INTERNATIONAL STANDARD

ISO/IEC 14496-1

Second edition
2001-10-01

AMENDMENT 3
2004-05-01

Information technology — Coding of audio-visual objects —

Part 1: **Systems**

AMENDMENT 3: Intellectual Property Management and Protection (IPMP) extensions

Technologies de l'information — Codage des objets audiovisuels —

Partie 1: Systèmes

AMENDEMENT 3: Gestion et extensions de protection de la propriété intellectuelle (IPMP)

Reference number
ISO/IEC 14496-1:2001/Amd.3:2004(E)

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Amendment 3 to ISO/IEC 14496-1:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Introduction

This document specifies IPMP extensions to the currently specified IPMP specification (the "Hooks"), documented in ISO/IEC 14496-1:2001.

"Hooks" Terminals do not support the additional functionality specified here. Hence, "Hooks" Terminals will not be able to fully process "Extensions" content. However, the syntax and semantics here are compliant with the "Hooks" framework, and therefore allow for graceful failure of a well-designed Terminals in either the event that they are compliant to non-extended IPMP only and receive extended IPMP conforming content, that they are compliant to extended IPMP as outlined in this document and able to process either non-extended conforming content or extended conforming content to the extent that their implementations allow.

Additionally, although this amendment does not preclude the use of "Hooks" and "Extensions" tools being used in the same MPEG-4 presentation, the behaviour of both "Hooks" as well as "Extensions" being used for the protection of the same stream is undefined.

This Amendment specifies

- Extensions to the OD framework in a backward compatible manner to support the use of IPMP in the protection of OD and scene description streams.
- Extensions to the OD framework to support the Identification of required IPMP tools using either 128 bit registered Ids or parametric descriptions.
- Extensions to IPMP_DescriptorPointer in a backward compatible manner to support the extended addressing of IPMP streams.
- Extensions to IPMP_Descriptor in a backward compatible manner to support the use of 16 bit identifiers as well as supporting the identification of the location within a given stream where the specified IPMP tool is to be placed as well as supporting the sequencing of multiple tools at the same location.
- Extensions to IPMP_Descriptor in a backward compatible manner to support the carriage of normative IPMP information.
- Extensions to IPMP_Message in a backward compatible manner to support the specific addressing of a given IPMP_Message to specific IPMP tools.
- Extensions to IPMP_Message in a backward compatible manner to support the carriage of normative IPMP information.
- The definition, as well as Extension tags, syntax and semantics for an IPMP_Data_BaseClass to support the following functionalities.
    - ✧ Mutual Authentication for IPMP tool to IPMP tool as well as IPMP tool to Terminal communication.
    - ✧ The requesting by IPMP tools of the connection/disconnection to requested IPMP tools.
    - ✧ The notification to IPMP tools of the connection/disconnection of IPMP tools.
    - ✧ Common IPMP processing.
    - ✧ IPMP tool to/from User interaction.

- Syntax and semantics for the carriage of IPMP tools in the bit stream.
- Syntax and semantics for IPMP information carriage to and from IPMP tools.
- Syntax and semantics for the requesting and transfer of content and IPMP Tools between Terminals as well as extension tags, syntax and semantics to the IPMP_Data_BaseClass ISO/IEC 14496-1 used therein.
- XML syntax and semantics for the description of the environment in which and MPEG-4 Terminal/application is operating.
- A list of registration authorities required for the support of the amended specifications found herein.

Information technology — Coding of audio-visual objects —

Part 1: **Systems**

AMENDMENT 3: Intellectual Property Management and Protection (IPMP) extensions

In Clause 0.6.1.1, replace the text:

*"*The intellectual property management and protection (IPMP) framework for ISO/IEC 14496 content consists of a normative interface that permits an ISO/IEC 14496 terminal to host one or more IPMP Systems. The IPMP
interface consists of IPMP elementary streams and IPMP descriptors. IPMP descriptors are carried as part of an
object descriptor stream. IPMP elementary streams carry time variant IPMP information that can be associated to multiple object descriptors.
The IPMP System itself is a non-normative component that provides intellectual property management and protection functions for the terminal. The IPMP System uses the information carried by the IPMP elementary streams and descriptors to make protected ISO/IEC 14496 content available to the terminal. An application may
choose not to use an IPMP System, thereby offering no management and protection features."

with:

"The intellectual property management and protection (IPMP) framework for ISO/IEC 14496 content consists of a normative interface that permits an ISO/IEC 14496 terminal to host one or more IPMP Systems in the form of monolithic IPMP Systems or modular IPMP Tools. The IPMP interface consists of IPMP elementary streams and IPMP descriptors. IPMP descriptors are carried as part of an object descriptor stream. IPMP elementary streams carry time variant IPMP information that can be associated to multiple object descriptors. The IPMP System, or IPMP Tools themselves are non-normative components that provides intellectual property management and protection functions for the terminal. The IPMP Systems or Tools uses the information carried by the IPMP elementary streams and descriptors to make protected ISO/IEC 14496 content available to the terminal. An application may choose not to use an IPMP System, thereby offering no management and protection features."

In Clause 4.38, replace the text:

"Only the interface to such systems is normatively defined."

with the text:

"The interface to such systems is defined as well as :

- The provision for the identification of IPMP Tools either through the use of a registration authority or through the use of a functional description of the IPMP Tools' capabilities in a parametric fashion.
- Controlling the time of instantiation of IPMP Tools either by the inclusion of references to the required IPMP Tools or at the request of already instantiated IPMP Tools.

- Providing secure messaging between IPMP Tools and the Terminal and between IPMP Tools and the User.
- Notification of the instantiation of IPMP Tools to IPMP Tools requesting such notification.
- Interaction between IPMP Tools, and/or the Terminal and the User.
- The carriage of IPMP Tools within the bitstream.”

In Clause 4, insert the following text alphabetically:

Binary Representation

In the context of an IPMP Tool, this is the format of the implementation of that IPMP Tool, Examples: Platform Dependent Native Code, Java ™ bytecode.

Content

This implies part or whole of an MPEG presentation.

Content Consumption

Any experience of given Content implies consumption of that content. Access, Playback, Denial of Access and Creation of a Copy are all types of content consumption.

Content Stream

This is the incoming content, of MPEG-4 format.

Control Point

A point on a given elementary stream in a Terminal where IPMP Processing on stream data shall be carried out.

IPMP Device

An implemented application that implements an MPEG-4 Terminal supporting the use of MPEG-4 IPMP.

IPMP Information

Information directed to a given IPMP Tool to enable, assist or facilitate its operation.

IPMP System

A monolithic IPMP protection scheme which requires implementation dependant access to protected streams at required Control Points and must provide any intra-communication within an IPMP System on an implementation basis.

IPMP Tool

IPMP tools are modules that perform (one or more) IPMP functions such as authentication, decryption, watermarking, etc. Conceptually the use of one or more IPMP Tools is combined to perform the functionality of an IPMP System. IPMP Tools, as opposed to IPMP Systems, are normatively identified as to which control points they function at as well as are provided normative methods for secure communications both within as well as outside of a given IPMP Tools comprised functional “IPMP System”. An additional difference between IPMP Tools and IPMP Systems is that IPMP Tools, or a combination thereof, may be used for the protection of Object streams.

In this amendment the use of the term "IPMP System" is used in some cases to indicate either an actual IPMP System or a combination of IPMP Tools whose combination provides the functionality of an IPMP System. In cases where the distinction is important the proper respective terms are used.

IPMP Tool Identifier

This refers to the IPMP Tool ID. It identifies a Tool in an unambiguous way, at the presentation level or at a universal level. Two different identifiers are provided to support the differentiation between the use of IPMP Systems and IPMP Tools.

IPMP Tool List

The IPMP Tool List identifies, and enables selection of, the IPMP Tools required to process the Content.

IPMP Tool Manager

The IPMP Tool Manager is a conceptual entity within the Terminal that processes IPMP Tool List(s) and retrieves the Tools that are specified therein.

IPMP Tool Message

A message passed between any combination of IPMP Tool or Terminal.

IPMP Tool Stream

An elementary stream carrying an implementation of an IPMP Tool.

Message Router

A conceptual entity within the Terminal that implements the Terminal-side behavior of the Terminal-Tool interface.

Mutual Authentication

Protocols carried out to determine the proper and correct identity of a communicating entity and to secure the communication channels between communicating entities.

Parametric Configuration

Information that carries task-specific parameter specification in an extensible form.

Parametric Description

Parametrically described tools shall be defined by an SDL declaration that governs a given description, the parametric configuration and other interface message(s) that drive the tool and the behaviour defined for fulfilment of such a description.

Representation Format

The binary format, platform and communication mechanisms applicable to a given implementation of an IPMP Tool or Terminal.

Scope of Protection

Scope of protection refers to the elementary stream and/or object governed by a given IPMP Tool instance.

Terminal

A Terminal is an environment that consumes possibly protected Content in compliance with the usage rules.

User

A hardware, software or human entity that is the initiator and/or target of content consumption.

In Subclause 8.2.1, replace Table 1 with the following:

Table 1 - List of Class Tags for Descriptors

Tag value	Tag name
0x00	Forbidden
0x01	ObjectDescrTag
0x02	InitialObjectDescrTag
0x03	ES_DescrTag
0x04	DecoderConfigDescrTag
0x05	DecSpecificInfoTag
0x06	SLConfigDescrTag
0x07	ContentIdentDescrTag
0x08	SupplContentIdentDescrTag
0x09	IPI_DescrPointerTag
0x0A	IPMP_DescrPointerTag
0x0B	IPMP_DescrTag
0x0C	QoS_DescrTag
0x0D	RegistrationDescrTag
0x0E	ES_ID_IncTag
0x0F	ES_ID_RefTag
0x10	MP4_IOD_Tag
0x11	MP4_OD_Tag
0x12	IPL_DescrPointerRefTag
0x13	ExtendedProfileLevelDescrTag
0x14	profileLevelIndicationIndexDescrTag
0x15-0x3F	Reserved for ISO use
0x40	ContentClassificationDescrTag
0x41	KeyWordDescrTag
0x42	RatingDescrTag
0x43	LanguageDescrTag
0x44	ShortTextualDescrTag
0x45	ExpandedTextualDescrTag
0x46	ContentCreatorNameDescrTag
0x47	ContentCreationDateDescrTag
0x48	OCICreatorNameDescrTag
0x49	OCICreationDateDescrTag
0x4A	SmpteCameraPositionDescrTag
0x4B-0x5F	Reserved for ISO use (OCI extensions)
0x60	IPMP_ToolsListDescrTag
0x61	`IPMP_ToolTag`
0x62-0xBF	Reserved for ISO use
0xC0-0xFE	User private
0xFF	Forbidden

In Clause 8.3, replace the title:

Intellectual Property Management and Protection (IPMP)

with:

Intellectual Property Management and Protection framework (IPMP)

In Subclause 8.3.1, replace the entire text with:

The intellectual property management and protection (IPMP) framework for ISO/IEC 14496 content consists of a normative interface that permits an ISO/IEC 14496 terminal to host one or more IPMP Systems or IPMP Tools. Additionally, the framework contains a secure messaging system usable between IPMP Tools as well as IPMP Tools and the Terminal and IPMP Tools and the User.
An IPMP System or IPMP Tools are non-normative components that provide intellectual property management and protection functions for the terminal.
The IPMP interface consists of IPMP elementary streams and IPMP descriptors. The normative structure of IPMP elementary streams is specified in this Subclause. IPMP descriptors are carried as part of an object descriptor stream and are specified in 8.6.14. The IPMP interface allows applications (or derivative application standards) to build specialized IPMP Systems or IPMP Tools. Alternatively, an application may choose not to use an IPMP System or IPMP Tools, thereby offering no management and protection features. The IPMP System and IPMP Tools use the information carried by the IPMP elementary streams and descriptors to make protected ISO/IEC 14496 content available to the terminal. The detailed semantics and decoding process of the IPMP System or IPMP Tools are not in the scope of ISO/IEC 14496. The usage of the IPMP System/Tools Interface, however, is explained in 8.8, where the usage of the IPMP framework is also explained.

8.3.1.1 IPMP Architecture overview

This clause describes the general IPMP architecture. For detailed IPMP architecture linked to MPEG-4 system, please refer to 8.8.6.

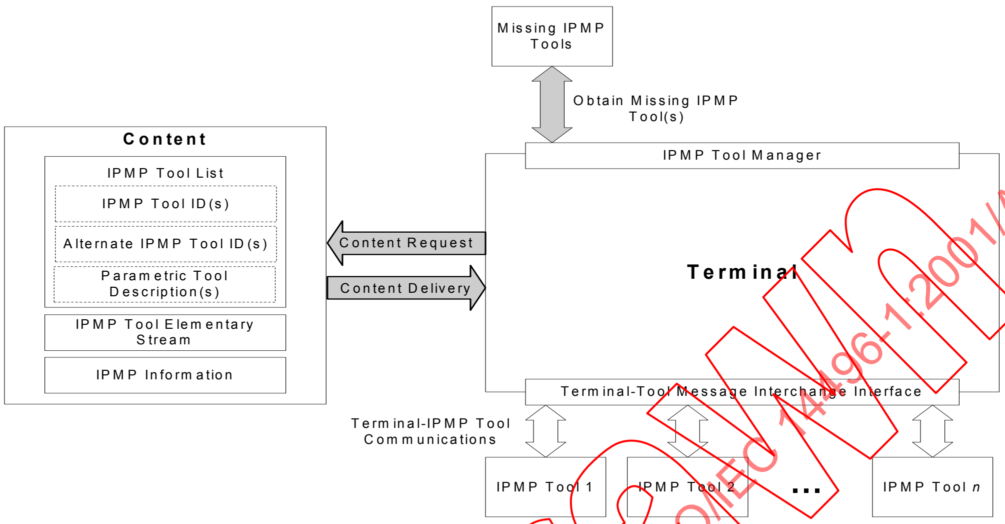

Missing IPMP Tools
Obtain Missing IPMP Tool(s)
Content
IPMP Tool List
IPMP Tool ID(s)
Alternate IPMP Tool ID(s)
Parametric Tool Description(s)
IPMP Tool Elementary Stream
IPMP Information
Content Request
Content Delivery
IPMP Tool Manager
Terminal
Terminal-Tool Message Interchange Interface
Terminal-IPMP Tool Communications
IPMP Tool 1
IPMP Tool 2
...
IPMP Tool n

Figure AMD3-1 — Architecture Diagram for General Concepts

8.3.1.1.1 Messaging

To facilitate the cooperation of multiple tools in the protection and governance of content, a message based architecture is provided. The message based architecture has three advantages over functional interface type architectures. The first is that security can more easily be maintained as messages are less difficult to protect in an open framework then parameters in a function parameters list. The second is that the only entities that need be concerned with a given message's definition are those that need to generate or act upon a given message and so additional functionality can be created and supported simply through the addition of required messages. The third is that full interoperability with IPMP tools can be easily achieved by using the `IPMP_ToolAPI_Config` [8.3.2.12.5] carried in IPMP Descriptor, or by defining a single messaging API by a third-party forum who adopts IPMP.

Physical routing of information and context resolution are handled by a conceptual entity called the Message Router. The Message Router abstracts all platform-dependent routing and delivery issues, from the IPMP Tools. The interface between the Message Router and the Tools, is non-normative and is not defined in this specification, however, the information on the messaging interface can be carried in `IPMP_ToolAPI_Config` to assist interoperability.

