INTERNATIONAL STANDARD

ISO/IEC 23006-2

Third edition
2016-12-01

Information technology — Multimedia service platform technologies —

Part 2: MPEG extensible middleware (MXM) API

Technologies de l'information — Technologies de la plate-forme de services multimédia —

Partie 2: Intergiciel MPEG extensible (MXM) API

Reference number
ISO/IEC 23006-2:2016(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC 23006-2:2013), which has been technically revised.

A list of all parts in the ISO/IEC 23006 series can be found on the ISO website.

Introduction

The ISO/IEC 23006 series is a suite of standards that has been developed for the purpose of enabling the easy design and implementation of media-handling value chains whose devices interoperate because they are all based on the same set of technologies, especially MPEG technologies, accessible from the middleware APIs, elementary services and aggregated services.

The ISO/IEC 23006 series is referred to as MPEG Extensible Middleware (MXM) in its first edition, and it specifies an architecture (ISO/IEC 23006-1), an API (ISO/IEC 23006-2), a conformance and reference software (ISO/IEC 23006-3) and a set of protocols which MXM Devices had to adhere (ISO/IEC 23006-4). It specifies also how to combine elementary services into aggregated services (ISO/IEC 23006-5).

The ISO/IEC 23006 series is subdivided into five parts:

Part 1 — Architecture: specifies the architecture that can be used as a guide to an MPEG-M implementation;

Part 2 — MPEG Extensible Middleware (MXM) Application Programming Interface (APIs) (this document): specifies the middleware APIs;

Part 3 — Conformance and Reference Software: specifies conformance criteria and a reference software implementation with a normative value;

Part 4 — Elementary Services: specifies elementary service protocols between MPEG-M applications;

Part 5 — Service Aggregation: specifies mechanisms enabling the combination of Elementary Services and other services to build Aggregated Services.

Information technology — Multimedia service platform technologies —

Part 2: MPEG extensible middleware (MXM) API

1 Scope

This document specifies a set of Application Programming Interfaces (called for short MXM APIs) so that MPEG-M Applications running on an MPEG-M Device can access the standard multimedia technologies contained in its Middleware as MPEG-M Engines, as specified by ISO/IEC 23006-1.

The MXM APIs belong to two classes:

— the MPEG-M Engine APIs, i.e. the collection of the individual MPEG-M Engine APIs providing access to a single MPEG technology (e.g. video coding) or to a group of MPEG technologies where this is convenient;

— the MPEG-M Orchestrator API, i.e. the API of the special MPEG-M Engine (called Orchestrator Engine) that is capable of creating chains of MPEG-M Engines to execute high-level application calls such as “Play a video”, as opposed to the typically low-level MPEG-M Engine API calls.

2 Normative references

There are no normative references in this document.

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

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

— IEC Electropedia: available at http://www.electropedia.org/

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

3.1.1
Aggregated Service
service resulting from the combination of *Elementary Services* (3.1.2)

3.1.2
Elementary Service
basic unit of *service* (3.1.13)

3.1.3
content
Digital Item and its component elements, namely resources (e.g. media, scripts, executable), identifiers, descriptions (e.g. metadata) and *event* (3.1.6) reports

3.1.4
contract
set of metadata, *licenses* (3.1.8), promises and signers agreed by *users* (3.1.15) of a multimedia *value chain* (3.1.16), where a promise is a signed collection of statements about, e.g. obligations, prohibitions and assertions, and a signer is a user whose signature makes the contract valid

3.1.5
device
hardware/software or simply software apparatus that enables a *user* (3.1.15) to play a *role* (3.1.12) in multimedia *value chains* (3.1.16)

3.1.6
event
performance of a specified set of functions or operations

3.1.7
entity
one of the following elements in the multimedia *value chain* (3.1.16): *content* (3.1.3), *contract* (3.1.4), *device* (3.1.5), *event* (3.1.6), *license* (3.1.8), *service* (3.1.13), and *user* (3.1.15)

3.1.8
license
collection of authorizations, conditions and payment terms granted by a *user* (3.1.15) to other users

3.1.9
protocol
set of rules and data format used by two *devices* (3.1.5) to communicate

3.1.10
resource
individually identifiable asset or a sequence of assets

EXAMPLE Video or audio clip, a 3D synthetic scene, an image, a textual asset, a 2D LASeR scene, a web page, a single program or a full 24-hour programming of a TV broadcast, a script or executable, etc.

3.1.11
right
ability of a *user* (3.1.15) to perform an operation in the multimedia *value chain* (3.1.16)

3.1.12
role
ability of a *user* (3.1.15) to perform a set of operations in the multimedia *value chain* (3.1.16)

3.1.13
service
operation performed on an *entity* (3.1.7) by a *user* (3.1.15) on behalf of other users

3.1.14
service provider
user (3.1.15) offering *services* (3.1.13) to other users

3.1.15
user
participant in multimedia *value chains* (3.1.16)

3.1.16
value chain
collection of *users* (3.1.15), including creators, end users and *service providers* (3.1.14), that conform to this document

3.1.17
MPEG-M Application
application that runs on an *MPEG-M device* (3.1.18) and makes calls to the MPEG-M Application API and *MPEG-M Engine APIs* (3.1.20)

3.1.18
MPEG-M Device
device (3.1.5) equipped with a selected set of *MPEG-M Engines* (3.1.19)

3.1.19
MPEG-M Engine
collection of specific technologies that are bundled together to provide a specific functionality that is needed by *MPEG-M Applications* (3.1.17)

3.1.20
MPEG-M Engine API
API of a single *MPEG-M Engine* (3.1.19)

3.1.21
MPEG-M orchestrator API
API of the *MPEG-M Orchestrator Engine* (3.1.22)

3.1.22
MPEG-M Orchestrator Engine
special *MPEG-M Engine* (3.1.19) capable of creating chains of MPEG-M Engines

Note 1 to entry: It is also to set-up a sequence of connected MPEG-M Engines for the purpose of executing a high-level application call such as Play.

3.1.23
MPEG-M Technology
technology that is required to implement an MPEG-M functionality

3.2 Abbreviated terms

AIT	Advanced IPTV Terminal
AS	Aggregated Service
BBL	Bitstream Binding Language
BPMN	Business Process Model and Notation
CEL	Contract Expression Language
DI	Digital Item
DIA	Digital Item Adaptation
DID	Digital Item Declaration
DIDL	Digital Item Declaration Language
DII	Digital Item Identification
DIS	Digital Item Streaming
ER	Event Report
ERR	Event Report Request

ES	Elementary Service
IPMP	Intellectual Property Management and Protection
IPTV	Internet Protocol Television
MDS	Multimedia Description Schemes
MPEG	Moving Picture Experts Group
MPEG-21	Multimedia Framework [see ISO/IEC 21000 (all parts)]
MPEG-A	Multimedia Application Format [see ISO/IEC 23000 (all parts)]
MPEG-M	Multimedia Service Platform Technologies [see ISO/IEC 23006 (all parts)]
MPEG-V	Multimedia Context and Control [see ISO/IEC 23005 (all parts)]
MPQF	MPEG Query Format
REL	Rights Expression Language
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
SE	Sensory Effect
SEM	Sensory Effect Metadata
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

4 Namespace conventions

For clarity, throughout this document, consistent namespace prefixes are used.

