INTERNATIONAL STANDARD ISO/IEC 23008-1:2014
TECHNICAL CORRIGENDUM 1

Published 2015-03-15

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

Part 1: MPEG media transport (MMT)

TECHNICAL CORRIGENDUM 1

Élément introductif — Élément central —

Partie 1: Titre de partie

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 23008-1:2014 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ICS 35.040 **Ref. No. ISO/IEC 23008-1:2014/Cor.1:2015(E)**

1 Scope

The corrigendum fixes an issue with forward compatibility with the signalling messages and corrects several typos and adds some clarifications to the protocol description.

1 Amended clauses

In subclause 6.2, replace
Non-timed media data are stored as metadata items inside 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 item of the `'meta'` box. The entry point to the non-timed media shall be marked as the primary item of the `'meta'` box. (see sub-clause 8.11.4 of 14496-12)

With

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 sub-clause 8.11.4 of 14496-12)
In subclause 7.1.3.1, replace
Each media sample will be assigned to one or more MFUs. Each sample of the MMT hint track will generate one or more MFUs.

With

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.

In subclause 8.1, replace

MMT provides a generic media streaming solution that supports the transport of virtually any media type and codec.
With

MMT provides a generic media streaming solution that supports the transport of different media types and codecs.

In subclause 8.1, replace

The MMTP payload format is defined as a generic payload format for the packetization of the 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 payloads that are to be delivered using an application layer transport protocol supporting delivery of media data. Thus MMTP payload format can be used as a payload format for MMT protocol or for other packet transport protocols, e.g. the Real Time Protocol (RTP). 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.

With

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.

In subclause 8.1,replace

In addition, MMTP delivers streaming support data such Application Layer Forward Error Correction (AL-FEC) repair data and signalling messages.

With

In addition, MMTP enables to deliver streaming support data such as Application Layer Forward Error Correction (AL-FEC) repair data and signalling messages.

In subclause 8.2.1, replace the text of the subclause

With

The MMT protocol is an application layer transport protocol that is designed to efficiently and reliably transport Packages. MMT protocol can be used for both timed and non-timed media data. 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 specification, the carriage of data formatted in a format other than the MMTP payload format as specified in 8.3 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. 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.MMT protocol defines two packetization modes, Generic File Delivery (GFD) mode as specified in 8.3.3 and MPU mode as specified in 8.3.2. 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. 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.

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.

In subclause 8.2.2 replace

Figure 7 illustrates the structure of an MMTP packet.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=0|C|FEC|r|X|R|RES|  type     |         packet_id             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    packet_sequence_number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          packet_counter                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       header_extension                  ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         payload data                    ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    source_FEC_payload_ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1 — Structure of MMTP packet

With

Figure 7 illustrates the structure of an MMTP packet.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=0|C|FEC|r|X|R|RES|  type     |         packet_id             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    packet_sequence_number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          packet_counter                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       header_extension                  ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         payload data                    ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    source_FEC_payload_ID                ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2 — Structure of MMTP packet

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.

In subclause 8.2.3, replace

`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. The RAP_flag shall be set to mark data units of MPU Fragment Type value 0 and 1 and for MFUs that contain a sync sample or a fragment thereof, in the case of timed media, and for the primary item of non-timed MPUs.

With

`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.

In subclause 8.2.3, replace

`packet_counter (32 bits)` – an integer value for counting MMTP packets. It is incremented by 1 when an MMTP packet is sent regardless of it's `packet_id` value. This field starts from arbitrary value and wraps around to `'0'` after its maximum value is reached.

With

`packet_counter (32 bits)` – an integer value for counting MMTP packets. It is incremented by 1 when an MMTP packet is sent regardless of it's `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.

In subclause 8.2.2.1, replace

Each file delivered using the GFD mode may also have associated content speicific 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.

With

Each file delivered using the GFD mode may also have associated content specifc 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 subclause 8.4.3.3, replace

3. The MMT receiving entity should process the remainder of the payload, including interpreting the other payload header fields appropriately, and using the source_offset and the payload data to reconstruct the corresponding object as follows:

With

3. 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:

In subclause 8.4.3.3, replace

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

with

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.

In section 9.3.9.2, replace the table:

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 == SUBSET_0_MPT_TABLE_ID) {			
MMT_package_id {	N1		
MMT_package_id_length		**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
}			
MP_table_descriptors {	N2		
MP_table_descriptors_length		**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 111’	**7**	**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**
}			
}			
asset_location {			
location_count	N6	**8**	**uimsbf**
for (i=0; i<N6; i++) {			
MMT_general_location_info()			
}			
}			
asset_descriptors {			
asset_descriptors_length	N5	**16**	**uimsbf**
for (j=0; j<N5; j++) {			
asset_descriptors_byte		**8**	**uimsbf**
}			
}			
}			
}			

with:

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 == SUBSET_0_MPT_TABLE_ID) {			
MMT_package_id {	N1		
MMT_package_id_length		**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
}			
MP_table_descriptors {	N2		
MP_table_descriptors_length		**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’	**6**	**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**
}			
}			
asset_location {			
location_count	N6	**8**	**uimsbf**
for (i=0; i<N6; i++) {			
MMT_general_location_info()			
}			
}			
asset_descriptors {			
asset_descriptors_length	N5	**16**	**uimsbf**
for (j=0; j<N5; j++) {			
asset_descriptors_byte		**8**	**uimsbf**
}			
}			

} }			

In section 9.3.9.3, add the following:

`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.

In section 9.6.1.3, replace

Table 1 — 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	A program within an MPEG-2 TS in a broadcast network. The program is indicated by a PMT PID is described in ISO/IEC 13818-1.
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