All IPMP Tool interactions take place via the Terminal. IPMP Tools do not communicate directly with each other within the scope of this standard.

The delivery of both bit stream sourced IPMP information as well as IPMP tool and Terminal generated information is supported through the use of three separate messages which are passed via the Message Router to IPMP tools. The three messages are, `IPMP_MessageFromBitstream` [8.3.2.12.2], which is used to deliver IPMP stream data, `IPMP_DescriptorFromBitstream` [8.3.2.12.3], which is used to deliver `IPMP_Descriptors` [ISO/IEC 14496-1] and `IPMP_MessageFromTool` [8.3.2.12.4], which is used to deliver messages from either other IPMP tools or the Terminal itself.

In these extensions, a core set of messages are provided which cover what are identified as core functional requirements.

8.3.1.1.2 Mutual Authentication

The most important aspect of a secure messaging architecture is the use of cryptographic algorithms and protocols that allow one to perform a number of important security functions.

At any point in IPMP Information or Content processing, IPMP Tools may be required to communicate with one another or the Terminal. The degree of security required for such communication is determined by a number of variables including information that may be included by the content provider in the Content and conditions of trust established between tool providers a priori and out of band. It is generally the case that a given ES is protected by multiple tools but that certain types of tools are complex (e.g. Rights Management tools) and others are utilities (e.g. Decryption engines). Complex tools may control the instantiation of other tools or make decisions about content use in response to usage queries from the terminal. Mutual authentication may occur between any pair of tools but the level of security required for this communication will in part be dictated by data contained in the bitstream in an opaque manner. The mechanism for making the determination of this security level is non-normative.

Mutual authentication is executed as follows:

1. The Tool that initiates mutual authentication with another tool determines the conditions of trust to be achieved by such authentication, i.e. the initiating tool determines whether it needs integrity protected communication or full secure, authenticated communication. This level may or may not be dictated by IPMP Information in the Content.

2. The communicating tools then engage in a message exchange to determine which authentication protocol will be used. In some cases, this protocol will have been determined by an a priori out of band negotiation between the tool providers in their security audits of one another.

These extensions provide a set of messages to support the identified functionalities. The first message `IPMP_InitAuthentication` [8.3.2.7.1] may be used and delivered to a given IPMP tool such that the receiving IPMP tool is informed as to its required communication partner as well as security measures that must be in place. Following this message or the absence thereof, IPMP tools required to do so will use the `IPMP_MutualAuthentication` [8.3.2.7.2] message as required to determine or create secure channels of communication as needed based on the application.

As one purpose of Mutual Authentication is the verification of trust relationships existing between two entities these specifications provide for the carriage of trust and security metadata. This metadata may include zero or more certificates, credentials or integrity verification information. The creation or establishment of trust relationships are established by out of band relationships between the different entities involved in protecting and managing the content. However, the trust metadata that results from such relationships needs to be made available to permit static and dynamic verification of trust.

During the Mutual Authentication process the carriage of `IPMP_TrustSecurityMetadata` [8.3.2.7.2.7] is supported to provide additional security related data usable by IPMP tools to determine trust related information.

Once Mutual Authentication is performed, the `IPMP_SecureContainer` [8.3.2.7.3] may be used to pass information securely between IPMP tools and IPMP tools and Terminal.

8.3.1.1.3 IPMP tool acquisition

ISO/IEC 14496-1 defines `IPMP_ToolListDescriptor` [8.6.14.2] which conveys the list of IPMP tools required to access the content associated with the `InitialObjectDescriptor` in which it is described, and may include a list of alternate IPMP tools or parametric descriptions of tools required to access the content. The conceptual entity Tool Manager parses the tool list, makes sure all tools are available, and retrieves missing tools if any.

If a given required tool is not present on a given Terminal, the `IPMPToolES_DecoderConfig` [8.3.2.8.3] descriptor can be used to indicate an IPMP tool bearing stream with `IPMP_ToolES_AU` [8.3.2.8.4] being used to actually carry the required tool.

A missing IPMP Tool can also be acquired from a neighbouring IPMP device via tool transfer messages defined in Annex D, or it can be acquired by connecting to a remote server and providing the terminal description as defined in Annex E.

8.3.1.1.4 IPMP Tool connection and disconnection

In the IPMP architecture, IPMP tool may be connected as the result of an `IPMP_DescriptorPointer` [ISO/IEC 14496-1] being processed and in addition may be connected due to requests from already connected IPMP tools. Note: Instantiation of the Tools to be connected is implementation dependent, however, the information on how to instantiate the Tools can be carried in `IPMP_ToolAPI_Config` [8.3.2.12.5] to assist interoperability.

IPMP tools may use the `IPMP_GetTools` [8.3.2.8.1] and `IPMP_GetToolsResponse` [8.3.2.8.2] messages to request a list of tools available for connection that exist as well as a response to the request, respectively.

IPMP tools as well as the Terminal may also query a given IPMP tool as to its capabilities and functionality by using the `IPMP_ToolParamCapabilitiesQuery` [8.3.2.8.5] message with the tool being queried using the `IPMP_ToolParamCapabilitiesResponse` [8.3.2.8.6]message as a reply.

Knowing that a given tool is needed for processing, an IPMP tool may request the connection of another IPMP tool by using the `IPMP_ConnectTool` [8.3.2.8.7] message and may request the disconnection of another IPMP tool by using the `IPMP_DisconnectTool` [8.3.2.8.8] message. A connection may require the actual instantiation of a tool or may be accomplished through physical/electronic means.

The `IPMP_ConnectTool` message contains control point and sequence information to determine the exact location to connect the requested tool. Tools connected at the request of other tools inherit the same scope of protection as the requesting tool. Note that some control points are not associated with any known point on an Elementary Stream. Note also that there are issues with scoping and scenarios that are somewhat illogical, especially as related to BIFS nodes referencing ODs.

Each instantiation of an IPMP Tool shall establish a new logical instance of the tool, for a particular scope of protection. The Terminal assigns a context identifier for the logical instance of the tool, which maps to the specific tool instance, and therefore to the associated scope of protection. These context identifiers shall be unique to ensure unambiguous message addressing.

The process of instantiation involves the following steps

1. Establish a context for the Tool being instantiated
2. Establish a link between the MR and the Tool instance
3. Establish a link between the Tool instance and the MR.

Details of this process are implementation specific. The normative result is a context/address being made available for communication and use of the instantiated tool by other tools as well as the Terminal.

The tool requesting connection shall receive an `IPMP_NotifyToolEvent` [8.3.2.9.3] message indicating the instantiation of the IPMP Tool, and its associated context. The requesting tool and the instantiated tool may perform mutual authentication thereafter.

If an IPMP tool knows of another tool with which it must communicate has already been connected, it may use the `IPMP_GetToolContext` [8.3.2.8.9] message and will receive an `IPMP_GetToolContextResponse` [8.3.2.8.10] message in reply if the requested IPMP tool is already connected with the message containing the address through which the requesting tool may use to communicate with the requested tool.

8.3.1.1.5 Notification of IPMP Tool connection and disconnection

During the processing of IPMP protected content, a number of IPMP tools may be involved, for communication and various security purposes, notification messages are supplied to notify IPMP tools when other IPMP tools have either been connected, disconnected or processed watermark information. Additionally IPMP tools may request at any time a list of all the IPMP tools currently connected at various specifiable scopes of protection.

The IPMP_AddToolNotificationListener [8.3.2.9.1] message may be used to indicate the sending IPMP tools intent to receive notifications of events and scopes of protection as specified in the IPMP_AddToolNotificationListener message. To remove one's self from being notified in the future for specified events, an IPMP tool may use the IPMP_RemoveToolNotificationListener [8.3.2.9.2] to do so.

As events occur for which notifications have been requested, the IPMP_NotifyToolEvent [8.3.2.9.3] message is sent to requesting IPMP tools.

8.3.1.1.6 Common IPMP processing

Direct support has been provided for the carrying out of common IPMP processing operations. Specific support and the related messages are as follows:

- IPMP_CanProcess [8.3.2.10.1] for the notification by an IPMP tool to the Terminal that stream processing may begin. All tools connected and processing data within a given stream must send permission before any tools in the stream receive stream data.
- IPMP_Opaquedata [8.3.2.10.2] for the carriage of user defined data.
- IPMP_KeyData [8.3.2.10.3] for the carriage of decryption key data as well as timing information to determine the validity period of time varying keys.
- IPMP_RightsData [8.3.2.10.4] for the carriage of rights expressions.
- IPMP_SelectiveDecryptionInit **[Error! Reference source not found.]** for the configuration of a decryption tool.
- IPMP_AudioWatermarkingInit **[Error! Reference source not found.]** for the configuration of an audio watermarking tool.
- IPMP_SendAudioWatermark **[Error! Reference source not found.]** for the sending of information to be embedded or that was extracted.
- IPMP_VideoWatermarkingInit **[Error! Reference source not found.]** for the configuration of a video watermarking tool.
- IPMP_SendVideoWatermark **[Error! Reference source not found.]** for the sending of information to be embedded or that was extracted.

8.3.1.1.7 IPMP tool to/from User interaction

During IPMP processing, direct interaction between IPMP tools and a User may be required. The IPMP_UserQuery [8.3.2.11.1] message is used to provide information to be relayed to a User and to request information as well. The IPMP_UserQueryResponse [8.3.2.11.2] is used to relay information provided by a User back to the originator of the User query."

In Subclause 8.3.2.5.1, replace the syntax with the following:

```
aligned(8) expandable(2^28-1) class IPMP_Message
{
   bit(16)  IPMPS_Type;
   if (IPMPS_Type == 0)
   (
      bit(8) URLString[sizeOfInstance-2];
   )
   else (if (IPMPS_Type == 0xFFFF)
   (
      bit(16) IPMP_DescriptorIDEx;
         IPMP_Data_BaseClass IPMP_ExtendedData[]
    } else {
         bit(8) IPMP_data[sizeOfInstance-2];
    }
}
```

In Subclause 8.3.2.5.2, replace the semantics with the following:

The `IPMP_Message` conveys time-varying IPMP information for associated IPMP Tool instances.

`IPMPS_Type` – The type of the IPMP System, in "Hooks" compliant Terminals as specified in ISO/IEC 14496-1:2001. The values "0x0002" to "0x2000" are reserved for future ISO use. A Registration Authority, as designated by ISO/IEC JTC 1, shall assign a unique valid value for this field for a specific IPMP System Type. If the `IPMP_DescriptorID` is "0", another URL is referenced. This process continues until an `IPMP_Message` with a non-zero `IPMP_DescriptorID` is accessed.

`URLString[]` - contains a UTF-8 [6] encoded URL that shall point to the location of a remote `IPMP_Message`.

`IPMP_DescriptorID` – this is one of the `IPMP_DescriptorIDs` in the scope of service of this IPMP Stream and identifies the recipient(s) of the `IPMP_Message`.

`IPMP_ExtendedData` - The IPMP data that is extended from `IPMP_Data_BaseClass` to be delivered to the IPMP tool.

`IPMP_data` - opaque data to be delivered to the IPMP Tool.

The `IPMP_Message` is backward compatible with the `IPMP_Message` of ISO/IEC 14496-1: 2001. However, in order to unambiguously identify the version of the IPMP stream, the `ObjectTypeIndication` shall be set to "0x02" for streams complying with this part of the specification. IPMP Streams complying with ISO/IEC 14496-1:2001 shall use an `ObjectTypeIndication` of "0xFF" as specified for in 8.6.6.2.

Insert the following after Subclause 8.3.2.5.2:

8.3.2.6 Extension tags for the IPMP_Data_BaseClass

IPMP_Data_BaseClass

The `IPMP_Data_BaseClass` is intended to be extended to provide the carriage of ISO defined as well as user defined IPMP related data.

Syntax

```
abstract aligned(8) expandable(2^28-1) class IPMP_Data_BaseClass:
    bit(8) tag=0…255
{
    bit(8)  Version;
    bit(32) dataID;
    // Fields and data extending this message.
}
```

Semantics

`Version` - indicates the version of syntax used in the IPMP Data and shall be set to "0x01".

`dataID` – used for the purpose of identifying the message. Tools replying directly to a message shall include the same `dataID` in any response.

`tag` indicates the tag for the extended IPMP data.

IPMP data extending from `IPMP_Data_BaseClass` can be carried in the following three places:

- `IPMP_Descriptor`
- `IPMP_Message` which is subsequently carried in IPMP Stream.
- IPMP_MessageFromTool defined to carry messages between IPMP tools.

Table AMD3-1 — Tags for IPMP Data extending IPMP_Data_BaseClass

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_OpaqueData_tag
0x02	IPMP_AudioWatermarkingInit_tag
0x03	IPMP_VideoWatermarkingInit _tag
0x04	IPMP_SelectiveDecryptionInit_tag
0x05	IPMP_KeyData _tag
0x06	IPMP_SendAudioWatermark_tag
0x07	IPMP_SendVideoWatermark _tag
0x08	IPMP_RightsData _tag
0x09	IPMP_Secure_Container_tag
0x0A	IPMP_AddToolNotificationListener_tag
0x0B	IPMP_RemoveToolNotificationListener_tag
0x0C	IPMP_InitAuthentication_tag
0x0D	IPMP_MutualAuthentication_tag
0x0E	IPMP_UserQuery_tag
0x0F	IPMP_UserQueryResponse_tag
0x10	IPMP_ParamtericDescription_tag
0x11	IPMP_ToolParamCapabilitiesQuery_tag
0x12	IPMP_ToolParamCapabilitiesResponse_tag
0x13	IPMP_GetTools_tag
0x14	IPMP_GetToolsResponse_tag
0x15	IPMP_GetToolContext_tag
0x16	IPMP_GetToolContextResponse_tag
0x17	IPMP_ConnectTool_tag
0x18	IPMP_DisconnectTool_tag
0x19	IPMP_NotifyToolEvent_tag
0x1A	IPMP_CanProcess_tag
0x1B	IPMP_TrustSecurityMetadata_tag
0x1C	IPMP_ToolAPI_Config_tag

0x1D– 0x3F	Reserved for Inter-device messages
0x40 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

8.3.2.7 Mutual Authentication

This Subclause defines the syntax and semantics of messages used for mutual authentication and secure communications.

8.3.2.7.1 IPMP_InitAuthentication

This Subclause defines the syntax and semantics of a message used to initiate mutual authentication between two components as well as indicate the type of authentication required.

8.3.2.7.1.1 Syntax

```
class IPMP_InitAuthentication extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_InitAuthentication_tag
{
    bit (32)    Context;
    bit (8)     AuthType;
}
```

8.3.2.7.1.2 Semantics

`Context` – the 32-bit Context ID of the logical instance of the Tool with which mutual authentication is to be performed. In the case of this message being contained in the bitstream, the only possible value for `Context` is "0", the Context ID of the Terminal, as that is the only context ID that can be known in advance. In the case of this message being used between instantiated tools, `Context` shall include the actual Context ID of the required tool.

`AuthType` has the following values:

Value of AuthType	Semantic Meaning
0x00	Forbidden
0x01	No Authentication Required
0x02	No ID verify, Do secure channel
0x03	Do ID verify, No secure channel
0x04	Do ID verify, Do secure channel
0x05-0xFE	ISO Reserved
0xFF	Forbidden

8.3.2.7.1.4 Response

IPMP_Mutual_Authentication between receiving tool and indicated tool.

