <xs:attribute name="mime_type" type="xs:string"/>
                     <xs:anyAttribute namespace="##other" processContents="skip"/>
                  </xs:complexType>
               </xs:element>
               <xs:any namespace="##other" processContents="skip"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="AssetT">
                        <xs:sequence>
                           <xs:element name="asset_clock_relation">
                              <xs:complexType>
                                 <xs:attribute name="asset_clock_relation_id"/>
                                 <xs:attribute name="asset_timescale"/>
                              </xs:complexType>
                           </xs:element>
                           <xs:element name="MMT_general_location_info" type="MMT_general_
location_info_type"/>
                           <xs:element name="asset_id" type="asset_id_T"/>
                           <xs:any namespace="##other" processContents="skip"/>
                        </xs:sequence>
                        <xs:attribute name="mime_type" type="xs:string"/>
                        <xs:attribute name="default_asset_flag" type="xs:boolean"/>
                        <xs:anyAttribute namespace="##other" processContents="skip"/>
                     </xs:complexType>
```

```
   <xs:complexType name="DescriptorT">
      <xs:attribute name="descriptor_tag" type="xs:integer"/>
   </xs:complexType>

   <xs:complexType name="CRIDT">
      <xs:complexContent>
         <xs:extension base="DescriptorT">
            <xs:attribute name="clock_relation_id" type="xs:integer"/>
            <xs:attribute name="STC_sample" type="xs:integer"/>
            <xs:attribute name="NTP_timestamp_sample" type="xs:dateTime"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="MPUTSD">
      <xs:complexContent>
         <xs:extension base="DescriptorT">
            <xs:attribute name="mpu_sequence_number" type="xs:integer"/>
            <xs:attribute name="mpu_presentation_time" type="xs:dateTime"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>


   <xs:simpleType name="ListT">
      <xs:list itemType="xs:integer"/>
   </xs:simpleType>

   <xs:complexType name="asset_id_T">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="asset_id_scheme" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <xs:complexType name="Identifier_mapping">
      <xs:choice>
         <xs:element name="asset_id" type="asset_id_T"/>
         <xs:element name="URL" type="xs:anyURI"/>
         <xs:element name="RegEx" type="xs:string"/>
         <xs:element name="RepresentationId" type="xs:string"/>
      </xs:choice>
      <xs:attribute name="identifier_type" type="xs:integer" use="optional" default="0"/>
      <xs:anyAttribute namespace="##other" processContents="skip"/>
   </xs:complexType>
</xs:schema>
```

B.2 MIME type

The type “application/mmt-signalling+xml” may be used for files containing XML formatted MMT messages.

MIME media type name: application

MIME subtype name: mmt-signalling+xml

Required parameters: none

Optional parameters: none

[Ongoing work related to this registration may introduce new optional parameters.]

Encoding considerations: Files are XML formatted with UTF-8 encoding

Security considerations: None

[There is currently no provision for signing or authentication of MMT signalling message.]

Interoperability considerations: The MPEG organization has defined the specification, which defines the XML schema to enable validation of the MMT signalling message files.

Published document: ISO/IEC 23008-1.

Applications which use this media type: Multimedia

Additional information: None

Magic number(s): None

File extension(s): .xml and .msm may both be used.

Person and e-mail address to contact for further information:

Intended usage: COMMON

Change controller: MPEG

Annex C
(normative)

Application layer forward error correction (AL-FEC) framework for MMT

C.1 General

MMT provides an AL-FEC mechanism for reliable delivery in IP network environments. The MMT FEC scheme is described as a building block of the delivery function. After packetization, MMTP packets are passed to the MMT FEC scheme for protection. The MMT FEC scheme returns repair symbols with FEC payload IDs. Then, the repair symbols are delivered by MMT protocol with the MMTP packets. The FEC configuration information provides the identification of FEC encoded flow, the information specified FEC coding structure and FEC code. It is delivered to the MMT receiving entity for FEC operation. The outlined architecture is depicted in Figure C.1.

The MMT sending entity determines the Assets within Packages which require protection and the number of FEC source flows. One or more of the Assets are protected as a single FEC source flow, which consists of MMTP packets carrying one or more Assets. The FEC source flow and its FEC configuration information are passed to the MMT FEC scheme for protection. The MMT FEC scheme uses FEC code(s) to generate repair symbols composing one or more FEC repair flows. They are passed to MMT protocol along with FEC payload IDs. Then, the MMT protocol delivers FEC source and repair packets to the MMT receiving entity. At the MMT receiving entity, the MMT protocol passes the FEC source flow and its associated FEC repair flow(s) to MMT FEC scheme. Then, the MMT FEC scheme returns recovered MMTP packets.

The MMT FEC scheme divides the FEC source flow into source packet blocks and generates source symbol blocks. The MMT FEC scheme then passes source symbol blocks to the FEC code for FEC encoding. Here, FEC encoding means the process to generate repairs symbols from the source symbol block. FEC code algorithms, which may be used to generate repairs symbols from source symbol block, are described in ISO/IEC 23008-10.

The MMT FEC scheme can be operated in two different modes. FEC payload ID mode 0, which is adding a source FEC payload ID to the MMTP packet, and FEC payload ID mode 1, which does not modify MMTP packets. C.3 to C.5 describe the details for FEC payload ID mode 0 and C.6 for FEC payload ID mode 1.