"`xml:`" and "`xmlns:`" are normative prefixes defined in W3C XMLNAMES. The prefix "`xml:`" is by definition bound to "http://www.w3.org/XML/1998/namespace". The prefix "`xmlns:`" is used only for namespace bindings and is not itself bound to any namespace name.

"`xsi:`" prefix is not normative. It is a naming convention in this document to refer to an element of the http://www.w3.org/2001/XMLSchema-instance namespace. All other prefixes used in either the text or examples of this document are not normative, e.g. "`mpegm:`", "`dia:`".

In particular, most of the informative examples in this document are provided as XML fragments without the normally required XML document declaration and, thus, miss a correct namespace binding context declaration.

Unless specified otherwise, all unqualified descriptions fragments assume the default namespace "`urn:mpeg:mpegM:schema:02-service-NS:2012`".

In these descriptions fragments the different prefixes are bound to the namespaces as given in Table 1. The schema locations of the namespaces in Table 1 are only an informative indication as schema locations may change over time.

Table 1 — Mapping of prefixes to namespaces used in examples and text

Prefix	Corresponding namespace	Schema location
mpegm	urn:mpeg:mpegM:schema:02-service-NS:2011	
mpegmb	urn:mpeg:mpegM:schema:01-base-NS:2011	
dia	urn:mpeg:mpeg21:2003:01-DIA-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dia-2nd/UED-2nd.xsd
erl	urn:mpeg:mpeg21:2005:01-ERL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/er/er.xsd
fru	urn:mpeg:mpegB:schema:FragmentRequestUnits:2007	Defined in ISO/IEC 23001-2:2008
mpeg7	urn:mpeg:mpeg7:schema:2004	
mpeg7s	urn:mpeg:mpeg7:systems:2001	
cel	urn:mpeg:mpeg21:cel:contract:2011	
bbl	urn:mpeg:mpeg21:2007:01-BBL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dis/bbl.xsd
dii	urn:mpeg:mpeg21:2002:01-DII-NS	
mpqf	urn:mpeg:mpqf:schema:2008	Defined in ISO/IEC 15938-12
mpeg4ipmp	urn:mpeg:mpeg4:IPMPSchema:2002	Defined in ISO/IEC 14496-13:2004
ipmpdidl	urn:mpeg:mpeg21:2004:01-IPMPDIDL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/ipmp/ipmp-didl.xsd
ipmpmsg	urn:mpeg:mpegB:schema:IPMP-XML-MESSAGES:2007	Defined in ISO/IEC 23001-3:2008
ipmpinfo	urn:mpeg:mpeg21:2004:01-IPMPINFO-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/ipmp/ipmp-general.xsd
didl	urn:mpeg:mpeg21:2002:02-DIDL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didl.xsd
mpegm-didl	urn:mpeg:mpegM:schema:06-didl-NS:2012	

Table 1 *(continued)*

Prefix	Corresponding namespace	Schema location
didmodel	urn:mpeg:mpeg21:2002:02-DIDMODEL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didmodel.xsd
didl-msx	urn:mpeg:maf:schema:mediastreaming:DIDLextensions	Defined in ISO/IEC 23000-5:2011
dii	urn:mpeg:mpeg21:2002:01-DII-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dii/dii.xsd
rel-r	urn:mpeg:mpeg21:2003:01-REL-R-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/rel-r/rel-r.xsd
rel-sx	urn:mpeg:mpeg21:2003:01-REL-SX-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/rel-r/rel-sx.xsd
xsd	http://www.w3.org/2001/XMLSchema	http://www.w3.org/2001/XMLSchema.xsd
xsi	http://www.w3.org/2001/XMLSchema-instance	
dsig	http://www.w3.org/2000/09/xmldsig#	http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd
xenc	http://www.w3.org/2001/04/xmlenc#	
bpmn	http://www.omg.org/spec/BPMN/20100524/MODEL	http://www.omg.org/spec/BPMN/20100501/BPMN20.xsd
bpmnext1	urn:mpeg:mpegM:schema:04-bpmn-ext-mfr-NS:2012	
ca	urn:mpeg:mpegM:service-type:04-content-adaptation-NS:2012	
cel	urn:mpeg:mpeg21:2010:01-CEL-NS	
cidl	urn:mpeg:mpeg-v:2010:01-CIDL-NS	
dc	http://purl.org/dc/elements/1.1/	http://dublincore.org/schemas/xmls/qdc/2008/02/11/dc.xsd
dcdv	urn:mpeg:mpeg-v:2010:01-DCDV-NS	
ebucore	urn:ebu:metadata-schema:ebuCore_2010	http://www.ebu.ch/metadata/schemas/EBU-Core/20100820/EBU_CORE_20100820.xsd
esi	urn:mpeg:mpegM:service-type:03-extract-sensory-information-NS:2012	

Table 1 *(continued)*

Prefix	Corresponding namespace	Schema location
etsi	urn:dvb:metadata:iptv:sdns:2008-1	Defined in ETSI TS 102 034[29]
ipmpinfo-msx	urn:mpeg:maf:Schema:mediastreaming:IPMPINFOextensions:2007	Defined in ISO/IEC 23000-5:2011
rs	urn:mpeg:mpegM:service-type:01-recognize-speech-NS:2012	
saml	urn:oasis:names:tc:SAML:2.0:assertion	http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd
samlp	urn:oasis:names:tc:SAML:2.0:protocol	http://docs.oasis-open.org/security/saml/v2.0/saml-schema-protocol-2.0.xsd
sedl	urn:mpeg:mpeg-v:2010:01-SEDL-NS	
sepv	urn:mpeg:mpeg-v:2010:01-SEPV-NS	
sev	urn:mpeg:mpeg-v:2010:01-SEV-NS	
sid	urn:mpeg:mpegM:schema:05-sid-NS:2012	
ss	urn:mpeg:mpegM:service-type:02-synthesize-speech-NS:2012	
tva	urn:tva:metadata:2010	Defined in ETSI TS 102 822-3-1[30]
wsdl	http://www.w3.org/ns/wsdl	http://www.w3.org/2002/ws/desc/ns/wsdl20.xsd
Xsl	http://www.w3.org/1999/XSL/Transform	http://www.w3.org/2007/schema-for-xslt20.xsd
xhtml	http://www.w3.org/1999/xhtml	http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd

5 Common MXM interfaces and classes

The MXM APIs are specified in the Java languages, as follows:

a) a high-level description of the interfaces defining the MXM APIs, provided in the current text;

b) an html format specification of the MXM APIs with a normative value provided as an attachment to this document, provided in an attachment or on the official MPEG-M website [6].