8.3.2.7.2 IPMP_Mutual_Authentication

This Subclause defines the syntax and semantics of a message used to negotiate the protocol used for mutual authentication as well as carrying out the agreed upon protocol. Additionally the message is used to negotiate how the secured communication channel is to be used.

8.3.2.7.2.1 Syntax

```
class IPMP_Mutual_Authentication extends IPMP_Data_BaseClass
: bit(8) tag=IPMP_MutualAuthentication_tag;
{
    bit (1)     requestNegotiation;
    bit (1)     successNegotiation;
    bit (1)     failedNegotiation;
    bit (1)     inclAuthenticationData
    bit (1)     inclAuthCodes;
    const bit (3)  reserved = 0b000;
    if (requestNegotiation) {
        bit(8)                     nCandidateAlgorithm;
        AlgorithmDesciptor     candidateAlgorithms[];
    }
    if (successNegotiation) {
        bit(8)                     nAgreedAlgorithm;
        AlgorithmDescriptor    agreedAlgorithms[];
    }
    if (inclAuthenticationData) {
        ByteArray                  AuthenticationData
    }
    if (inclAuthCodes) {
        unsigned int          type;
        if (type == 0x01) {
            unsigned int           nCertificate;
            unsigned int           certType;
            ByteArray          certificates[nCertificate];
        } elseif (type == 0x02) {
            KeyDescriptor          publicKey;
        } elseif (type == 0xFE) {
            ByteArray              opaque;
        }
        IPMP_TrustSecurityMetadata       trustData;
        ByteArray                  authCodes;
    }
}
```

8.3.2.7.2.2 Semantics

An instance of `IPMP_Mutual_Authentication` is generated by and exchanged between IPMP tools, and IPMP Tools and the Terminal for the purpose of mutual authentication.

`inclAuthCodes` – if this flag is set, authentication codes for the current message are specified.

`type` – specifying the type of the authentication codes. Semantics of each value for this variable is defined as follows:

Value of Type	Semantics
0x00:	No information regarding the type is explicitly specified.
0x01	The value specified in the variable `authCodes` is digital signature. One or more SPKI certificates or X.509 V3 certificates, depending on the value of `certType` are specified as a method to verify the signature in the variable certificates[].
0x02	The value specified in the variable `authCodes` is a digital signature. At the same time, a public key accompanied by algorithm specification is specified as a method to verify the signature.
0x03 – 0xDF	ISO Reserved.
0xE0 – 0xFD	User Private.
0xFE	Application-dependent information regarding the type is specified in the variable opaque.
0xFF	Forbidden.

`certType` – specifies type of a certificate, value assigned by a Registration Authority.

`certificates` [] – specifies data of one or more certificates. The certificate type is determined by the value of `certType`. This variable is specified, only when the variable type specifies 0x01.

`publicKey` – specifies a cryptographic public key accompanied by algorithm specification. This variable is specified, if the variable `type` specifies 0x02.

`opaque` – specifies an application-dependent method to verify the authentication codes. This variable is specified if the variable `type` specifies 0xFE.

`trustData` – if present, specifies trust and security metadata.

`authCodes` – specifies authentication codes to this message. Note that data to be signed excludes data beginning at the `type` specification of the `authCodes`.

`requestNegotiation` – indicates that the originator is requesting negotiation about an authentication method to be used in the subsequent communication.

`candidateAlgorithms` – specifies a list comprised of one or more algorithm identifiers of candidate authentication methods. The originator of the `IPMP_Mutual_Authentication` message shall specify identifiers of algorithms and/or protocols that it supports. The recipient of this message shall select ones out of the list to fulfil the requirements, which the recipient IPMP tool supports. The recipient sends another `IPMP_Mutual_Authentication` message such that `agreedAlgorithm` specifies the identification of the selected algorithms and/or protocols. If the recipient cannot find algorithms and/or protocols that it supports in the list, it returns an `IPMP_Mutual_Authentication` message specifying a "0b1" for `failedNegotiation`. The syntax of `AlgorithmDescriptor` is provided in [8.3.2.7.2.6].

NOTE When the field `candidateAlgorithms` contains algorithms of mutual authentication that support the generation of a shared secret, additional algorithm identifiers are listed to support the negotiation of message encryption, and message authentication. The purpose of each algorithm included will be implicit from its functionality. i.e. if DES is a candidate algorithm it is assumed it is specified for message encryption/decryption.

Additional Note: For algorithms that support a number of options within one algorithm such as AES supporting a number of block lengths and key lengths, only one configuration will be specified for a given identifier or registration authority listed ID. Additionally padding requirements and solutions will be included in the identifiers and Ids as well.

`inclAuthenticationData` – when set to '1', this implies that the message contains method specific data used during mutual authentication.

`AuthenticationData` – specifies data to be used for mutual authentication and whose value depends on method specific processing. This variable exists only when the flag inclAuthenticationData is set.

8.3.2.7.2.3 Response

`IPMP_Mutual_Authentication`, until authentication is successful.

8.3.2.7.2.4 BaseAuthenticationDescriptor

This Subclause defines the syntax and semantics for a descriptor that defines a base from which authentication descriptors may be extended.

8.3.2.7.2.4.1 Syntax

```
abstract aligned(8) expandable(2^28 -1) class BaseAuthenticationDescriptor : bit(8) tag=0 {
// empty. To be filled by classes extending this class.
}
```

tag table

0x00	Forbidden
0x01	IPMP_AlgorithmDescr_tag
0x02	IPMP_KeyDescr_tag
0x03-0x7F	ISO Reserved
0x7F-0xFE	User defined
0xFF	Forbidden

8.3.2.7.2.5 Key Descriptor

This Subclause defines the syntax and semantics for the carriage of a cryptographic key.

8.3.2.7.2.5.1 Syntax

```
class KeyDescriptor extends BaseAuthenticationDescriptor:
bit(8) tag = IPMP_KeyDescr_tag
{
   ByteString keyBody;
}
```

8.3.2.7.2.5.2 Semantics

This class is for specifying a cryptographic algorithm and a key conforming to the algorithm.

`keyBody` – the body of the key. The value shall be data that conforms to a rule for data structure of the key defined outside of this document.

Example – If an encryption algorithm is RSA, the value of this parameter shall be the BER encoded data which conforms to PKCS#1.

```
RSAPublicKey ::=
        SEQUENCE {
            modulus         INTEGER, -- n
            publicExponent      INTEGER -- e
        }
```

8.3.2.7.2.6 AlgorithmDescriptor

This Subclause defines the syntax and semantics for the carriage of a cryptographic algorithm or protocol identification.

8.3.2.7.2.6.1 Syntax

```
class AlgorithmDescriptor extends BaseAuthenticationDescriptor:
bit(8) tag = IPMP_AlgorithmDescr_tag {

    bit (1)     isRegistered;
    const bit (7)       reserved = 0b0000.000;
    if (isRegistered) {
        bit (16)        regAlgoID;
    }
    else {
        ByteArray       specAlgoID;
    }
    ByteArray OpaqueData;
}
```

8.3.2.7.2.6.2 Semantics

This class is for specifying an identifier of an arbitrary algorithm.

`isRegistered` – a 0b1 indicates the ID included is a normative identifier of the algorithm. A value of “0b0” indicates the algorithm is identified by an ID assigned by a registration authority

`regAlgoID` – a identifier of the algorithm as assigned by a registration authority.

`specAlgoID` – a normative identifier of the algorithm.

`SpecAlgoID` –

`SpecAlgoID`	Content	Confidentiality	Integrity
DH-G-H-x-y	Diffie-Hellman Key Agreement Scheme on a base group G. The parameter setting of the both parties is specified in certificates.	No	Yes
EDH-G-H-x-y	Ephemeral Diffie-Hellman Key Agreement Scheme on a base group G. Both parties are assumed to agree on the base group G, and each party is supposed to generate ephemeral parameters (a public key pair) for the purpose of authentication and the parameters are submitted being signed by the party. A certificate of the party is used for the opponent party to verify the signature.	No	Yes
DH-G-E-H-x-y-z	DH-G-H-x-y being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
EDH-G-E-H-x-y-z	EDH-G-H-x-y being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
RSA-OAEP-H-x	Needham-Schroeder Scheme deploying RSA-based OAEP as the public key encryption.	No	Yes
RSA-OAEP-E-H-x-z	RSA-OAEP-H-x being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
SCHNRR- G-x-y	Schnorr Identification Scheme on a base group G.	No	No

In the above table, the significance of the variables is as follows:

Variable	Definition
G	Specifies the type of a base group (e.g. a sub-groups of an elliptic curve)
E	Specifies a symmetric key cipher algorithm to realize confidentiality of messages.
H	Specifies a hash function to realize message integrity. Hash function to be used being SHA-1.
X	Specifies the size of a base field (e.g. the length in bits of the base field of an elliptic curve).
Y	Specifies a size of the order of the base group (e.g. the length in bits of the order of a base group).
Z	Specifies the length in bits of a symmetric key cipher to be used.

Remark. The actual cryptographic scheme is constructed on a proper sub-group instead of being constructed on an entire elliptic curve or an entire multiplicative group of a finite field. Further, the order of the base sub-group shall be a prime number, and its length is specified to be y bits. This value of y influences the strength of the scheme against the Pohlig-Hellman method, which finds solutions to DLP on arbitrary groups.

Generation of a MAC

In the case where a value is specified for the variable H, if either or both of the parties require the functionality of message integrity, MAC (Message Authentication Code) is attached to messages. Usage of a MAC as part of the message is optional. MAC is generated based on the hash function specified by the variable H and the shared secret being used as a key to generate MAC. MAC is generated in accordance with the following formula, where H denotes Hashing and | denotes concatenation.

```
MAC = H(H(Shared_Secret)| Message | H(Shared_Secret))
```

Generation of Keys for Message Encryption

Keys of a symmetric key cipher to be used between the parties are generated from the secret shared by the parties through the authentication procedures. For example, in the case that the key length is specified as 56 bits, the write_key_A, which the party A uses to encrypt messages, is generated by the following formula.

```
write_key_A =  TRUNC(56, SHA-1(100_LSBits_Of_Secret | "WRITE" | ID_Of_A | Rest_Of_Secret))
```

`ID_Of_A` is defined as the byte data with value "0" if Party A is the initiator of the communication, whilst it is the byte data with value 0xFF otherwise.

8.3.2.7.2.7 IPMP_TrustSecurityMetadata

This Subclause defines syntax and semantics used to carry security audit and/or trust related information.

8.3.2.7.2.7.1 Syntax

```
class IPMP_TrustSecurityMetadata : public IPMP_Data_BaseClass
: bit(8) tag = IPMP_TrustSecurityMetadata_tag
{
    bit(16) numTrustedTools;
    for (int I=0; I<numTrustedTools; I++)
    {
        bit(128)       toolID;
        DateClass   AuditDate;
        bit(16)    numTrustSpecification;
        for (int i=0; i<numTrustSpecification; i++)
        {
            bit (1)    hasCCBasedTrustMetadata;
            const bit(7) reserved = 0b0000.000;

            if (!hasCCBasedTrustMetadata) {
                DateClass   startDate;
                bit(2)      attackerProfile;
                const bit(6) reserved = 0b0000.00;
                bit(32)     trustedDuration;
            } else {
                ByteArray   CCTrustMetadata;
            }
        }
    }
}
```

8.3.2.7.2.7.2 Semantics

For `numTrustedTools` referenced by `toolID`, the trust metadata for each tool consists of the date of audit, `AuditDate` by the trust auditor.

For each of `numTrustSpecification`, the tool shall be trusted as of `startDate`, to protect against the specified `AttackerProfile` for the `TrustedDuration` minutes.

Or an alternative, though opaque, means of specifying the trust metadata using Common Criteria contained in `CCTrustMetadata`.

Attacker profile is defined as class I, II, or III where;

0x00	forbidden
0x01	Class I
0x02	Class II
0x03	Class III
0x04 – 0x0F	User defined
0x10 – 0xFE	ISO reserved
0xFF	forbidden

8.3.2.7.2.7.3 DateClass

This Subclause defines syntax and semantics for the carriage of date related information.

8.3.2.7.2.7.3.1 Syntax

```
class DateClass : {
    bit(40) Date;
}
```

8.3.2.7.2.7.3.2 Semantics

This descriptor identifies the date on which the audit took place.

`Date` – contains the audit date of IPMP tool in question, in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 least significant bits of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD). If the audit date is undefined all bits of the field are set to 1.

8.3.2.7.3 IPMP_SecureContainer

This Subclause defines syntax and semantics for the carriage of secured data as well as indicating how the contained data is secured.

8.3.2.7.3.1 Syntax

```
class IPMP_SecureContainer extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ Secure_Container_tag
{
    bit(1)  hasEncryption;
    bit(1)  hasMAC;
    bit(1)  isMACEncrypted;
    const bit(5) reserved=0b0000.0;

    if(hasEncryption)
        ByteArray  encryptedData;

        if(hasMAC && !isMACencrypted) {
            ByteArray  MAC;
        }
    }
    else {
        IPMP_Data_BaseClass  protectedMsg;
        if(hasMAC) {
            ByteArray  MAC;
        }
    }
}
```

8.3.2.7.3.2 Semantics

This forms a secure container for any message extending the IPMP_Data_BaseClass. The use of this message is optional and is defined for transmitting any message in a secure manner when needed by the application. The payload message is optionally encrypted. The MAC allows verification of the integrity of the protectedMsg, and is optionally encrypted. The key and algorithm for creating and verifying the MAC, and decrypting the encrypted payload, is negotiated as part of an external authentication mechanism.

hasEncryption - Indicates that the encrypted message is contained in encryptedData.

hasMAC – Indicates that a MAC, Message Authentication Code is contained in the field MAC.

isMACEncrypted - indicates that the encrypted portion contained in encryptedData also contains MAC information.

protectedMsg – Unencrypted message derived from IPMP_Data_BaseClass to which the contained MAC has been applied.

8.3.2.8 IPMP Tool connection and disconnection

This clause defines syntax and semantics for messages related to:

- Querying the availability of either IPMP Tool availability or the Terminal's internally implemented IPMP capabilities.
- The acquiring of tools contained in the bit stream.
- The requesting of IPMP capabilities of a given IPMP Tool or the Terminal.
- Requesting the instantiation and disconnection of IPMP Tools either as discrete loadable entities or requesting the use of the Terminal's internally implemented IPMP capabilities.
- Requesting a communication channel to an already instantiated tool.

8.3.2.8.1 IPMP_GetTools

This Subclause defines syntax and semantics to request available IPMP Tools or capabilities on/of the Terminal.

8.3.2.8.1.1 Syntax

```
class IPMP_GetTools extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_GetTools_tag
{
}
```

8.3.2.8.1.2 Semantics

Message IPMP_GetTools is used by a tool to find all the Tools, instantiated or not, that are available on the terminal.

8.3.2.8.1.3 Response

`IPMP_GetToolsResponse.`

8.3.2.8.2 IPMP_GetToolsResponse

This Subclause defines syntax and semantics to inform the presence of available IPMP Tools or capabilities on/of the Terminal.

8.3.2.8.2.1 Syntax

```
class IPMP_GetToolsResponse extends IPMP_Data_BaseClass:
    bit(8) tag = IPMP_GetToolsResponse_tag
{
    IPMP_Tool  tools[];
}
```

8.3.2.8.2.2 Semantics

`tools` – A list of tools available to the Terminal.

8.3.2.8.2.3 Response

None required

8.3.2.8.3 IPMPToolES_DecoderConfig

This Subclause defines syntax and semantics for the configuration of the Terminal to receive an IPMP Tool contained in the bitstream.

8.3.2.8.3.1 Syntax

```
class IPMPToolES_DecoderConfig extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag
{
    bit(128)        IPMP_ToolID;
    bit(1)          hasFormatID;
    bit(1)          hasPackageID;
    const bit(6)    reserved = 0b0000.00;
    if (hasFormatID)
        bit(32)         Tool_Format_ID;
    if (hasPackageID)
        bit(16)         Tool_Package_ID;
    ByteArray       OpaqueData;
}
```

8.3.2.8.3.2 Semantics

`IPMP_ToolID` – the ID of the Tool carried in this stream.

`Tool_Format_ID` – This is defined as 0x0001 for a structurally described tool. Otherwise, the `Tool_Format_ID` indicates the Binary Representation of the Tool and is assigned by a registration authority.

`Tool_Package_Id` – a 16-bit unsigned integer that indicates the details of the package of the Tool – examples are CAB, Zip, TAR. Values are assigned by a registration authority.

`OpaqueData` – Any opaque data.

A streamType identifier "IPMPToolStream" is defined for elementary streams to carry binary IPMP Tools in ISO/IEC 14496-1.

One implementation of a given tool is carried as the payload of one IPMP Tool stream, the representation format, package information and IPMP Tool ID of which is specified in DecoderConfigDescriptor in the associated ESD.

IPMP_ToolESs shall be referenced through the IOD. The IPMP Tool Manager serves as a decoder for IPMP Tool Streams. IPMP Tools carried within IPMP_ToolESs become part of the Terminal resources, and shall be installed, used and retained at the discretion of the Terminal implementation. As such, once the downloaded tool is installed or otherwise made available to the Terminal implementation, it shall be referenced via its IPMP Tool ID just like any other IPMP Tool.

Handling and usage of Tools carried in Tool elementary streams are implementation dependant.

8.3.2.8.4 IPMP_ToolES_AU

This Subclause defines syntax and semantics of the Access Unit which carries IPMP Tools in the bitstream.

8.3.2.8.4.1 Syntax

```
class IPMP_ToolES_AU
{
    bit(1) isSigned;
    const bit(7) reserved = 0b0000.000;
    if (isSigned)
    {
      ByteArray IPMP_Tool_Signature;
      bit(16) numCerts;
      for(int i=0; i<numCerts; i++)
      {
              bit(8) CertType;
              ByteArray certificates[numCerts];
      }
      bit(128) Verifying_Tool_Id;
    }
    bit(32) sizeOfTool;
    bit(8)  Tool_Body[sizeOfTool];
}
```

8.3.2.8.4.2 Semantics

IsSigned – When set to '1', shall indicate the presence of a signature in the tool stream.

IPMP_Tool_Signature – the signature of the IPMP Tool being delivered in Tool_Body in the tool stream.

NumCerts – The number of certificates included.

CertType – The type of certification mechanism being used, value assigned by a Registration Authority.

Certificates – The array of certificates.

Verifying_Tool_Id – The ID of the Tool that is required to verify the certificate(s). This shall be the ID of the Terminal or a trusted tool. When the Verifying_Tool_ID is implicit by the CertType and/or certificates, this shall be set to 0.

SizeOfTool – the size in bytes of the Tool Body

Tool_Body – the tool.

8.3.2.8.5 IPMP Tool Parametric Capabilities Query

This Subclause defines syntax and semantics to request IPMP capabilities available from either IPMP Tools or the Terminal.

8.3.2.8.5.1 Syntax

```
class IPMP_ToolParamCapabilitiesQuery extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ToolParamCapabilitiesQuery_tag
{
    IPMP_ParamtericDescription       toolParamDesc;
}
```

8.3.2.8.5.2 Semantics

This message allows a terminal to query a tool as for support for a specific parametric description. The contents of the message are the actual tool parametric descriptor that would be contained in the tools list.

`toolParamDesc`: The parametric description of the capability being queried.

8.3.2.8.5.3 Response

IPMP_ToolParamCapabilitiesResponse.

8.3.2.8.6 IPMP Tool Parametric Capabilities Query Response

This Subclause defines syntax and semantics to inform the IPMP Capabilities of an IPMP Tool or the Terminal.

8.3.2.8.6.1 Syntax

```
class IPMP_ToolParamCapabilitiesResponse extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ToolParamCapabilitiesResponse_tag
{
   bit(1) capabilitiesSupported;
   const bit(7) reserved = 0b0000.000;
}
```

8.3.2.8.6.2 Semantics

This message is the response to the above parametric capabilities query and simply returns a boolean value as to whether or not the parametric description is supported by the tool.

capabilitiesSupported: If this bit is set to ‘1’, this implies that the tool does support the parametric description queried in the preceding message. If set to ‘0’ this implies that the tool does not support the parametric description.

8.3.2.8.6.3 Response

None required.

8.3.2.8.7 IPMP_ConnectTool

This Subclause defines syntax and semantics used to request the instantiation of an IPMP tool. Note, this request may be satisfied by either the loading/instantiation of a discrete loadable IPMP Tool or by the Terminal making the requested IPMP functionality available at the point requested.

8.3.2.8.7.1 Syntax

```
class IPMP_ConnectTool extends IPMP_Data_BaseClass:
    bit(8) tag = IPMP_ConnectTool_tag
{
    IPMP_Descriptor toolDescriptor;
}
```

8.3.2.8.7.2 Semantics

Message `IPMP_ConnectTool` allows a tool to request the Terminal to create a connection to a tool identified in the `toolDescriptor`.

`toolDescriptor` - contains an IPMP descriptor to be used for determining location of new tool context to be connected. Scope of connected tool shall be the same as the requesting tool.

The Terminal shall link the new Tool Context to the Message router, before it responds to this message.

8.3.2.8.7.3 Response

`IPMP_NotifyToolEvent` with an `eventType` of "CONNECTED" or an `eventType` of "CONNECTIONFAILED".

8.3.2.8.8 IPMP_DisconnectTool

This Subclause defines syntax and semantics used to inform that a given IPMP Tool, or Terminal IPMP functionality is no longer needed. Note, the precise timing of when the IPMP Tool or functionality becomes no longer available, one should assume that once the request is made, no further use of the IPMP Tool or functionality is possible.

8.3.2.8.8.1 Syntax

```
class IPMP_DisconnectTool extends IPMP_Data_BaseClass:
    bit(8) tag = IPMP_DisconnectTool_tag
{
        bit(32)  IPMP_ToolContextID;
}
```

8.3.2.8.8.2 Semantics

Message `IPMP_DisconnectTool` allows a tool to disconnect a tool it has previously connected at a control point.

`IPMP_ToolContextID` is the ID of the Tool Context to be disconnected. `IPMP_ToolContextID` is randomly generated by the Terminal and is to be unique for each connected IPMP tool.

8.3.2.8.8.3 Response

`IPMP_NotifyToolEvent` - with an `eventType` of "DISCONNECTED" or an `eventType` of "DISCONNECTIONFAILED".

8.3.2.8.9 IPMP_GetToolContext

This Subclause defines syntax and semantics to request the logical address of a given IPMP Tool or Terminal provided IPMP functionality for messaging purposes.

8.3.2.8.9.1 Syntax

```
class IPMP_GetToolContext extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_GetToolContext_tag
{
    bit(3) scope;
    bit(1) hasIPMPDescriptorIDEx;
    const bit(4) reserved = 0b0000;

    if (hasIPMPDescriptorID)
        bit(16) IPMP_DescriptorIDEx;
}
```

8.3.2.8.9.2 Semantics

The value of the `scope` parameter can be:

SCOPE_ESD = 0x0

SCOPE_OD = 0x1

SCOPE_LOCAL = 0x2

Other values are reserved for future use.

When called with a scope of SCOPE_ESD, this message will limit the search to all tools protecting the same elementary stream as the tool who sends the message is protecting.

When called with a scope of SCOPE_OD, this message will limit the search to all tools declared in the same OD as the tool who sends the message is declared in (at the OD level or the ESD level).

When called with a scope of SCOPE_LOCAL, this message will limit the search to all tools declared in the same name scope as the tool which sends the message is declared in.

`IPMP_DescriptorIDEx` – an optional parameter to limit reply to tools from a given declaration.

8.3.2.8.9.3 Response

`IPMP_GetToolContextResponse.`

8.3.2.8.10 IPMP_GetToolContextResponse

This Subclause defines syntax and semantics used to inform the logical address of a requested IPMP Tool or Terminal provided functionality.

8.3.2.8.10.1 Syntax

```
class IPMP_GetToolContextResponse extends IPMP_Data_BaseClass:
    bit(8) tag = IPMP_GetToolContextResponse_tag
{
    bit(1)     hasESD_ID;
    bit(5)      reserved = 0.0000b;
    bit(10)     OD_ID;
    if (hasESD_ID)
       bit(16)     ESD_ID;
    bit(32)     IPMP_ToolContextID;
}
```

Where OD_ID is the ID of the OD where the tool is declared (at the OD level or ESD level), and ESD_ID, if present, is the ID of the ESD where the tool is declared.

8.3.2.8.10.2 Semantics

IPMP_ToolContextID - is the identifier of the tool Context. A null value indicates that this context does not exist.

8.3.2.8.10.3 Response

None required.

8.3.2.9 IPMP Tool notification

This clause defines syntax and semantics used for:

- A request to be notified in the event that an IPMP Tool is instantiated.
- A request to terminate IPMP Tool instantiation notification.
- The notification of an instantiation.

8.3.2.9.1 IPMP_AddToolNotificationListener

This Subclause defines syntax and semantics to request notification of the defined events of IPMP Tools.

8.3.2.9.1.1 Syntax

```
class IPMP_AddToolNotificationListener extends IPMP_Data_BaseClass :
bit(8) tag = IPMP_AddToolNotificationListener_tag
{
    bit(3) scope;
    bit(5) reserved; 0b0000.0
    bit(8) eventTypeCount;
    bit(8) eventType[eventTypeCount];
}
```

8.3.2.9.1.2 Semantics

Message IPMP_AddToolNotificationListener allows an existing IPMP Tool running on the terminal to request notification of any defined event in the terminal.

scope - The value of the scope parameter can be:

SCOPE_ESD = 0x0

SCOPE_OD = 0x1

SCOPE_LOCAL = 0x2

Other values are reserved for future use.

When called with a scope of SCOPE_ESD, this message will limit notifications for tools protecting the same elementary stream as the tool who sends the message.

When called with a scope of SCOPE_OD, this message will limit notifications for tools protecting the same OD as the tool who sends the message is declared in (at the OD level or the ESD level).

When called with a scope of SCOPE_LOCAL, this message will limit notifications to all tools connected in the same name scope as the tool who sends the message is declared in.

eventType – Type of event for which the sender shall receive notifications for and defined by the following table.

0x00	CONNECTED
0x01	CONNECTIONFAILED
0x02	DISCONNECTED
0x03	DISCONNECTIONFAILED
0x04	WATERMARKDETECTED
0x05-0xC0	ISO Reserved
0xC1-0xFF	User Defined

8.3.2.9.1.3 Response

None required.

8.3.2.9.2 IPMP_RemoveToolNotificationListener

This Subclause defines syntax and semantics used to request the termination of instantiation notifications.

8.3.2.9.2.1 Syntax

```
class IPMP_RemoveToolNotificationListener extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_RemoveToolNotificationListener_tag
{
    bit(8) eventTypeCount;
    bit(8) eventType[eventTypeCount];
}
```

8.3.2.9.2.2 Semantics

Message IPMP_RemoveToolNotificationListener allows an existing IPMP Tool running on the terminal to terminate the receipt of a previous registered event listener. If an eventTypeCount of “0” is present, all previously registered event types shall be unregistered.

8.3.2.9.2.3 Response

None required.

8.3.2.9.3 IPMP_NotifyToolEvent

This Subclause defines syntax and semantics used to notify an IPMP tool that has requested the notification of IPMP Tool instantiations that an IPMP Tool has been instantiated. Note, this message is to be delivered to both IPMP Tools that have requested notification as well as IPMP Tools that have requested the instantiation of a given IPMP Tool.

8.3.2.9.3.1 Syntax

```
class IPMP_NotifyToolEvent extends IPMP_Data_BaseClass:
    bit(8) tag = IPMP_NotifyToolEvent_tag
{
    bit(1)      hasID;
    bit(7)      reserved; 0b0000.000
    if (hasID){
        bit(10) OD_ID;
        bit(6)  reserved; 0b0000.00
        bit(16) ESD_ID;
    }
    bit(8)      eventType;
    bit(32)     toolContextID;
}
```

8.3.2.9.3.2 Semantics

Message IPMP_NotifyToolEvent notifies an IPMP Tool of a tool event for which it had previous registered as a listener.

`HasID` – Indicates that the tool is connected to a given stream declared within the given OD and/or ESD.

`OD_ID` – The ID of the OD in which the stream that the tool has been connected to.

`ESD_ID` - The ID of the stream that the tool has been connected to.

`toolContextID` – the ContextID of the tool either connected or disconnected, or the failed state of either, as indicated by the eventType.

8.3.2.9.3.3 Response

None required.

8.3.2.10 IPMP Processing

This Subclause defines syntax and semantics for the carriage of common IPMP information including:

- The notification from IPMP Tools to begin or discontinue stream processing.
- Opaque data.
- Cryptographic key data as well as the term or validity information.
- Rights expressions.
- Information used to parametrically configure decryption services.
- Information used to parametrically configure watermarking services.
- Information to be embedded or has been extracted using watermarking services.

8.3.2.10.1 IPMP_CanProcess

This Subclause defines syntax and semantics for IPMP Tools to notify the Terminal of the ability to begin or requirement to discontinue stream processing.

8.3.2.10.1.1 Syntax

```
class IPMP_CanProcess extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_CanProcess_tag
{
   bit(1) canProcess;
   bit(7) reserved = 0b0000.000;
}
```

8.3.2.10.1.2 Semantics

`canProcess` – used to indicate to the receiver of the message that the sender is able to begin processing data or must discontinue processing.

0x0	DISCONTINUE
0x1	BEGIN

8.3.2.10.2 IPMP Opaque data

This Subclause defines syntax and semantics for the carriage of opaque data.

8.3.2.10.2.1 Syntax

```
class IPMP_OpaqueData extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_OpaqueData_tag
{
    ByteArray opaqueData;
}
```

8.3.2.10.2.2 Semantics

`opaqueData` – information to be delivered.

8.3.2.10.3 IPMP_KeyData

This Subclause defines syntax and semantics for the carriage of cryptographic data as well as the indication of the terms of validity for the contained key.