The core part of MXM consists of a number of interfaces and classes which are common to all MPEG-M Engines. These are:

— MXM: the main class that enables MPEG-M Applications to obtain instances of MPEG-M Engines. It acts as a factory instantiating the MPEG-M Engines listed in a configuration file (so-called MXM Configuration File). The MXM Configuration File contains the list of MPEG-M Engines that are required by an MPEG-M Application; hence, each MPEG-M Application needs to have an MXM Configuration File (the format of the MXM Configuration file is depicted in Annex A);

— MXMAbstractEngine: the abstract class at the highest level of an MPEG-M Engine. Every MPEG-M Engines has to extend this abstract class;

— MXMObject: the basic interface for most of the MXM classes. MXMObject is a wrapper of any objects exchanged by MPEG-M Engines. It defines the version of MXM and it provides a number of fundamental methods used to extract the actual class name and type of the object wrapped in the MXMObject. Furthermore, it provides getters and setters for wrapping/unwrapping an object into/from an MXMObject;

— MXMAdapter: the basic class implementing the MXMObject interface. By means of an MXMAdapter, it is possible to convert any object into an MXMObject;

— MXMEngineName: the enumeration listing all possible names (types) of MPEG-M Engines, so that any entity can be unambiguously identified;

— MXMException: the highest level of exception thrown by MXM. All other exceptions in MXM have to extend this abstract class;

— MXMInvalidConfigurationException: specialized exception informing an MPEG-M Application that the MXM Configuration File is trying to use non-compliant MXM Configuration Schema specifications;

— MXMEngineResponse: the enumeration used as a general-purpose return value for a number of methods;

— MXMCreatorAndParser: the interface that extends the MXMCreator and MXMParser interfaces which specify a set of common methods to be implemented by all classes employed to create and parse data managed by the entire MXM middleware;

— MXMKeys: the class that wraps a key-value construct used to specify an MPEG-M Engine's parameter.

6 MPEG-M Engines

6.1 General

Throughout Clause 6, the technologies supported by each MPEG-M Engines are detailed. If a specific profile has not been indicated, it implies that the full standard is supported. Moreover, the APIs of the aforementioned engines are specified. These APIs have been designed in such a way that any MPEG-M Application can rely on top of them, independently of their implementation. MPEG-M Part 3 provides a reference implementation of these MPEG-M Engines; however, this implementation may vary. The only requirement is that all implementations have to produce the same results.

The MPEG-M Engine APIs can be divided in two subclasses:

— APIs that represent particular data the MPEG-M Engine is specialized on (called "schema handler APIs");

— APIs that provide some functionality on the data described above (called "functional APIs").

The MPEG-M Engines, as already depicted in MPEG-M Part 1, are subdivided in MPEG-M Technology Engines and MPEG-M Protocol Engines. The difference between them is the way they are acceded: locally for MPEG-M Technology Engines, remotely in the case of MPEG-M Protocol Engines. A special case of MPEG-M Technologies Engines are the MPEG-M Orchestrator Engines, which are supposed to deal with several and different MPEG-M Engines.

6.2 Technology Engines

6.2.1 Digital Item Engine

6.2.1.1 Technologies

The Digital Item Engine provides access to the technologies specified in Table 2.

Table 2 — Technologies supported by the Digital Item Engine

Standard	Profile or technology	Reference
Digital Item Declaration	Main	ISO/IEC 21000-2
	MSAF	ISO/IEC 23000-7
Digital Item Identification		ISO/IEC 21000-3

6.2.1.2 APIs

The Digital Item Engine interface defines the APIs for handling ISO/IEC 21000-2 and ISO/IEC 23000-7 Digital Item Declaration (DID) data structures. Classes implementing the Digital Item Engine interface shall provide the necessary functionality to

— create Digital Items and retrieve data from them;

— be able to manage:

  — Item, Statement, Descriptor from Digital Items;

  — Component, Resource from Digital Items;

  — License, Metadata, ERRs from Item in Digital Items.

6.2.2 MPEG-21 File Format Engine

6.2.2.1 Technologies

The MPEG-21 File Format Engine provides access to the technologies specified in Table 3.

Table 3 — Technologies supported by the MPEG-21 File Format Engine

Standard	Profile or technology	Reference
MPEG-21 File Format		ISO/IEC 21000-9

6.2.2.2 APIs

6.2.2.2.1 General

The MPEG-21 File Format Engine interface defines the methods for operating over ISO/IEC 21000-9 MPEG-21 File Format files. Classes implementing the MPEG-21 File Format Engine interface shall provide the necessary functionality to

— create an MPEG-21 file, and

— access data of an MPEG-21 File.

6.2.2.2.2 MPEG21 File Creation

Creating an MPEG-21 File involves the following interfaces:

— MPEG21FileCreator: an interface defining the methods to create an MPEG-21 file.

6.2.2.2.3 MP21 File Access

Accessing an MPEG-21 File involves the following interfaces:

— MPEG21FileReader: an interface defining the methods to access an MPEG-21 file.

6.2.3 REL Engine

6.2.3.1 Technologies

The REL Engine provides access to the technologies specified in Table 4.

Table 4 — Technologies supported by the REL Engine

Standard	Profile or technology	Reference
Rights Expression Language	MAM profile	ISO/IEC 21000-5
	DAC profile	ISO/IEC 21000-5
	OAC profile	ISO/IEC 21000-5

6.2.3.2 APIs

The REL Engine interface defines the methods for handling ISO/IEC 21000-5 Rights Expression Language (REL) data structures. Classes implementing the REL Engine interface shall provide the following functionality:

— create Rights Expressions and access data contained in them;

— authorize users to exercise rights.

6.2.3.3 Rights Expression creation and access

6.2.3.3.1 General

Creating and accessing a REL statement involves the following interfaces:

— License: an interface defining the methods to create and parse an r:license element;

— Grant: an interface defining the methods to create and parse an r:grant;

— DigitalResource: an interface defining the methods to create and parse an r:digitalResource;

— ProtectedResource: an interface defining the methods to create and parse an m1x:protectedResource;

— IdentityHolder: an interface defining the methods to create and parse an m1x:identityHolder;

— Issuer: an interface defining the methods to create and parse an r:issuer;

— KeyHolder: an interface defining the methods to create and parse an r:keyHolder.

6.2.3.3.2 Authorization

Authorizing a user to exercise a right involves the following interfaces:

— AuthorisationManager: an interface defining the methods to authorize a user to exercise a right and retrieve information from the validation operation;

— AuthorisationResult: an enumeration defining possible result of an authorization.

6.2.4 IPMP Engine

6.2.4.1 Technologies

The IPMP Engine provides access to the technologies specified in Table 5.

Table 5 — Technologies supported by the IPMP Engine

Standard	Profile or technology	Reference
MPEG-21 IPMP Components	Main	ISO/IEC 21000-4
	MSAF	ISO/IEC 23000-5
IPMP XML Messages		ISO/IEC 23001-3

6.2.4.2 APIs

6.2.4.2.1 General

The IPMP Engine interface defines the methods for operating over ISO/IEC 21000-4 and ISO/IEC 23000-5 Intellectual Property Management and Protection data structures. Classes implementing the IPMP Engine interface shall provide the following functionality:

— create IPMP data structures and access data contained in them.