8.3.2.10.3.1 Syntax

```
class IPMP_KeyData extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_KeyData_tag
{
    ByteArray keyBody;
    bit(1) hasStartDTS;
    bit(1) hasStartPacketID;
    bit(1) hasExpireDTS;
    bit(1) hasExpirePacketID;
    const bit(4) reserved = 0b0000;
    if (hasStartDTS)
    {
        bit(64) startDTS;
    }
    if (hasStartPacketID)
        bit(32) startPacketID;
    if (hasExpireDTS)
    {
        bit(64) expireDTS;
    }
    if (hasExpirePacketID)
        bit(32) expirePacketID;
    ByteArray OpaqueData;
}
```

8.3.2.10.3.2 Semantics

`keyBody` – the body of the key. The value shall be data that conforms to a rule for data structure of the key defined outside of this document.

hasStartDTS – a value of “1” indicates the presence of the following startDTS field.

hasStartPacketID – a value of “1” indicates the presence of the following startPacketID field.

hasExpireDTS – a value of “1” indicates the presence of the following expireDTS field.

hasExpirePacketID – a value of “1” indicates the presence of the following expirePacketID field.

startDTS – decoding time stamp of the first access unit for which the contained key is valid;

startPacketID – the packet ID of the first access unit for which the contained key is valid.

expireDTS – decoding time stamp of the first access unit for which the contained key is no longer valid;

expirePacketID – the packet ID of the first access unit for which the contained key is no longer valid.

OpaqueData – Any other opaque data carried in this IPMP data.

8.3.2.10.4 IPMP_RightsData

This Subclause defines syntax and semantics to carry rights expressions.

8.3.2.10.4.1 Syntax

```
class IPMP_RightsData extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_RightsData_tag
{
    ByteArray rightsInfo;
}
```

8.3.2.10.4.2 Semantics

rightsInfo – This contains the details of usage rights information.

8.3.2.10.5 IPMP_SelectiveDecryptionInit

Defined in **Error! Reference source not found.**.

8.3.2.10.6 IPMP_AudioWatermarkingInit

Defined in Annex B.

8.3.2.10.7 IPMP_SendAudioWatermark

Defined in Annex B.

8.3.2.10.8 IPMP_VideoWatermarkingInit

Defined in Annex C.

8.3.2.10.9 IPMP_SendVideoWatermark

Defined in Annex C.

8.3.2.11 User Interaction Messages

This clause defines syntax and semantics for messages which allow information to be exchanged between the User and an entity requiring information from the User. Although the word 'text' is used repeatedly throughout the following syntax and semantics, how the requested information is displayed or presented to the user as well as how the User supplies possibly requested information is application and implementation dependent. An example could be that although text may be requested with a textual prompt, the User input and output available on the Terminal may be voice recognition and generation instead.

8.3.2.11.1 IPMP_UserQuery

This Subclause defines syntax and semantics used to request or present information from/to the User.

8.3.2.11.1.1 Syntax

```
class IPMP_UserQuery extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_UserQuery_tag
{
    bit(1)             inclDisplayTitle;
    bit(1)             inclDisplayText;
    bit(1)             needReplyText;
    bit(1)             inclOptionSelect;
    bit(1)             inclSMIL;
    const bit(3)    reserved = 0b000;
    bit(24)         numOfAltText;
    for(int i=0; i< numOfAltText; i++)
    {
           bit(24)        languageCode;
           if  (inclDisplayTitle) {
               ByteArray  titleText;
           }
           if  (inclDisplayText) {
               bit(8)     nDisplayText;
              DTArray     displayText[];
           }
           if  (needReplyText) {
               bit(8)     nPromptText;
           QTArray     promptText[];
           }
           if  (inclOptionSelect) {
               bit(8)         nOption;
               OptionArray    option[]
           }
           if (inclSMIL)  {
               bit(1)     isReferenced;
               const bit(7)   reserved = 0b0000.000;
           if (isReferenced){
               ByteArray  SMIL_URL;
               }
               else {
               ByteArray  SMIL;
               }
           }
    }
}
```

8.3.2.11.1.2 Semantics

`languageCode` - Contains the ISO 639-2:1998 bibliographic three character language code of the corresponding audio/speech or text object that is being described.

`titleText` - Title of dialog display.

`displayText` - Text to be displayed to the user.

`needReplyText` - Text expected back from the user.

`promptText` - Text to be displayed to the end user to indicate purpose of text input field, i.e. “User ID”, “Password”, “PIN”, etc.

`inclOptionSelect` - An option, true/false, input is needed from the user.

`isExclusive` - If more than one option is associated with a given display text, this indicates mutual exclusivity of options.

`optionText` - Text to be displayed indicating purpose of option selection, i.e. “One month purchase”, “One time play”, “Render at 1024/768”, etc.

`SMIL_URL` - Fully qualified location of SMIL file to be displayed.

`SMIL` - SMIL file to be displayed.

8.3.2.11.1.3 DTArray

This Subclause defines syntax and semantics to carry information used to provide a general heading for the information and/or request for information presented to the User.

8.3.2.11.1.3.1 Syntax

```
class DTArray
{
    bit(16)     ID;
    ByteArray   displayText;
}
```

8.3.2.11.1.4 QTArray

This Subclause defines syntax and semantics to indicate that information is requested from the user as well as an indication of what the information that is requested is. An example could be that information is requested in textual format, such as a password or User ID and what is requested, “Password” or “User ID” is displayed textually to the User along with an appropriate area to receive textual input from the User.

8.3.2.11.1.4.1 Syntax

```
class QTArray
{
    bit(16)     ID;
    bit(16)     SubID;
    bit(1)      isHidden;
    bit(7)      reserved = 0b0000.000;
    ByteArray   promptText;
}
```

8.3.2.11.1.5 OptionArray

This Subclause defines syntax and semantics to indicate a single option or choice must be made. If more than one `OptionArray` is included with the same ID, the `isExclusive` bit may be used to indicate whether more than one option may be selected or if only one option of those available must be selected.

8.3.2.11.1.5.1 Syntax

```
class OptionArray
{
    bit(16)     ID;
    bit(16)     SubID;
    bit(1)      isExclusive;
    bit(7)      reserved = 0b0000.000;
    ByteArray   promptText;
}
```

8.3.2.11.1.5.2 Semantics

`IsExclusive` - When set to '1', this implies a mutually exclusive condition of the option selector.

`ID` - A serial number to be associated with the contained data or to associate contained data with other data.

`SubID` - A serial number to associate items within an ID group.

`promptText` – Text to be presented to the User to indicate the type of information requested.

`isHidden` – If set to one, the response field for this text shall not reveal the text input by the user, e.g. password input.

8.3.2.11.1.5.3 Response

`IPMP_UserQueryResponse`

8.3.2.11.2 IPMP_UserQueryResponse

This Subclause defines syntax and semantics to carry information provided by the User to an `IPMP_UserQuery` request.

8.3.2.11.2.1 Syntax

```
class aligned (8)UserIPMP_UserQueryResponse extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_UserQueryResponse_tag
{

        bit(1)              inclReplyText;
        bit(1)              inclOptionSelect;
        const bit(6)        reserved = 0b0000.00;
        bit(24)             languageCode;
        if   (inclReplyText) {
            bit(8)          nReplyText;
            QTArray         ReplyText[];
        }
        if   (inclOptionSelect) {
            bit(16)       numOfOptions;
            for (i=0; i<numOfOptions; i++)
            {
                bit(1)      optionResult;// 0b1 = TRUE, 0b0 = FALSE;
            }
    }
}
```

8.3.2.11.2.2 Semantics

`replyText` - Text entered by user. Only fields with entered text need be included. If the original request contained reply text fields but the terminal does not support text input then an `inclReplyText` of "`0b0`" would indicate so.

`optionResult` - Identical semantics and rules as with `replyText` except that all options must be represented and in the order they were initially specified.

8.3.2.11.2.3 Response

None Required.

8.3.2.12 IPMP Information Delivery Functions

This clause defines syntax and semantics for messages that facilitate delivery of IPMP Information among IPMP Tools and terminal, and from bitstream to IPMP Tools.

8.3.2.12.1 IPMP_ToolMessageBase

This Subclause defines syntax and semantics for a message base which can be extended to deliver information of different types to IPMP Tools.

8.3.2.12.1.1 Syntax

```
Aligned(8) abstract expandable(2^28-1)class IPMP_ToolMessageBase: bit(8) tag = 0 {
    bit(8)  Version;
    bit(32) sender;
    bit(32) recipient;
}
```

8.3.2.12.1.2 Semantics

`IPMP_ToolMessageBase` - an expandable base class for IPMP Tool Messages. The extension tags are defined in Table AMD3-2.

`Version` – indicates the version of syntax used in the messages and shall be set to "0x01".

`Sender` - indicates the context ID of the originator of the message. The value "0x00" is reserved for the terminal.

`Recipient` - indicates the context ID of the intended recipient of the message. "0x00" is reserved for the terminal.

Table AMD3-2 — Tags for messages extending IPMP_ToolMessageBase

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_MessageFromBitstream_tag
0x02	IPMP_DescriptorFromBitstream_tag
0x03	IPMP_MessageFromTool_tag
0x04 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

8.3.2.12.2 IPMP_MessageFromBitstream

This Subclause defines syntax and semantics for an extension to `IPMP_ToolMessageBase` to support the delivery of `IPMP_Message`s to IPMP Tools. Note, the delivery of `IPMP_Message`s to IPMP Systems, as identified by `IPMP_DescriptorID`, as opposed to `IPMP_DescriptorIDEx`, is done on an implementation basis.

8.3.2.12.2.1 Syntax

```
class IPMP_MessageFromBitstream extends IPMP_ToolMessageBase :
    bit(8) tag = IPMP_MessageFromBitstream_tag
{
    IPMP_Message messages[];
}
```

8.3.2.12.2.2 Semantics

Message `IPMP_MessageFromBitstream` is used to deliver `IPMP_Messages` received in the Content to the IPMP Tool context specified in the `IPMP_Message`. If an IPMP Access Unit delivered in the IPMP Elementary Stream contains more than one `IPMP_Message` for a specific IPMP Tool, all `IPMP_Message` for that tool will be included in a single `IPMP_MessageFromBitstream` message.

`messages[]` - an array of `IPMP_Message's`.

8.3.2.12.2.3 Response

None required.

8.3.2.12.3 IPMP_DescriptorFromBitstream

This Subclause defines syntax and semantics for an extension to `IPMP_ToolMessageBase` to support the delivery of `IPMP_Descriptor`s to IPMP Tools. Note, the delivery of `IPMP_Descriptor`s to IPMP Systems, as identified by `IPMP_DescriptorID`, as opposed to `IPMP_DescriptorIDEx`, is done on an implementation basis.

8.3.2.12.3.1 Syntax

```
class IPMP_DescriptorFromBitstream extends IPMP_ToolMessageBase :
    bit(8) tag = IPMP_DescriptorFromBitstream_tag
{
    IPMP_Descriptor toolDescriptor;
}
```

8.3.2.12.3.2 Semantics

Message `IPMP_DescriptorFromBitstream` is used to deliver an `IPMP_Descriptor` received in the bitstream to the IPMP Tool specified in the `IPMP_Descriptor`.

`toolDescriptor` - the `IPMP_Descriptor` received in the bitstream.

8.3.2.12.3.3 Response

None required.

8.3.2.12.4 IPMP_MessageFromTool

This Subclause defines syntax and semantics for an extension to `IPMP_ToolMessageBase` to support the delivery of IPMP information to and between IPMP Tools and to and between IPMP Tools and the Terminal.

Note, the delivery of IPMP Information to IPMP Systems, as identified by IPMP_DescriptorID, as opposed to IPMP_DescriptorIDEx is done on an implementation basis.

8.3.2.12.4.1 Syntax

```
class IPMP_MessageFromTool extends IPMP_ToolMessageBase:
    bit (8) tag = IPMP_MessageFromTool_tag
{
    IPMP_Data_BaseClass messages[];
}
```

8.3.2.12.4.2 Semantics

message – a container for IPMP_Data_BaseClass derived data.

8.3.2.12.4.3 Response

Received message dependant.

8.3.2.12.5 IPMP_ToolAPI_Config

This Subclause defines syntax and semantics for the carriage of IPMP Tools' instantiation and messaging API information. When this data is put in IPMP_Descriptor, the carried API information applies to the IPMP Tool that is referenced by the IPMP_ToolID in the same IPMP_Descriptor, i.e., when this Instantiation_API_ID is present in the IPMP Descriptor, the terminal should use the carried API information to instantiate and to pass messages to/from the associated IPMP Tool.

8.3.2.12.5.1 Syntax

```
class IPMP_ToolAPI_Config extends IPMP_Data_BaseClass:
    bit (8) tag = IPMP_ToolAPI_Config_Tag
{
    bit(1)      has_Instantiation_API_ID;
    bit(1)      has_Messaging_API_ID;
    const bit(6)    reserved = 0b0000.00;
    if  (has_Instantiation_API_ID)
    bit(32)     Instantiation_API_ID;
   if   (has_Messaging_API_ID)
   bit(32)  Messaging_API_ID;
   ByteArray    OpaqueData;
}
```

8.3.2.12.5.2 Semantics

has_Instantiation_API_ID – Indicates that the Instantiation_API_ID is contained in the IPMP_ToolAPI_Config.

has_Messaging_API_ID – Indicates that the Messaging_API_ID is contained in the IPMP_ToolAPI_Config.

Instantiation_API_ID – A 32 bits ID that indicates the API for instantiation of the IPMP Tool and is assigned by a registration authority.

Messaging_API_ID – A 32 bits ID that indicates the messaging API to pass messages to/from the IPMP Tool and is assigned by a registration authority.

OpaqueData – Any opaque data that may assist the terminal of the instantiation or messaging of the IPMP Tool.

8.3.2.12.5.3 Response

None required.

End insertion after Subclause 8.3.2.5.2

Two additional new descriptors have been created for inclusion in the IOD. They are `IPMP_ToolsListDescrTag`, which is used to identify a descriptor class that contains the declaration of all IPMP tools required for the protection of a given MPEG-4 content and `IPMP_ToolTag`, which is used to identify a descriptor class contained within an `IPMP_ToolsListDescr` and identifies a single IPMP tool.

In Subclause 8.6.3.1, insert the following text after the existing `IPMP_DescriptorPointer` found in the `ObjectDescriptor`:

```
        IPMP_Descriptor ipmpDescr [0 .. 255];
```

End Subclause 8.6.3.1 insertion.

In Subclause 8.6.3.2, insert the following text after the existing semantics for `ipmpDescrPtr[]` for the `ObjectDescriptor`:

`ipmpDescr []` – a list of `IPMP_Descriptors` that may be referenced by streams declared in `esDescr`. The array shall have any number of zero up to 255 elements. The following scope and usage rules apply:

i. Entries in the `ipmpDescr` table define IPMP System/Tools that can be referenced by `IPMP_DescriptorPointers` located in the OD itself or ESDs declared in this OD.
ii. OD contained `IPMP_Descriptors` have scope within the given OD only and shall not be referenced by subsequently declared IODs, ODs, streams nor available for updating via `IPMP_DescriptorUpdates`.
iii. The OD contained `IPMP_Descriptors` shall not be referenced by IODs, ODs or streams declared in OD declared OD or Scene streams.