6.2.4.2.2 IPMP Information creation and access

Creating and accessing IPMP data structures involves the following interfaces:

— IPMPGeneralInfoDescriptor: an interface defining the methods to create and parse an ipmpinfo:IPMPGeneralInfoDescriptor element;

— IPMPInfoDescriptor: an interface defining the methods to create and parse an ipmpinfo:IPMPInfoDescriptor element;

— ProtectedAsset: an interface defining the methods to create and parse an ipmpdidl:ProtectedAsset element;

— RightsDescriptor: an interface defining the methods to create and parse an ipmpinfo:RightsDescriptor element;

— Tool: an interface defining the methods to create and parse an ipmpinfo:Tool element.

6.2.5 GreenMetadata Engine

6.2.5.1 Technologies

The GreenMetadata Engine provides access to the technologies specified in Table 6.

Table 6 — Technologies supported by the Green Metadata Engine

Standard	Profile or technology	Reference
Green Metadata		ISO/IEC 23001-11

6.2.5.2 APIs

The Green Metadata TE (Encoder) exposes APIs that are called by the Media Pre-Processor and/or the Media Encoder to enable the Green Metadata Generator (Encoder) to produce appropriate metadata. The APIs are called in realtime during pre-processing/encoding. The metadata is used for decoder and display power reduction, quality recovery and for energy-efficient media selection.

The Green Metadata TE (Decoder) exposes APIs through which it can receive the Green Metadata stream. The Green Metadata Extractor (Decoder) receives the Green Metadata stream and feeds it to the Power Optimization Module (Decoder). The metadata is used by the power optimization module to reduce decoder and display power.

6.2.6 Media Framework Engine

6.2.6.1 Technologies

6.2.6.1.1 General

The Media Framework Engine has to provide access to, at least, the technologies specified in Table 7 to Table 10.

6.2.6.1.2 Audio

Table 7 — Audio technologies supported by the Media Framework Engine

Standard	Profile or technology	Reference
MPEG-1 Audio	layer II	ISO/IEC 11172-3
MPEG-1 Audio	layer III	ISO/IEC 11172-3
MPEG-4 AAC		ISO/IEC 14496-3
MPEG-4 HE-AAC		ISO/IEC 14496-3

6.2.6.1.3 Video

Table 8 — Video technologies supported by the Media Framework Engine

Standard	Profile or technology	Reference
MPEG-1 Video		ISO/IEC 11172-2
MPEG-2 Video	Main Profile	ISO/IEC 13818-2
MPEG-4 Visual	Simple Profile	ISO/IEC 14496-2
	Advanced Simple Profile	ISO/IEC 14496-2
MPEG-4 AVC		ISO/IEC 14496-10

6.2.6.1.4 Reconfigurable video

Table 9 — Reconfigurable video technologies supported by the Media Framework Engine

Standard	Profile or technology	Reference
Codec Configuration Representation		ISO/IEC 23001-4
Video tool library		ISO/IEC 23002-4

6.2.6.1.5 Muxed content

Table 10 — Muxed content technologies supported by the Media Framework Engine

Standard	Profile or technology	Reference
MPEG-1 Systems		ISO/IEC 11172-1
MPEG-2 Systems	Transport Stream	ISO/IEC 13818-1
	Program Stream	ISO/IEC 13818-1
MPEG-4 M4Mux		ISO/IEC 14496-1
ISO Base Media File Format		ISO/IEC 14496-12

Figure 1 shows the various components of the MPEG-M Media Framework engine at a high level.

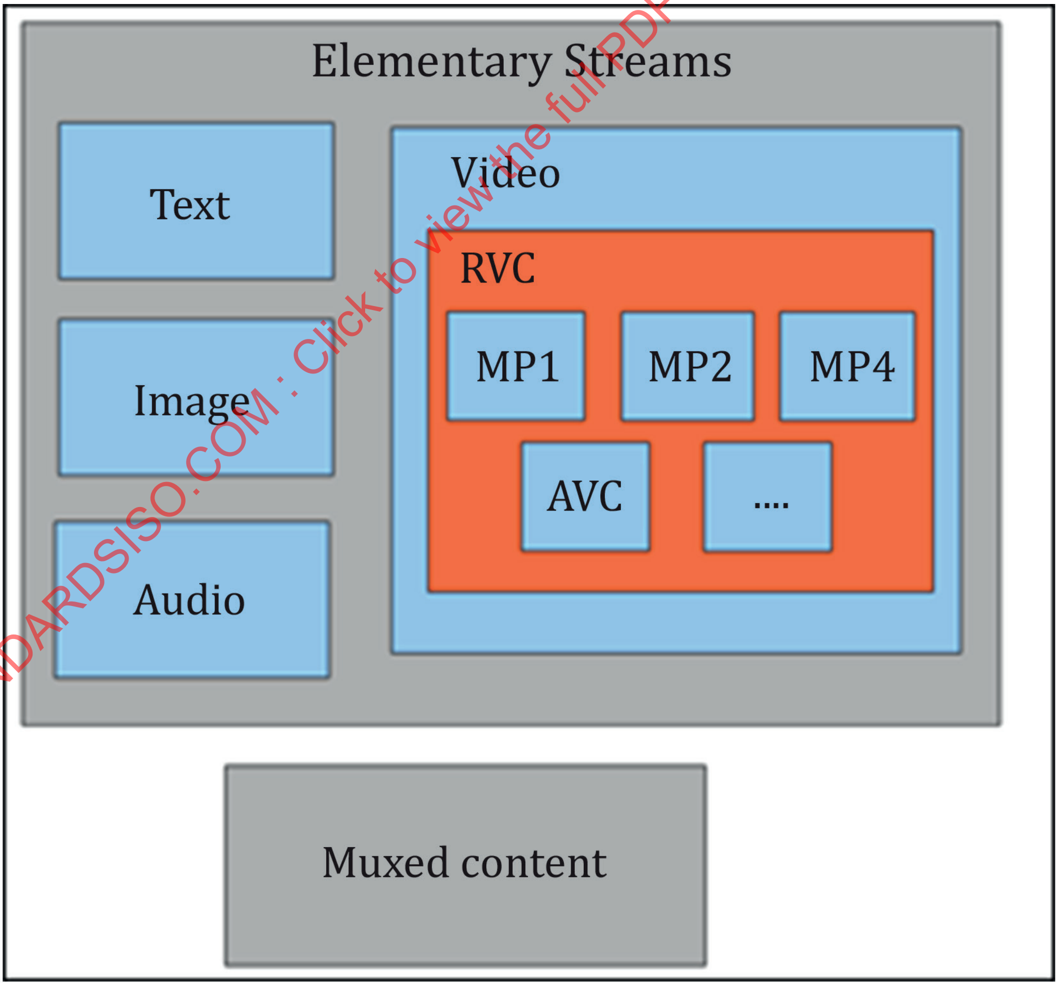

Figure 1 — Technologies of the MPEG-M Media Framework Engine

6.2.6.2 APIs

The MediaFramework is a high level MPEG-M Engine, grouping together several media specific engines such as Video, Image and Audio Engine. It also implements common functionalities (independent on the media type) such as resource loading and saving.

The MediaFrameworkEngine holds three main interfaces:

— an interface for accessing the Content (called AccessMedia);

— an interface for creating the Content (called CreationMedia);

— an interface to communicate with Green Metadata engine.