In Subclause 8.6.4.1, insert the following text after the existing `IPMP_DescriptorPointer` found in the `InitialObjectDescriptor`:

```
        IPMP_Descriptor ipmpDescr [0 .. 255];
        IPMP_ToolListDescriptor toolListDescr[0 .. 1];
```

In Subclause 8.6.4.2, insert the following text after the existing semantics for `ipmpDescrPtr[]` for the `InitialObjectDescriptor`:

`ipmpDescr []` – a list of `IPMP_Descriptors` that may be referenced by streams declared in `esDescr`. The array shall have any number of zero up to 255 elements. The following scope and usage rules apply:

i. Entries in the `ipmpDescr` table define IPMP System/Tools that can be referenced by `IPMP_DescriptorPointers` located in the IOD itself or ESDs declared in this IOD.
ii. IOD contained `IPMP_Descriptors` have scope within the given IOD only and shall not be referenced by subsequently declared IODs, ODs, streams nor available for updating via `IPMP_DescriptorUpdates`.
iii. The IOD contained `IPMP_Descriptors` shall not be referenced by IODs, ODs, streams declared in IOD declared OD or Scene streams.

`toolListDescr` – a list of all IPMP tools required for the processing of the content. The array shall have zero or 1 element.

End Subclause 8.6.4.2 insertion

An additional stream type has been defined to carry IPMP tools in the content stream.

In Subclause 8.6.6.2, replace Table 9 with the following:

Table 9 - streamType Values

`streamType` value	streamType description
0x00	Forbidden
0x01	ObjectDescriptorStream
0x02	ClockReferenceStream
0x03	SceneDescriptionStream
0x04	VisualStream
0x05	AudioStream
0x06	MPEG7Stream
0x07	IPMPStream
0x08	ObjectContentInfoStream
0x09	MPEGJStream
0x0A	InteractionStreamType
0x0B	IPMPToolStream
0x0C - 0x1F	reserved for ISO use
0x20 - 0x3F	user private

Insert the following text after Subclause 8.6.14.2.

IPMP Tool List Specification

For each tool, this includes

1) IPMP Tool Identifier

2) Optional Parametric Description of the Tool.

3) Alternative Tools to the given Tool, any one of which replace the others without loss of functionality.

The Tool List shall be in the IOD, in an `IPMP_ToolListDescriptor`. Binary IPMP Tools are carried in separate elementary streams associated with the IOD. The specification of the syntax for the Tool List is as follows.

The `IPMP_ToolListDescriptor` conveys the list of IPMP tools required to access the content associated with the `InitialObjectDescriptor` in which it is described, and may include a list of alternate IPMP tools or parametric descriptions of tools required to access the content.

IPMP_ToolListDescriptor

This Subclause defines syntax and semantics for the carriage of a list of IPMP Tools required for the processing of the presentation.

Syntax

```
class IPMP_ToolListDescriptor extends BaseDescriptor :
    bit(8) tag= IPMP_ToolsListDescrTag
{
    IPMP_Tool ipmpTool[0 .. 255];
}
```

Semantics

`IPMP_Tool` – a class describing a logical IPMP Tool required to access the content.

IPMP_Tool

The IPMP Tool Identifier (or `IPMP_ToolID`) is 128-bits long, and shall contain a unique identification number for the IPMP Tool. A registration authority for IPMP Tools that use a unique ID is required. The registration authority shall maintain an optional association of the download URLs for various implementations of the given tool for various platforms. These platforms will be described to adequate detail using a structured representation. The `IPMP_ToolID` identifies a specific IPMP Tool (not a specific implementation of such a tool), unless in the reserved range for parametrically defined tools. Currently assigned 16-bit `IPMPS_Types` shall be directly mapped to a 128-bit ID by prepending with 112 zero bits; the RA will be initialized with such values. Specific values within this 128-bit space are reserved for indicating parametric tools, the bitstream, the terminal, and other special addresses. These values shall not be assigned to registered Tools.

Table Amd3-3 - Values of IPMP_ToolID

IPMP_ToolID	Semantics
0x0000	Forbidden
0x0001	Content
0x0002	Terminal
0x0003 - 0x2000	Reserved for ISO use
0x2001 - 0xFFFF	Carry over from 14496-1 RA
0x10000 - 0x100FF	Parametric Tools or Alternative Tools
0x100FF – 2^128-2	Open for registration
2^128-1	Forbidden

Syntax

```
class IPMP_Tool extends BaseDescriptor :
    bit(8) tag= IPMP_ToolTag
{
    bit(128)    IPMP_ToolID;
    bit(1)      isAltGroup;
    bit(1)      isParametric;
    const bit(6)        reserved=0b0000.00;

    if(isAltGroup){
        bit(8)      numAlternates;
        bit(128)    specificToolID[numAlternates];
    }
    if(isParametric)
        IPMP_ParamtericDescription    toolParamDesc;
    ByteArray ToolURL[];
}
```

Semantics

Each instance of Class `IPMP_Tool` identifies one IPMP Tool that is required by the Terminal to Consume the Content. This Tool shall be specified either as a unique implementation, as one of a list of alternatives, or through a parametric description.

A unique implementation is indicated by the `isAltGroup` and `isParametric` fields both set to zero. In this case, the `IPMP_ToolID` shall be from the range reserved for specific implementations of an IPMP Tool and shall directly indicate the required Tool.

In all other cases, the `IPMP_ToolID` serves as a Content-specific abstraction for an IPMP Tool ID since the actual IPMP Tool ID of the Tool is not known at the time of authoring the Content, and will depend on the Terminal implementation at a given time for a given piece of Content.

A parametric description is indicated by setting the isParametric field to one. In this case, the Terminal shall select an IPMP Tool that meets the criteria specified in the following parametric description. In this case, the `IPMP_ToolID` shall be from the range reserved for Parametric Tools or Alternative Tools. The actual IPMP Tool ID of the Tool that the terminal implementation selects to fulfill this parametric description is known only to the Terminal. All the Content, and other tools, will refer to this Tool, for this Content, via the `IPMP_ToolID` specified. Note, this is not for message addressing.

A list of alternative Tools is indicated by setting the `isAltGroup` flag to "1". The subsequent specific Tool IDs indicate the Tools that are equivalent alternatives to each other. If the `isParametric` field is also set to one, any Tool that is selected under the conditions for parametric tools (as discussed in the paragraph above) shall be considered by the Terminal to be another equivalent alternative to those specified via specific Tool IDs. The Terminal shall choose one from these equivalent alternatives at its discretion. The actual IPMP Tool ID of this Tool is known only to the Terminal.

`IPMP_ToolID` – the identifier of the IPMP Tool, as discussed above.

`isAltGroup` – if set to one, this `IPMP_Tool` contains a list of alternate IPMP Tools.

`numAlternates` – the number of alternative IPMP Tools specified in `IPMP_Tool`.

`specificToolID` – an array of the IDs of specific alternative IPMP Tools that can allow consumption of the content.

`isParametric` – `IPMP_Tool` contains a parametric description of an IPMP Tool. In this case, `IPMP_ToolID` is an identifier for the parametrically described IPMP Tool, and the Terminal shall route information specified in the bitstream for `IPMP_ToolID` to the specific IPMP Tool instantiated by the terminal.

`ToolURL` – An array of informative URLs from which one or more tools specified in `IPMP_Tool` may be obtained in a manner defined outside the scope of these specifications.

IPMP_ParametricDescription

Using a parametric description, the content provider can now describe what type of IPMP tool is required to playback the content, instead of using fixed tool IDs. For example, the content provider can specify that an AES tool, with block size of 128 bits is required to decrypt video stream. The IPMP terminal, upon receiving such description specifying this tool, can then choose an optimised AES tool from the embedded tools.

This clause contains an illustration of the hierarchy that a parametric description would follow. It does not attempt to define any specific scheme for any specific Tool type. We anticipate that only a basic framework will appear in the current version of the specification, and enhancements to the same will be left for future addendums and/or versions.

1. Optional comment
2. Version of parametric description syntax
3. Class of Tool

   e.g. Decryption, Rights Language Parser
4. Sub-class of Tool
   a. e.g. for Decryption: AES, DES, NESSIE etc
   b. e.g. for Watermarking: “Panos’s watermarking tool” etc
   c. e.g. for Rights Language Parser: “Fred’s Rights Parser”
   d. e.g. for Protocol Parser: “Mary’s Protocol Parser”
5. Sub-class-specific information
   a. e.g. for DES: number of bits, stream and/or block decipher capability
   b. e.g. for Rights Language Parser : version

The parametric description is defined to allow a generic description of any type of IPMP tool, no matter the type of tool.

Syntax

```
class IPMP_ParamtericDescription extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_ParamtericDescription_tag = 0x10
{
    ByteArray       descriptionComment;
    bit(8)          majorVersion;
    bit(8)          minorVersion;
    Int i               numOfDescriptions;
    For (i = 0 to numOfDescriptions - 1){
        ByteArray   class;
        ByteArray   subClass;
        ByteArray   typeData;
        ByteArray   type;
        ByteArray   addedData;
    }
}
```

Semantics

`class` - class of the parametrically described tool, for example, decryption.

`subClass` - sub-class of the parametrically described tool, for example, AES under decryption class.

`typeData` - specific type data to describe a particular type of tool, for example, Block_length, to further specify a AES decryption tool `type` - value of the type data above, for example, 128 for the Block_length.

`addedData` - Any additional data which may help to further describe the parametrically defined tool.

ByteArray

This Subclause defines syntax and semantics to carry a generic string or array of bytes which is used extensively throughout the IPMP specifications.

Syntax

```
expandable class ByteArray
{
    bit(8) data[sizeOfInstance()];
}
```

Semantics

`data` - the string or array of bytes carried.

End text insertion after Subclause 8.6.14.2.

The extensions to the `IPMP_DescriptorPointer` support the extended ability to identify which specific IPMP stream or streams the IPMP tools declared in the corresponding `IPMP_Descriptor`, identified by the `IPMP_DescriptorIDEx`, wish to receive. Previously it was necessary for every single IPMP tool to reference a unique IPMP elementary stream to receive updates. With these extensions, multiple IPMP tools may receive updates from the same stream.

In Subclause 8.6.13.1, replace the syntax with the following text:

```
class IPMP_DescriptorPointer extends BaseDescriptor :
bit(8) tag = IPMP_DescrPtrTag
{
       bit(8) IPMP_DescriptorID;
         if (IPMP_DescriptorID == 0xff){
         bit(16) IPMP_DescriptorIDEx;
         bit(16) IPMP_ES_ID;
    }
}
```

In Subclause 8.6.13.2, replace the semantics with the following text:

The `IPMP_DescriptorPointer` appears in the `ipmpDescPtr` section of an OD or ESD structures.

The presence of this descriptor in an object descriptor indicates that all streams referred to by embedded `ES_Descriptors` are subject to protection and management by the IPMP Tool specified in the referenced `IPMP_Descriptor`.

The presence of this descriptor in an `ES_Descriptor` indicates that the stream associated with this descriptor is subject to protection and management by the IPMP Tool specified in the referenced `IPMP_Descriptor`.

`IPMP_DescriptorID` is the ID of the `IPMP_Descriptor` being referred to. The `bit(8)` field is to support backward compatibility, for which support for extended IPMP stream association is not provided for.

`IPMP_DescriptorIDEx` is the ID of the `IPMP_Descriptor` being referred to. The `bit(16)` field refers to extension defined `IPMP_Descriptors` and also supporting the extended IPMP stream association.

`IPMP_ES_ID` is the id of an IPMP stream that may carry messages intended to the tool pointed to by `IPMP_DescriptorIDEx`. In case more than one IPMP stream is needed to feed the IPMP tool, several

IPMP_DescriptorPointer can be given with the same IPMP_DescriptorIDEx and different IPMP_ES_ID. If the IPMP_ES_ID is null, it means the IPMP tool does not require an IPMP stream. A value of 2^16-1 for IPMP_ES_ID indicates that this field should be ignored, meaning that the tool pointed to by IPMP_DescriptorIDEx may receive messages from any IPMP stream within the presentation.

The list of IPMP streams identified by occurrences of the IPMP_ES_ID field (with a value different than 2^16-1) for a single IPMP_DescriptorIDEx is exhaustive: the IPMP tool identified by the IPMP_DescriptorIDEx may not receive messages from any other IPMP streams than the ones identified in this list. In order to facilitate editing, the IPMP_DescriptorPointer must be modified when stored in a file: the IPMP_ES_ID field must be replaced with the corresponding index in the OD track's 'mpod' table as defined in ISO/IEC 14496-1:2001, Subclause 13.2.3.7.2.

In Subclause 8.6.14.1, replace the syntax with the following text:

```
class IPMP_Descriptor() extends BaseDescriptor : bit(8) tag = IPMP_DescrTag
{
   bit(8) IPMP_DescriptorID;
    unsigned int(16) IPMPS_Type;
    if (IPMP_DescriptorID == 0xFF && IPMPS_Type == 0xFFFF){
        bit(16) IPMP_DescriptorIDEx;
        bit(128) IPMP_ToolID;
        bit(8) controlPointCode;
    if (controlPointCode > 0x00)
        bit(8) sequenceCode;
            IPMP_Data_BaseClass IPMPX_data[];
    }
    else if (IPMPS_Type == 0)
        bit(8) URLString[sizeOfInstance-3];
    else
        bit(8) IPMP_data[sizeOfInstance-3];
}
```

In Subclause 8.6.14.2, replace the semantics with the following text:

The IPMP_Descriptor carries IPMP information for one or more IPMP Tool instances. It shall also contain optional instantiation information for one or more IPMP Tool instances. IPMP_Descriptors are conveyed in either initial object descriptors, object descriptors or object descriptor streams via IPMP_DescriptorUpdate commands. Multiple definitions of the same IPMP_Descriptor within a single IPMP_DescriptorUpdate command or a single decoder specific information structure for an IPMP stream are not allowed. The behavior in such a situation is undefined. Note that, however, an *IPMP_Descriptor* may be modified/updated through subsequent IPMP_DescriptorUpdate commands received in the OD stream. IPMP_Descriptors shall be referenced by object descriptors or ES_Descriptors, using IPMP_DescriptorPointer.

IPMP_DescriptorID - a unique ID for this IPMP_Descriptor within its name scope. Values of "0x00" and "0xFF" are forbidden in the case of signaling an extension descriptor. The scope of the IPMP_DescriptorID is suggested to be the same as the OD, or IOD in which is it contained. IPMP_DescriptorID is for use in systems conforming to the previous definition as well as a signal indicating the use of IPMP_DescriptorIDEx for IPMP extensions.

NOTE Although it is possible to implement an application supporting both the use of IPMP protection schemes defined through the use of IPMP_Descriptors some of which contain IPMP_DescriptorID and some of which contain IPMP_DescriptorIDEx to protect separate streams, the behavior of the association of a single stream to both types of IPMP_Descriptors is undefined.

IPMP_DescriptorIDEx - a unique ID for this IPMP_Descriptor within its name scope. Values of "0x0000" and "0xFFFF" are forbidden. The scope of the IPMP_DescriptorIDEx is suggested to be the same as the OD, or IOD in which is it contained.

IPMP_ToolID – the IPMP_ToolID of the IPMP Tool described by this IPMP_Descriptor. A zero, "0" value does not correspond to an IPMP Tool but is used to indicate the presence of a URL.

`URLString[]` - contains a UTF-8 encoded URL that shall point to the location of a remote `IPMP_Descriptor`. If the `IPMPS_Type` of this `IPMP_Descriptor is` 0, another URL is referenced. This process continues until an `IPMP_Descriptor` with a non-zero `IPMPS_Type` is accessed.

`controlPointCode` – specifies the IPMP control point at which the IPMP Tool resides, and is one of the following values:

controlPointCode	**Description**
0x00	No control point.
0x01	Control Point between the decode buffer and the decoder. This is between the decode buffer and class loader for MPEG-J streams.
0x02	Control Point between the decoder and the composition buffer.
0x03	Control Point between the composition buffer and the compositor.
0x04	BIFS Tree
0x05-0xDF	ISO Reserved
0xE0-0xFE	User defined
0xFF	Forbidden

NOTE The only difference between receiving composition units before the CB and after the CB in the MPEG-4 Systems decoder model is the order in which the units are received when the associated `DTS` is different from the `CTS`; in this case the decoding order is different from the composition order. For example, suppose that a watermark payload is embedded in more than a single video frame; if the watermark payload was embedded in decoding order, it has to be extracted before the CB; instead, if it was embedded in composition order, it has to be extracted after the CB.

NOTE For streams of type “0x01”, `ObjectDescriptor` and of type “0x02”, `ClockReferenceStream`, only a `controlPointCode` of “0x00”, “0x01” or the range “0xE0-0xFE” are meaningful.

`sequenceCode` - The higher the sequence code, the higher the sequencing priority of the IPMP Tool instance at the given control point. Thus the tool with the highest `sequenceCode` for a given control point on a given stream shall process data first for that control point for that stream. Two tools shall not have the same sequence number at the same control point for the same stream.

`IPMPX_data` - The IPMP data that is extended from `IPMP_Data_BaseClass`, for the given IPMP tool.

`IPMP_data` – Data of unspecified format.

In Subclause 8.8.1, replace the text:

“This Subclause specifies methods how to associate an IPMP system to an elementary stream or a set of elementary streams.”

with the text:

“This Subclause specifies methods to:

a. Indicate IPMP Tools required for the processing of a given MPEG-4 presentation.

b. Associate an IPMP System to an elementary stream or a set of elementary streams.

c. Associate an IPMP Tool to a specified Control Point of an elementary stream or set of elementary Streams.

d. Perform Mutual Authentication between IPMP Tools and between IPMP Tools and the Terminal.

e. Request the instantiation of one or more IPMP Tools by another IPMP Tool.

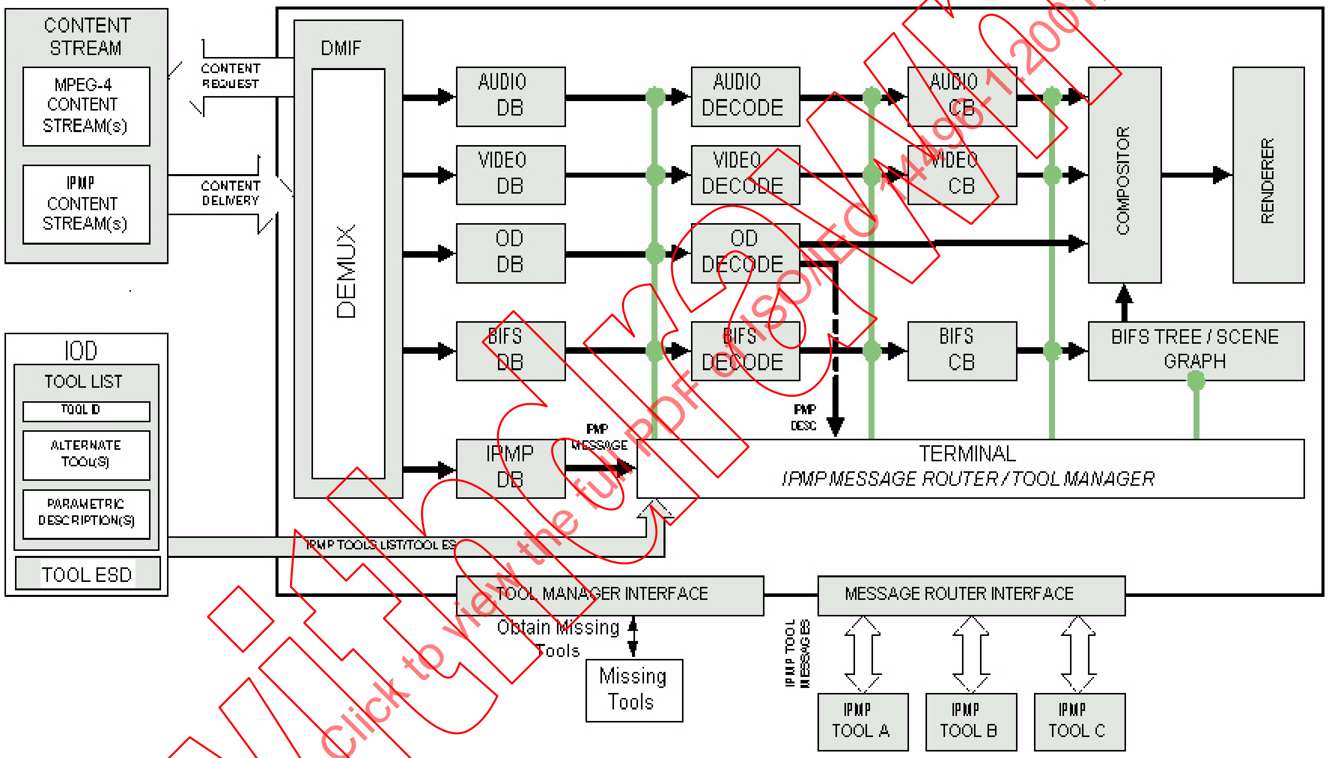
f. Request and receive notification of the instantiation of IPMP Tools.

g. Provide a communication channel between IPMP Tools and the User.”

In Subclause 8.8.3, replace the entire text with:

“Object Descriptor streams shall not be affected by IPMP Systems, i.e., they shall always be available without protection by IPMP Systems. However, management may be applied using IPMP Tools. IPMP_Descriptors, which reference one or more IPMP Tools, may be directly included in an Object Descriptor for use by elementary streams referenced within the same Object Descriptor.
The scope of the IPMP_Descriptors included and used in this way is limited to only the Object Descriptor itself and the streams defined by reference within the Object Descriptor and may not be referenced by any subsequent descriptors which may be included in the streams referenced in the Object Descriptor. Additionally, IPMP_Tools referenced in this way shall not receive updates either by IPMP Streams or IPMP descriptor updates.”

In Subclause 8.8.6, replace the existing diagram with:



Add the following annexes after Annex W.

Annex X
(normative)

Selective Decryption Configuration Data

X.1 Introduction

The selective decryption configuration data is used to communicate to the decryptor, how the encryption of the received content bitstream is encrypted, whether all bits are encrypted or only portions of the received bits are encrypted. In the case when only portions of the received bits are encrypted, what portions of the received are encrypted and therefore need to be decrypted.

X.2 IPMP_SelectiveDecryptionInit

X.2.1 Syntax

```
class IPMP_SelectiveDecryptionInit extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_SelectiveDecryptionInit_tag;
{
    bit(8) mediaTypeExtension;
    bit(8) mediaTypeIndication;
    bit(8) profileLevelIndication;
    bit(8) compliance;
    bit(8) numBufs;
    for(i=0, i<numBufs; i++)
        Struct bufInfoStruct {
            bit(128) cipher_Id;
            bit(8) syncBoundary;

            bit(1) isBlock;
            const bit(7) reserved = 0b0000.000;
            if(isBlock) {
                bit(8) mode;
                bit(16) blockSize;
                bit(16) keySize;
            } else {
                ByteArray Stream_Cipher_Specific_Init_Info;
            }
        }
    }
    bit(1) isContentSpecific;
    const bit(7) reserved = 0b0000.000;
    if(isContentSpecific) {
        bit(8) numFields;
        for(i=0, i< numFields; i++) {
            Struct fieldStruct {
                bit(8) field_Id;
                bit(3) field_Scope;
                const bit(5) reserved = 0b0000.000;
                bit(8) buf;
                bit(1) isMapped;
                const bit(7) reserved = 0b0000.000;
                if(isMapped){
                    bit(1) sendMapTable;
                    bit(1) isShuffled;
                    const bit(6) reserved = 0b0000.000;
                    if(sendMapTable){
                        bit(16) sizeMapTable;
                        bit(16) mappingTable[sizeMapTable]
                    }
                    if(isShuffled){
                        ByteArray shuffleSpecificInfo;
                    }
                }
            }
        }
    } else {
        bit(16) nSegments;
        bit(16) RLE_Data[nSegments]
    }
}
```

X.2.2 Semantics

This message allows a terminal to configure a selective decryption tool.

mediaTypeExtension: extends mediaTypeIndication below; The following values are defined.

mediaTypeExtension	Semantics
0x00	ISO/IEC
0x01	ITU
0x02 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

mediaTypeIndication: indication of format definition, e.g., MPEG-4 or MPEG-2 etc.; example values could be those of objectTypeIndication in 8.6.6 ofISO/IEC 14496-1.

profileLevelIndication: indication of profile/level of mediaTypeIndication; example values could be those of visualProfileLevelIndication in 8.6.4 ofISO/IEC 14496-1.

compliance: level of compliance to the compression format, i.e, the smallest logical unit in the encrypted bitstream that are correctly recognizable/parsable by the media decoder. For any non-compliant stream, the decryption tool will need to know what marker emulation prevention mechanism was deployed by the encryption tool. The method to emulation prevention is out of scope of this definition. The following values are defined.

Compliance	Semantics
0x00	fully compliant
0x01	compliant to video packet level for video
0x02	compliant to VOP level for video
0x03	non-compliant for video
0x04	Compliant to GOB level for H.263 baseline
0x05-2F	ISO Reserved for video
0x30	compliant to data frame level for AAC audio
0x31	non-compliant for AAC audio
0x32 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

numBufs: number of buffers needed to hold data to be decrypted separately. The cipher text will be de-multiplexed into different buffers for separate decryption.

bufInfoStruct: a structure holding information needed for each buffer

cipher_Id: type of decryption algorithm used, e.g. , DES, 3DES, AES etc., registered through an RA.

syncBoundary: this field provides the tool information on what shall be considered a "unit of cipher-text". Bits from different cipher-text unit are decrypted separately. The following values are defined.

syncBoundary	Semantics
0x00	Video packet for video
0x01	VOP (AU) for video
0x02	GOV for video
0x03-2F	ISO Reserved for video
0x30	Data frame for AAC audio
0x31 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

isBlock: block or stream cipher.

mode: mode of block cipher, e.g., CBC, ECB, CFB, OFB, etc., registered through an RA [Annex AC].

blockSize: block size, in bytes, used for the block cipher

keySize: key size, in bytes, used for the block cipher.

stream_cipher_specific_init_info: normative initialization information if a stream cipher is specified by the cipher_Id.

The above data structures are for representing information pertaining to the buffers used for the decryption process. Below is information pertaining to the fields chosen for decryption.

isContentSpecific: a value of 0b1 indicates that the selective decryption is based on the selection of the fields. Otherwise RLE_Data will specify a collected range of the bitstream selected for decryption, using run length coding of this range information.

numFields: number of fields to be selected for decryption, e.g., MV, DC, DCTsign, Dquant, etc.

fieldStruct: a structure holding information about the fields chosen for decryption

field_Id: index of field based on a predefined list for the given syntax. The following values are defined.

field_Id	Semantics
0x00	Motion vector (MV) for video
0x01	DC coefficients for video
0x02	DCT sign bits for video
0x03	Quantization parameter Dquant for video
0x04	DCT coefficients for video
0x05	All fields ie. All bits in a “unit of cipher text”
0x06-2F	ISO Reserved for video
0x30	Sign bits for AAC audio
0x31	Run-length codewords for AAC audio
0x32	Scale factors for AAC audio
0x32 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

field_scope: represented in three bits, i.e., b2b1b0. A value of 0b1 for b2, b1, and b0 indicates that this field in I, P, and B VOPs, respectively, is selected and put into the buffer that it is associated with.
buf: which buffer this field will be put into.

isMapped: a value of 0b1 indicates that the codewords of a specific field will be mapped, using a mapping table, to an index which is then subject to decryption.

sendMapTable: a value of 0b1 indicates that the mapping table mappingTable will follow. Otherwise, the default mapping of the codeword table defined in the media format definition (e.g. MPEG4 video specification) is used.

sizeMapTable: size, number of entries of the mapping table.

mappingTable: entries of the mapping table. MappingTable[i] indicates that the *ith* codeword in the codeword table defined in the media format definition is mapped to the index of MappingTable[i].

isShuffled: a value of 0b1 indicates the mapped index sequence will be shuffled using a shuffling table specified in Shuffle_Specific_Info.

nSegments: number of disjoint segments that have been generated to signify which segments are selected for decryption

RLE_Data: specifies the number of bits that are to be decrypted or skipped interleavingly, starting from the first segment that is to be decrypted. If the first segment is not to be decrypted, then the value of the first entry shall be zero.

X.3 An example of a selective decryption configuration data (Informative)

One good example of using parametrically configured tools is a configurable selective encryption framework for securing MPEG-4 video content.

It is recognized that for some applications, simplistic, direct application of encryption to content bitstreams poses many problems, most often due to the lack of syntactic structure of the result. The complexity of encrypting the entire content bitstream can also be prohibitive for both very high bit rate content and low power devices. Selective encryption solves both of these problems, the latter due to the reduced complexity associated with only encrypting a portion of the stream and the former by designing the encryption method such that it results in a format compliant, yet encrypted stream. To achieve format compliance, the tool extracts bits from the fields that have been chosen for encryption, concatenates them in an appropriate way, encrypts the concatenation with a chosen cipher appropriate for the application, and then puts the encrypted bits back into their original positions. To maintain compliance, a fixed length index is assigned to each codeword in the VLC code table, and instead of encrypting the code word concatenation, the index concatenation is encrypted, and then the encrypted index concatenation is mapped back to codeword. Any pattern resulting from the encryption of index concatenation can be mapped back to a compliant codeword concatenation. FLC coded fields are treated as special cases of VLC, where the code length does not vary.

In MPEG IPMP extensions, the goal to define a standardized messaging framework between the tools and the terminal provides for an important functionality, namely that of a single configurable tool that could support a wide range of requirements and could be configured to apply only the subset of those that a particular application specified. Using the guidelines for field selection and tools for encrypting VLC in a syntax compliant way, this section illustrates an example of a selective decryption configuration message. It shall be noted that in designing these messages, it is assumed that the framework is intended to be applied to MPEG-4 video.

An example of a message is defined here that could be used to configure a format-compliant, selective decryption tool that implemented the method described above. The for-loops are unrolled, but the for-loop syntax is there so it is clear that repetition of various fields was a result of the logic in the data structures. This particular example message tells the decryption tool that it is working with a compliant bit stream. The tool will need two buffers, both of which will use DES as the decryption algorithm in CBC mode with a block size of 64 and a key length of 64. The fields will be grouped on a video packet basis. The fields involved will be MV, DC, DCT sign, and Dquant. The latter three are inserted into a buffer using the values that they take on in the bit stream, while the MVs are mapped to a set of 64 indices (known by the tool) and those are inserted in a separate buffer.