A typical implementation of the AccessMedia interface of the MediaFrameworkEngine first loads a resource, demultiplex it, check the type of the elementary streams within the resource and call the associated elementary stream access engines.

A typical implementation of the CreationMedia interface of the MediaFrameworkEngine call the associated elementary stream creation engines, initialize them with encoding parameters and finally save the multiplexed resource.

Table 11 shows which class of APIs are provided in the second part of this document for the various media categories.

Table 11 — MXM Media Framework APIs

Class	Type		Creation	Decoding	Rendering
Elementary	Image		Yes, only if it is a JPEG	Yes	Yes, only if it is a texture
	Audio				Yes
	Video	Traditional (MPEG-1, MPEG-2, etc.)			Yes
		RVC		Yes	Yes
Muxed content			Yes	Yes	

The Media Framework Engine acts like a factory that returns an instance of the AccessMediaFramework or CreationMediaFramework interface.

a) public AccessMediaFramework getAccessMediaFramework(Canvas canvas) throws MediaFrameworkEngineException;

b) public CreationMediaFramework getCreationMediaFramework() throws MediaFrameworkEngineException.

In the first case, an optional java.awt.Canvas can be passed to specify where the MediaFramework have to render the resource.

The AccessMediaFramework holds the following methods:

— boolean consumeResource(List<MXMObject> tools, URI inputLocationResource, String mediaType, OutputStream os, MXMObject useType);

  This method is used to consume a given resource.

  — List<MXMObject> tools: one or more IPMPTools (or other kind of tools) used to manipulate the resource (e.g. ciphering, filtering) and defining;

  — control points involved;

   - name or ID of the tool;
   - Tool messages (IPMPMessage) that could contains one or a set of keys;
   - URI inputLocationResource: where the resource handled is located;
   - String mediaType: what kind of media is handled;
   - OutputStream os: where the resource will be eventually stored;
   - MXMObject useType: the type of interaction the user requested on the passed resource;

- void open(MXMObject media);

  This method is a high-level API that is used to prepare a resource (passed as parameter in a MXMObject) to be rendered.

- Void play(), void pause(), void stop(), void release(), void setVolume(int newVolume);

  These methods are used to manage the media chain created to render the resource. It's possible to start, to pause, to stop the rendering of the passed resource; to release the allocated resource and to set the volume of audio stream inside the resource.

- CaptureFrameFromMedia: return an image captured from media.

The CreationMediaFramework holds the following methods:

- boolean createResource(List<MXMObject> tools, URI locationResource, URI newLocationResource).

  This method is used to create and handle a given resource:

  1) List<MXMObject> tools: one or more IPMPTools (or other kind of tools) used to manipulate the resource (e.g. ciphering, filtering) and defining;
  2) control points involved;
  3) name or ID of the tool;
  4) tool messages (IPMPMessage) that could contain one or a set of keys;
  5) URI locationResource: where the resource handled is located;
  6) URI newLocationResource: where the resource handled will be located.

GreenMediaFramework specifies a set of APIs that MediaFramework, on both decode and encoder sides, should support so that an external component can use them in order to reduce the power consumption required to receive, decode and display the media. It involves the following subinterfaces.

- GreenMediaEncoderFramework: This interface specifies API useful to instruct the Media Framework to encode media in such a way that the power required for decoding respects the capabilities and power condition of the decoding device. This is done by allowing external components (mainly the GreenMetadataManager) to change resolution, framerate and complexity reduction scaling. To implement a MediaFramework that supports such functionality and the semantics of the relevant parameters, refer to ISO/IEC 23001-11.

- GreenMediaDecoderFramework: This interface specifies APIs to control the amount of power required by the decoding Media Framework to decode the media. This allows another component (mainly the GreenMetadataManager) to reduce the power consumption rate with respect to power supply conditions (i.e. battery level). Controlled parameters include clock scaling, voltage scaling and display scaling contrast bounds. To implement a MediaFramework that supports such functionality and the semantics of the relevant parameters, refer to ISO/IEC 23001-11.

6.2.7 Metadata Engine

6.2.7.1 Technologies

The Metadata Engine provides access to the technologies specified in Table 12.

Table 12 — Technologies supported by the Metadata Engine

Standard	Profile or technology	Reference
MPEG-7 MDS	Version 3	ISO/IEC 15938-5

6.2.7.2 APIs

6.2.7.2.1 General

The Metadata Engine interface defines the methods for operating over metadata structures. Classes implementing the Metadata Engine interface shall provide the following functionality:

— create metadata structures and access data in them.

6.2.7.2.2 Metadata creation and access

Creating and accessing metadata structures involves the following interfaces:

— Metadata: a superinterface to be further extended, defining a basic method to create and parse a metadata structure;

— Mpeg7: an interface defining the methods to create and parse MPEG-7 metadata structures;

— Abstract: an interface defining the methods to create and parse an mpeg7:Abstract element;

— CreationCoordinates: an interface defining the methods to create and parse an mpeg7:CreationCoordinates element;

— CreationDescription: an interface defining the methods to create and parse an mpeg7:CreationDescription element;

— Creator: an interface defining the methods to create and parse an mpeg7:Creator element;

— Genre: an interface defining the methods to create and parse an mpeg7:Genre element;

— ParentalGuidance: an interface defining the methods to create and parse an mpeg7:Creator element;

— TitleMedia: an interface defining the methods to create and parse an mpeg7:Creator element.

6.2.8 Event Reporting Engine

6.2.8.1 Technologies

The Event Reporting Engine provides access to the technologies specified in Table 13.

Table 13 — Technologies supported by the Event Reporting Engine

Standard	Profile or technology	Reference
Event Reporting		ISO/IEC 21000-15

6.2.8.2 APIs

The Event Reporting Engine interface defines the methods for operating over ISO/IEC 21000-15 Event Reporting data structures. Classes implementing the request event engine interface implement the methods allowing performing the following functionalities:

— Create Event Report Requests: provides the capability to create well-formed ERR objects. An instance of ERR object is needed to parse/create a ERR xml fragment. An Event Report Request (ERR) is used to define the conditions (predicates) under which an Event is deemed to have occurred. Events defined by ERRs trigger the creation of an associated Event Report (ER), which contains information describing the Event, as specified in the associated ERR;

— Create Event Reports: provides the capability to create well-formed ER objects. An instance of ER object is needed to parse/create an ER xml fragment;

— Store Event Reports: capability to store Event Reports to a local repository;

— Store Event Report Requests: capability to store Event Report Requests to a local repository;

— Load Event Requests: capability to load Event Reports from a local repository;

— Load Event Report Requests: capability to load Event Report Requests from a local repository;

— Send Event Reports to the specified recipients: Upon the Event occurring, an Event Report may be generated and delivered to the specified recipient(s);

— Receive Event Reports from another peer: ER objects can be received from other Peers and they contain an ERR inside them.

6.2.9 Security Engine

6.2.9.1 Technologies

The Security Engine provides access to a number of technologies that are not specified by ISO/IEC JTC 1/SC 29, although some of them are specified by external organizations as reported in Table 14. The Security Engine provides MPEG-M Applications with an API to access security technologies and functionalities which are required when dealing with governed or protected content or when required to operate in a trusted environment.

Table 14 — Technologies supported by the Security Engine

Standard	Profile or technology	Reference
XML-Encryption Syntax and Processing		See Reference [27]
XML-Signature Syntax and Processing		See Reference [28]
Trusted Software Stack		See Reference [31]

6.2.9.2 APIs

6.2.9.2.1 General

The Security Engine interface defines security-related methods. Classes implementing the Security Engine interface implement methods providing the following functionalities:

— classes to create new credentials and manage public-key based certificates;

— classes to generate symmetric keys and encrypt/decrypt data;

— classes to store confidential information such as licenses and keys in the secure repository;

— classes to certify the integrity of MXM tools;

— classes to enable complex authentication protocols.

6.2.9.2.2 CertificateManager

The certificate manager involves the following interfaces:

— CertificateManager, to generate private/public keys, import and export of public keys and certificates in various formats, perform cryptographic services, secure storage and retrieval of information, generation of keys, signature calculation and validation, etc.

6.2.9.2.3 KeyManager

The key manager involves the following interfaces:

— KeyManager, to generate symmetric keys, providing generation of keys, hashes, signature calculation and validation, etc.

6.2.9.2.4 SecureDeviceManager

The Secure Device Manager involves the following interfaces:

— SecureDeviceManager, to certify and verify the integrity of MXM Devices, requesting the calculation of a fingerprint of the device with the hardware software installed, and the verification of these values.

6.2.9.2.5 SecureRepositoryManager

The Secure Repository Manager involves the following interfaces:

— SecureRepositoryManager, to store, retrieve and manage confidential information in the secure repository.

6.2.9.2.6 SignatureManager

The SignatureManager involves the following interfaces:

— SignatureManager, to generate and verify hash and signatures.

6.2.9.2.7 AuthenticationManager

The AuthenticationManager involves the following interfaces:

— AuthenticationManager, to generate and handle challenges for challenge-response authentication protocols.

6.2.10 Search Engine

6.2.10.1 General

The Search Engine provides access to the technologies specified in Table 15.

Table 15 — Technologies supported by the Search Engine

Standard	Profile or technology	Reference
MPEG Query Format		ISO/IEC 15938-12

6.2.10.2 APIs

6.2.10.2.1 General

The Search Engine interface defines the methods for operating over metadata structures. Classes implementing the Search Engine interface shall provide the following functionality:

— create and parse MPQF query structures.

6.2.10.2.2 MPQF query creation and access

Creating and accessing MPQF query structures involves the following interfaces:

- MPEGQuery: The basic interface that holds the MPQF query with the inputQuery and outputQuery;
- InputQuery: the basic interface that holds the conditions of the query;
- QueryCondition: an interface for setting all the conditions of the query that need to be evaluated;
- OutputDescription: an interface that defines the structure of the requested output results;
- OutputQuery: the basic interface that holds the results of a query evaluation;
- BooleanExpression: a superinterface for all query conditions. These include:
  - AND/OR/NOT/XOR: interfaces for logical operators;
  - ComparisonExpression: a superinterface for all the comparison expressions. These include:
    - Equal/NotEqual/LessThan/GreaterThan/Contains: Interfaces for comparison operators;
  - SemanticExpression: a superinterface for all semantic expressions. These include:
    - SemanticRelation: an interface for stating triple patterns;
    - InverseOf/ComplementOf/EquivalentClass: interfaces for semantic operators;
  - Query: a superinterface for all the supported query types. These include:
    - QueryByXQuery: an interface for stating an XQuery query;
    - QueryByDescription: an interface for stating the description of the resource;
    - QueryBySPARQL: an interface for defining a SPARQL query;
    - QueryByFreeText: an interface for defining free text conditions by field;
    - QueryByFeatureRange: an interface for defining range conditions by feature;
    - QueryByMedia: an interface for setting similarity conditions with a media;
  - ArithmeticExpression: a superinterface for all arithmetic functions. These include:
    - Add/Subtract/Divide/Abs: interfaces for arithmetic operators;
  - AggregateExpression: a superinterface for all aggregate functions. These include:
    - MIN/MAX/SUM/AVG: interfaces for aggregation operators.

6.2.11 Contract Engine

6.2.11.1 Technologies

The Contract Engine provides access to the technologies specified in Table 16.

Table 16 — Technologies supported by the Contract Engine

Standard	Profile or technology	Reference
Contract Expression Language		ISO/IEC 21000-20
Media Contract Ontology		ISO/IEC 21000-21

6.2.11.2 APIs

Contract Engine interface defines the methods for handling ISO/IEC 21000-20 Contract Expression Language (CEL) and ISO/IEC 21000-21 Media Contract Ontology (MCO) data structures. Classes implementing the Contract Engine interface shall provide the following functionality:

— classes to create Contract Expressions and access data contained in them.

6.2.12 Overlay Engine

6.2.12.1 General

The Overlay Engine specifies a minimum set of interfaces that should be implemented by any device participating in a content delivery network, with emphasis in peer-to-peer networks.

6.2.12.2 APIs

Overlay Engine specifies interfaces for the schema handler that are used for creating and parsing the data units needed by the engine and the interfaces for the technology handler that are needed for setting up the service daemon, propagating and storing messages in the devices.

— OverlayEngine: interface acting as entry point of the Overlay Technology Engine, providing access to the schema and to the technology handler.

— OverlayEngineSchemaHandler: abstract class that provides access to the APIs related to the schema of the registry and the messages of the Overlay Engine.

— OverlayMessage: interface that defines the methods for creating and parsing a general message of an Overlay Engine implementation.

— OverlayMessageHeader: interface that defines the methods for creating and parsing the header part of the Overlay Engine message.

— OverlayMessagePayload: interface that defines the methods for creating and parsing the payload part of the Overlay Engine message.

— OverlayRegistry: interface that defines the methods for creating and parsing the registry of the Overlay Engine.

— OverlayEngineTechnologyHandler: abstract class that provides access to the APIs related to the technologies used by the Overlay Engine.

— OverlayMessageHandler: interface that defines the methods for sending and receiving messages of type OverlayMessage.

— OverlayMessageRepository: interface that defines the methods for storing and retrieving Overlay messages.

— OverlayRegistryRepository: interface that defines the methods for storing and accessing the registry of the Overlay Engine.

6.2.13 Sensory Effect Engine

6.2.13.1 Technologies

The SE (Sensory Effect) Engine provides access to the technologies specified in Table 17.

Table 17 — Technologies supported by the Sensory Effect Engine

Standard	Profile or technology	Reference
Control Information		ISO/IEC 23005-2
Sensory Information		ISO/IEC 23005-3
Data Formats for interactive devices		ISO/IEC 23005-5

6.2.13.2 APIs

6.2.13.2.1 General

The SE engine interface defines methods used for operations over ISO/IEC 23005-2, ISO/IEC 23005-3, and ISO/IEC 23005-5. Classes implementing the SEEngine interface act as factories that create instances of classes which perform the following functions:

— SE manipulation to manipulate a Sensory Effect Metadata (SEM) structure;

— SE retrieval to retrieve effects by an identifier or an object;

— SE adaptation to get device capabilities, user preferences, sensor capabilities and sensed data;

— SE actuation to set device commands of the actuator.