Syntax field name	Syntactic Value	Semantic value	Bits
mediaTypeExtension	0	ISO/IEC MPEG set	8
mediaTypeIndication	32	Visual ISO/IEC 14496-2	8
(visual)profileLevelIndication	3	Simple Profile @ Level 1	8
compliance	0	Bit-level compliant	8
numBufs	2	2 buffers	8
/* For(i=0;i<numBufs;i++) */			
/* numBufs=2 */			
/* buffer 0: motion vectors */			
cipher_Id	0	DES	128
syncBoundary	0	Video packet	8
isBlock	1	Block cipher	1
if(isBlock==1) {			
mode	1	CBC	8
blockSize	64	64	16
keySize	64	64	16
}			
/* i=1 buffer 1: DCT information buffer */			
cipher_Id			
syncBoundary	0	DES	128
isBlock	0	Video packet	8
if(isBlock==1) {	1	Block cipher	1
mode			
blockSize	1	CBC	8
keySize	64	64	16
}	64	64	16

IsContentSpecific	1	Based on the selection of the fields	1
numFields	4	4 fields for decryption	8
/* For(i=0;i<numFields;i++) *//* numFields = 4 */			
/* i=0, MV */			
field_Id	0	0 (MV)	8
field_scope	2	encrypt MVs for P frames	3
buf	0	0 (mv buff)	8
isMapped	1	mapped	1
isShuffled	0	no shuffle of the index	1
sendMapTable	0	don't send mapping	1
/* i=1, DC */			
field_Id	1	1 (DC)	8
field_scope	6	encrypt DCs for I and P frames	3
buf	1	1 (other buffer)	8
isMapped	0	not mapped	1
/* i=2, DCT sign */			
field_Id	2	2 (DCT sign)	8
field_scope	6	encrypt DCT signs for I and P frames	3
buf	1	1 (other buffer)	8
isMapped	0	not mapped	1
/* i=3, Dquant */			
field_Id	3	3 (Dquant)	8
field_scope	6	encrypt Dquants for I and P frames	
buf	1	1 (other buffer)	8
isMapped	0	not mapped	1

Annex Y
(normative)

Audio Watermarking Configuration and Notification

Y.1 Introduction

This annex details two types of IPMP Data for audio watermarking. The IPMP_AudioWatermarkingInit Data is extended from IPMP_Data_BaseClass. It could be carried in either IPMP_Descriptor or IPMP Stream, and sent to a Watermarking Tool in order to initialize the process of insertion/extraction of the watermarking payload into/from an audio stream. The Watermarking Tool receives the audio stream and in case of watermarking extraction, constructs an IPMP_SendAudioWatermark data, either send it upstream, or informs the terminal with an IPMP_MessageFromTool message carrying the IPMP_SendAudioWatermark data

This Annex provides means for facilitating audio watermarking technologies capable not only for insertion/extraction but also for remarking and perhaps the most important, distinguishing between legal from illegal audio material.

Y.2 IPMP_AudioWatermarkingInit

Y.2.1 Syntax

```
class IPMP_AudioWatermarkingInit extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_AudioWatermarkingInit_tag
{
    bit(8) inputFormat;
    bit(4) requiredOp;
    bit(1) hasOpaqueData;
    const bit(3) reserved = 0b000;

    if (inputFormat == PCM)
    {
        bit(8) nChannels;
        bit(8) bitPerSample;
        bit(32) frequency;
    }

    if ((requiredOp == INSERT_WM)||(requiredOp == REMARK_WM))
    {
        bit(16) wmPayloadLen;
        bit(8) wmPayload[wmPayloadLen];
    }

    if ((requiredOp == EXTRACT_WM)||(requiredOp == DETECT_COMPRESSION))
    {
        bit(16) wmRecipientId;
    }

    if (hasOpaqueData)
    {
        bit(16) opaqueDataSize;
        bit(8) opaqueData[opaqueDataSize];
    }
}
```

Y.2.2 Semantics

The IPMP_AudioWatermarkingInit data delivers to an audio watermarking tool all the information about the characteristics of the audio content, the type of action to be performed on it and, possibly other related proprietary data required by the watermarking tool. Furthermore in case of:

insertion, the watermarking payload to be inserted;

extraction, the ID of the recipient of the watermarking payload is provided;

remarking, the watermarking payload to be inserted;

compression detection, the ID of the recipient of the decision (that compression has taken place or not) is provided.

As an example of *compression detection*, considering that the audio material has been distributed only in a non-compressed legacy (CD) format, thus, if detected that compression took place on it, it will be unauthorised (pirated) content. This implies that the watermarking tool will be able to discriminate compression from other common signal processing manipulations which may also took place in the signal but are not considered prohibited (as expressed by the rights attached to the particular audio stream) such as sample rate conversion, tremble/bass, spatialisation, echo, etc. It should be clear that this is only an example of the possible usages of this flag. Its goal in general is to facilitate means and technologies for distinguishing legal from illegal audio content.

inputFormat	The format of the audio input stream, as indicated in a Table to be maintained by a registration authority. The Table shall contain at least the PCM format, signaled by the value "0x01" and all audio formats indicated in Table 8 "ObjectTypeIndication Values" in [W3C recommendation, Extensible Markup Language (XML) 1.0 (2nd Edition)]

RequiredOp	The operation that the watermarking tool is required to perform on the audio stream. The following values are allowed:
	
	INSERT_WM = 0
	
	EXTRACT_WM=1
	
	REMARK_WM =2
	
	DETECT_COMPRESSION =3
	
	ISO reserved = 4..10
	
	User defined = 11..15

NChannels	the number of channels (1 = mono, 2 = stereo…) of the input stream
Frequency	the number of samples per second (in Hz. e.g. 44100) of the input stream
BitPerSample	the number of bits per sample (e.g. 8, 16) of the input stream

WmPayload	the watermarking payload to be inserted in the audio content
WmRecipientId	the destination tool identified by the IPMP_Descriptor_ID, to which the watermarking payload and compression information must be delivered. A value of 0x00 indicates the terminal.
HasOpaqueData	a flag that indicates if the message also carries opaque data information for the watermarking tool.
OpaqueDataSize	the length of the opaque data field in bytes
OpaqueData	the opaque data field carrying proprietary information to the watermarking tool (e.g. initialization parameters, like specific algorithm id, keys, etc.)

Y.3 IPMP_SendAudioWatermark

Y.3.1 Syntax

```
class IPMP_SendAudioWatermark extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_SendAudioWatermark_tag
{
    bit(2) wm_status;
    bit(2) compression_status;
    bit(1) hasOpaqueData;
    bit(3) reserved = 0b000;
    if (wm_status == WM_PAYLOAD)
    {
        ByteArray payload;
    }
    if (hasOpaqueData)
    {
        ByteArray opaqueData;
    }
}
```

Y.3.2 Semantics

An audio watermarking tool, which has been required to perform payload extraction will construct this IPMP data, and either send it upstream, or wrap this IPMP data in IPMP_MessageFromTool message and send this message to wmRecipientId each time a new watermarking payload is extracted from the audio content.

An audio watermarking tool, which has been required to detect if compression has been taken place on a raw (PCM) audio stream, will construct this IPMP data, and either send it upstream, or wrap this IPMP data in IPMP_DescriptorFromBitstream message and send this message to wmRecipientId each time it detects that the raw audio stream has been either compressed (and as such perhaps has been illegally distributed) or not.

wm_status	the result of the check if watermarking was present. If watermark was detected, then this value also says if the payload extracted is carried inside the message or not. Possible values are listed in Table Y.1 below.
compression_status	the result of the check if the audio stream was compressed. Possible values are listed in Table Y.2 below.
hasOpaqueData	a flag indicating whether this message carries opaque data.
payload	the watermarking payload extracted from the audio content.
opaqueData	opaque data from the Watermarking Tool.

Table Y.1: wm_status

00	WM_PAYLOAD	watermarking was present in the audio stream, payload is carried in the message.
01	WM_NOPAYLOAD	watermarking was present in the audio stream, no payload is carried in the message.
10	NO_WM	watermarking was not present in the audio stream.
11	WM_UNKNOWN	the Watermarking Tool was unable to detect whether watermarking was present in the audio stream or not.

Table Y.2: compression_status

00	COMPRESSION	the audio stream was compressed
01	NO_COMPRESSIO N	the audio stream was not compressed
10	COMP_UNKNOWN	the Watermarking Tool was unable to detect if the audio stream was compressed
11	ISO Reserved	

Annex Z
(normative)

Video Watermarking Configuration and Notification Data

Z.1 Introduction

This annex details two types of IPMP Data for video watermarking. The IPMP_VideoWatermarkingInit Data is extended from IPMP_Data_BaseClass. It could be carried in either IPMP_Descriptor or IPMP Stream, and sent to a Video Watermarking Tool in order to initialize the process of insertion/extraction of the watermarking payload into/from an video stream. The Watermarking Tool receives the video stream and in case of watermarking extraction, constructs an IPMP_SendVideoWatermark data, either send it upstream, or informs the terminal with an IPMP_MessageFromTool message carrying the IPMP_SendAudioWatermark data.

This annex provides means for facilitating video watermarking technologies capable not only for insertion/extraction but also for remarking and perhaps the most important, distinguishing between legal from illegal video material.

Z.2 IPMP_VideoWatermarkingInit

Z.2.1 Syntax

```
class IPMP_VideoWatermarkingInit extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_VideoWatermarkingInit_tag
{
    bit(8) inputFormat;
    bit(4) requiredOp;
    bit(1) hasOpaqueData;
    const bit(3) reserved = 0b000;

    if (inputFormat == YUV)
    {
        bit(16) frame_horizontal_size;
        bit(16) frame_vertical_size;
        bit(8) chroma_format;
    }

    if ((requiredOp == INSERT_WM)||(requiredOp == REMARK_WM))
    {
        bit(16) wmPayloadLen;
        bit(8) wmPayload[wmPayloadLen];
    }

    if ((requiredOp == EXTRACT_WM) ||(requiredOp == DETECT_COMPRESSION))
    {
        bit(16) wmRecipientId;
    }

    if (hasOpaqueData)
    {
        bit(16) opaqueDataSize;
        bit(8) opaqueData[opaqueDataSize];
    }
}
```

Z.2.2 Syntax

The IPMP_VideoWatermarkingInit data delivers to a watermarking tool all the information about the characteristics of the video content, the type of action to be performed on it and, possibly other related proprietary data required by the watermarking tool. Furthermore in case of:

insertion, the watermarking payload to be inserted;

extraction, the ID of the recipient of the watermarking payload is provided;

remarking, the watermarking payload to be inserted;

compression detection, the ID of the recipient of the decision (that compression has taken place or not) is provided.

inputFormat	The format of the video input stream, as indicated in a Table to be maintained by a registration authority. The Table shall contain at least all video formats indicated in Table 8 "ObjectTypeIndication values" in [3]

RequiredOp	The operation that the watermarking tool is required to perform on the audio stream. The following values are allowed:
	INSERT_WM = 0
	EXTRACT_WM = 1
	REMARK_WM = 2
	DETECT_COMPRESSION = 3
	ISO reserved = 4..10
	User defined = 11..15

frame_horizontal_size	Horizontal size of the yuv frame
frame_vertical_size	vertical size of the yuv frame
chroma_format	chroma_format: 0x01=4:2:0, 0x02=4:2:2, 0x03=4:4:4 ISO reserved =0x04..0xA0 User defined = 0xA1..0xFE Forbidden: 0x00, 0xFF
WmPayloadLen	the length of the watermarking payload in bytes to be inserted in the video content.
WmPayload	the watermarking payload to be inserted in the video content
WmRecipientId	the destination tool identified by the IPMP_Descriptor_ID, to which the watermarking payloadand compression information must be delivered. A value of 0x00 indicates the terminal.

HasOpaqueData	a flag that indicates if the message also carries opaque data information for the watermarking tool.
OpaqueDataSize	the length of the opaque data field in bytes
OpaqueData	the opaque data field carrying proprietary information to the watermarking tool (e.g. initialisation parameters, like specific algorithm id, keys, etc.)

Z.3 IPMP_SendVideoWatermark

Z.3.1 Syntax

```
class IPMP_SendVideoWatermark extends IPMP_Data_BaseClass :
    bit(8) tag = IPMP_SendVideoWatermark_tag
{
    bit(2) wm_status;
    bit(2) compression_status;
    bit(1) hasOpaqueData;
    bit(3) reserved = 0b000;
    if (wm_status == WM_PAYLOAD)
    {
        ByteArray payload;
    }
    if  (hasOpaqueData)
    {
        ByteArray opaqueData;
    }
}
```

Z.3.2 Semantics

A video watermarking tool, which has been required to perform payload extraction will construct this IPMP data, and either send it upstream, or wrap this IPMP data in IPMP_MessageFromTool message and send this message to wmRecipientId each time a new watermarking payload is extracted from the video content.

A video watermarking tool, which has been required to detect if compression has been taken place on a raw video stream, will construct this IPMP data, and either send it upstream, or wrap this IPMP data in IPMP_MessageFromTool message and send this message to wmRecipientId each time it detects that the raw video stream has been either compressed (and as such perhaps has been illegally distributed) or not.

wm_status	the result of the check if watermarking was present. If watermark was detected, then this value also says if the payload extracted is carried inside the message or not. Possible values are listed in Table Z.1 below.
compression_status	the result of the check if the video stream was compressed. Possible values are listed in Table Z.2 below.
hasOpaqueData	a flag indicating whether this message carries opaque data.
Payload	the watermarking payload extracted from the video content.
OpaqueData	opaque data from the Watermarking Tool.

Table Z.1 — wm_status

00	WM_PAYLOAD	watermarking was present in the video stream, payload is carried in the message.
01	WM_NOPAYLOAD	watermarking was present in the video stream, no payload is carried in the message.
10	NO_WM	watermarking was not present in the video stream.
11	WM_UNKNOWN	the Watermarking Tool was unable to detect whether watermarking was present in the video stream or not.

Table Z.2 — compression_status

00	COMPRESSION	the video stream was compressed
01	NO_COMPRESSION	the video stream was not compressed
10	COMP_UNKNOWN	the Watermarking Tool was unable to detect if the video stream was compressed
11	ISO Reserved	

Annex AA
(normative)

Tool/Content Transfer Messages Among Distributed IPMP Devices

AA.1 Introduction

This annex is normative only for devices that implement IPMP inter-device communication for the transfer of content and/or IPMP tools. This annex provides for an IPMP framework that is supported and operates outside the scope of MPEG-4 Systems.

AA.2 Addressing of distributed devices

To address different IPMP devices in a network domain, every IPMP device shall be assigned with a unique 128 bit deviceID. How the deviceID is assigned and maintained unique is an implementation issue.

AA.3 IPMP_DeviceMessageBase

AA.3.1 Syntax

```
Aligned(8) abstract expandable(2^28-1) class IPMP_DeviceMessageBase: bit(8) tag = 0
{
    bit(8) Version;
    bit(128) sender_deviceID;
    bit(128) recipient_deviceID;
}
```

AA.3.2 Semantics

`IPMP_DeviceMessageBase` is an expandable base class for IPMP Device to Device Messages. The extension tags are defined in Table AA.1.

`Version` indicates the version of syntax used in the messages and shall be set to 0x02.[1)]

`Sender_DeviceID` indicates the device ID of the originator of the message.

`Recipient_DeviceID` indicates the device ID of the intended recipient of the message.

Table AA.1 - Tags for messages extending IPMP_DeviceMessageBase

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_MessageFromDevice_tag
0x02 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

1) Version number 0x01 is reverved for IPMP_ToolMessageBase