6.2.13.2.2 SE Manipulation

Accessing SEM structures involves the following interfaces:

— InitSEEngine: defines the method of initializing a SE engine;

— ParseSEM: defines the method of parsing SEM structures;

— InsertEffects: defines the method of storing parsed SEM parameters in memory or storage;

— DeleteEffects: defines the method of removing SEM parameters stored in memory or storage;

— UpdateEffects: defines the method of updating SEM parameters stored in memory or storage.

6.2.13.2.3 SE Retrieval

Retrieving SEM structures involves the following interfaces:

— RetrieveEffectsById: defines the method of retrieving SEMs from memory or storage by a SEM identifier;

— RetrieveEffectsByType: defines the method of retrieving SEMs from memory or storage by a type;

— RetrieveEffectsByFilter: defines the method of retrieving SEMs from memory or storage by filter (filter elements can be multiple attributes of an object, e.g. we can set three filter elements of fade, intensity-value and id attributes);

— RetrieveEffectsByComparison: defines the method of retrieving SEMs from memory or storage by comparison (in this case, object can be retrieved by conditional comparisons such as greater than, less than, greater than or equal to, less than or equal to, not equal to, not less than, not greater than, etc.).

6.2.13.2.4 SE Adaptation

Getting device capabilities, user preferences, sensor capabilities and sensed data involves the following interfaces:

— GetDeviceCapability: defines the method of getting the information of actuator device capability;

— GetUserPreference: defines the method of getting the information of user preferences;

— GetSensorCapability: defines the method of getting the information of sensor capabilities;

— GetSensedData: defines the method of getting data detected by sensors.

6.2.13.2.5 SE Actuation

Mapping SEM elements to device commands based upon device capabilities, user preferences, sensor capabilities and sensed data involves the following interfaces:

— SetDeviceCommand: defines the method of setting the control commands of a device.

6.2.14 Compact Descriptor for Visual Search Technology Engine

6.2.14.1 Technologies

The Compact Descriptors for Visual Search (CDVS) Engine provides access to the technologies specified in Table 18.

Table 18 — Technologies supported by the Compact Descriptors Engine

Standard	Profile or technology	Reference
Compact Descriptors for Visual Search		ISO/IEC 15938-13

6.2.14.2 APIs

6.2.14.2.1 General

The Compact Descriptors interface defines the APIs for handling ISO/IEC 15938-13 Compact Descriptors for Visual Search (CDVS) data. Classes implementing this interface shall provide the necessary functionality to

— create CDVS Descriptor and access data contained in them,

— check the conformance of the CDVS Descriptor Data.

6.2.14.2.2 CDVS descriptor creation and access

Creating and accessing a CDVS descriptor involves the following interfaces:

— CDVSDescriptor: an interface defining the methods to create and read CDVS descriptor.

6.2.14.2.3 Check Conformance

Check the conformance of the CDVS Descriptor Data involves the following interfaces:

— CheckConformanceManager: an interface defining the methods to check the conformance of the CDVS Descriptor Data;

— CheckConformanceResult: an enumeration defining possible result of check.

6.3 Protocol Engines

6.3.1 General

The MPEG-M Protocol Engines have a one-to-one correspondence to the MPEG-M Elementary Services specified in ISO/IEC 23006-4 and they are used for

— creating and parsing the message format defined in the corresponding Elementary Service (schema handler),

— making a call (remote or local) to the Elementary Service (technology handler).

In particular, the Protocol Engines technology handler comprises two modules: the local and the remote. The former is used for making a local call from a higher layer (e.g. application) to the middleware and the latter for making a remote call to the Elementary Service. Both these modules have the same API for transparency reason; then, depending on the implementation that each middleware has, the application can make the call to the local or the remote module using the same request and response.

So far, Elementary Service got the form <*Action*><*Entity*> (such as *CreateContent*), the corresponding Protocol Engine will have the following specification:

<*Action*><*Entity*>Engine:

<*Action*><*Entity*>SchemaHandler:

<*Action*><*Entity*>Request extends ProtocolRequest

<*Action*><*Entity*>Response extends ProtocolResponse:

<*Action*><*Entity*>Success extends ProtocolSuccess

<*Action*><*Entity*>Failure extends ProtocolFailure

<*Action*><*Entity*>TechnologyHandler:

<*Action*><*Entity*>Protocol

The ProtocolRequest, ProtocolResponse, ProtocolSuccess and ProtocolFailure types are specified by the Base Protocol Engine (see 6.3.2).

This Protocol Engine structure is not necessarily, fully applicable to all the Protocol Engines. For example, there may be Engines that do not specify their own failure type (such as ActionEntityFailure) but they are using directly the base ProtocolFailure. On the other hand, there may be Engines specifying more types, especially in the schema handler, for creating and parsing other information given in the schema of the corresponding Elementary Service.

6.3.2 Base Protocol Engine

The Base Protocol Engine corresponds to the base protocol specified by ISO/IEC 23006-4. It specifies only schema handler related interfaces, which are used either directly or by being extended by other interfaces, specified in the context of the full Protocol Engines.

In detail, the Base Protocol Engine specifies the following types:

— BaseProtocolEngine: interface acting as entry point of the Base Protocol Engine providing access to the schema handler;

— BaseProtocolSchemaHandler: abstract class acting as entry point for the schema related interfaces of the Base Protocol Engine. It contains getters for all the types that are used in the context of the Base Protocol Engine;

— ApplicationSpecificData: interface used to pass into the protocol request or response any application specific data;

— BasicFailureCode: enumeration that specifies a set of failure codes that are used in some protocol responses in case of failure;

— BasicSuccessCode: enumeration that specifies a set of success codes that are used in some protocol responses in case of success;

— ContentEntity: interface that extends the base entity type and specifies the type that represents content in an MPEG-M value chain. It may contain an MPEG-21 Digital Item, a reference to any kind of content or an identifier of any kind of content.

— ContractEntity: interface that extends the base entity type and specifies the container for an MPEG-21 Contract. Contracts are referred with unique identifiers;

— DeviceEntity: interface that extends the base entity type and specifies the container for referring to a specific MPEG-M Device. This device may be referred either with a reference or an MPEG-21 IPMP Tool Body;

— EntityBase: interface that specifies the base entity type for MPEG-M. It is does not carry any specific data; it is only used as a single point of reference for any other MPEG-M entity, such as ContentEntity, LicenseEntity, ContractEntity, ServiceEntity, UserEntity or entities specified out of the context of MPEG-M by other middlewares extending MXM;

— KeyValueData: interface used by the protocol request and response to store any key-value pairs that may be needed to be exchanged;

— LicenseEntity: interface used as an MPEG-21 REL License container. The license may be included directly or referred with a license identifier or a plain reference;

— ProtocolBase: interface used as the base type for all protocol requests and responses in MPEG-M. It contains information such as the transaction identifier, application specific data, key-value paired data, timestamp and a digital signature to provide the authenticity of the request/response message;

— ProtocolFailure: interface acting as the base type for all protocol failure types. It may contain a result code, that could come from the BasicFailureCode or it could be any other String. Moreover, it includes a display message, in case the failure code is not readable;

— ProtocolRequest: interface that extends the base protocol and is the base type for all requests specified by the Protocol Engines. It contains a SAML assertion that can be used to prove an established user authentication context;

— ProtocolResponse: interface that extends the base protocol and is the base type for all responses specified by the Protocol Engines. It does not contain any additional information rather than what is inherited by the ProtocolBase;

— ProtocolSuccess: interface that is the base type for any successful responses of MPEG-M protocols. It contains a custom display message and a success code that may come from BasicSuccessCode enumeration;

— Revocation: enumeration specifying a set of default revocation reason;

— Service: interface used to represent any service in MPEG-M system. It contains information such as the service identifier, service parameters and service name;

— ServiceEntity: interface acting as a service container and may contain either the service itself, or a reference to that service using a plain link or a service identifier;

— SupportedMetadataSchema: interface that specifies the metadata schema that is supported. This interface is used in protocols such as DescribeEntity for indicating which schemas are supported by the remote server;

— SupportedTransferProtocol: interface that specifies the transfer protocol that is supported for a transaction between two devices;

— UserEntity: interface used to represent a user in MPEG-M ecosystem. The user may either be directly included (using an MPEG-7 description) or referred with a plain reference or a unique identifier.

6.3.3 Authenticate Services APIs

6.3.3.1 Authenticate Content Engine

The Authenticate Content Engine APIs provide interfaces allowing Users to confirm the authenticity of some content represented by a ContentEntity.

6.3.3.2 Authenticate Contract Engine

The Authenticate Contract Engine APIs provide interfaces allowing Users to confirm the identity and signers of a Contract.

6.3.3.3 Authenticate License Engine

The Authenticate License Engine APIs provide interfaces allowing Users to confirm the identity and signers of a License.

6.3.3.4 Authenticate User Engine

The Authenticate User Engine APIs provide interfaces allowing Users to Authenticate Users in an AIT value chain.

6.3.4 Authorize Services APIs

The Authorize User Engine APIs provide interfaces allowing Users to obtain authorization of some usages.

6.3.5 Check With Services APIs

6.3.5.1 Check With Contract Engine

The Check With Contract Engine APIs provide interfaces allowing Users to verify if a usage request matches with the content (e.g. obligations, prohibitions) expressed in a Contract.

6.3.5.2 Check With License Engine

The Check With License Engine APIs allow to obtain authorization of a usage request according to Rights expressed in a License.

6.3.6 Create Services APIs

6.3.6.1 Create Content Engine

The Create Content Engine APIs provide interfaces allowing Users to generate a Content.

6.3.6.2 Create Contract Engine

The Create Contract Engine APIs provide interfaces allowing Users to generate a Contract.

6.3.6.3 Create License Engine

The Create License Engine APIs provide interfaces allowing Users to generate a License.

6.3.7 Deliver Services APIs

6.3.7.1 Deliver Content Engine

The Deliver Content Engine APIs provide interfaces allowing Users to transfer Content between Users.

6.3.7.2 Deliver Contract Engine

The Deliver Contract Engine APIs provide interfaces allowing Users to transfer Contract between Users.

6.3.8 Describe Services APIs

6.3.8.1 Describe Content Engine

The Describe Content Engine APIs provide interfaces allowing Users to associate metadata to a Content.

6.3.8.2 Describe Device Engine

The Describe Device Engine APIs provide interfaces allowing Users to associate metadata to a Device.

6.3.8.3 Describe Service Engine

The Describe Service Engine APIs provide interfaces allowing Users to associate metadata to a Service.

6.3.8.4 Describe User Engine

The Describe User Engine APIs provide interfaces allowing Users to associate metadata to a User.

6.3.9 Identify Services APIs

6.3.9.1 Identify Content Engine

The Identify Content Engine APIs provide interfaces allowing Users to assign Identifiers to a Content.

6.3.9.2 Identify Contract Engine

The Identify Contract Engine APIs provide interfaces allowing Users to assign Identifiers to a Contract.

6.3.9.3 Identify Device Engine

The Identify Device Engine APIs provide interfaces allowing Users to assign Identifiers to a Device.

6.3.9.4 Identify License Engine

The Identify License Engine APIs provide interfaces allowing Users to assign Identifiers to a License.

6.3.9.5 Identify User Engine

The Identify User Engine APIs provide interfaces allowing Users to assign Identifiers to a User.

6.3.10 Negotiate Services APIs

6.3.10.1 Negotiate Contract Engine

The Negotiate Contract Engine APIs provide interfaces allowing Users to achieve an agreement on the terms and conditions of use, e.g. obligations and prohibitions with respect to a Contract.

6.3.10.2 Negotiate License Engine

The Negotiate License Engine APIs provide interfaces allowing Users to achieve an agreement on the terms and conditions of use, e.g. rights and conditions of a License.

6.3.11 Package Services APIs

The Package Content Engine APIs provide interfaces allowing Users to make contents ready for delivery. It defines the following interface allowing the indicated functionalities to be performed:

packageContentProcess: enables the packaging of one or more digital Items, from within a specified DID, into an appropriate format for delivery.

6.3.12 Post Services APIs

The Post Content Engine APIs provide interfaces allowing Users to let other Users access their Content.

6.3.13 Present Services APIs

6.3.13.1 Present Contract Engine

The Present Contract Engine APIs provide interfaces allowing Users to understand Contract.

6.3.13.2 Present License Engine

The Present License Engine APIs provide interfaces allowing Users to understand License.

6.3.14 Process Services APIs

6.3.14.1 Process Content Engine

The Process Content Engine APIs provide interfaces allowing Users to perform operations on Content.

6.3.14.2 Process License Engine

The Process License Engine APIs provide interfaces allowing Users to perform operations on License.

6.3.15 Request Services APIs

6.3.15.1 Request Content Engine

The Request Content Engine APIs provide interfaces allowing Users to request for a Content.

6.3.15.2 Request Contract Engine

The Request Contract Engine APIs provide interfaces allowing Users to request for a Contract.

6.3.15.3 Request Device Engine

The Request Device Engine APIs provide interfaces allowing Users to request for a Device.

6.3.15.4 Request Event Engine

The Request Event Engine APIs provide interfaces allowing Users to request for an Event Report Request or a set of Event Reports.

6.3.15.5 Request License Engine

The Request License Engine APIs provide interfaces allowing Users to request for a License.

6.3.16 Revoke Services APIs

6.3.16.1 Revoke Contract Engine

The Revoke Contract Engine APIs provide interfaces allowing Users to discontinue the validity of a Contract.

6.3.16.2 Revoke License Engine

The Revoke License Engine APIs provide interfaces allowing Users to discontinue the validity of a License.

6.3.16.3 Revoke Content Engine

The Revoke Content Engine API allows users to discontinue the validity of a specific content. It defines the following interfaces allowing the indicated functionalities to be performed:

revokeContentTool: enables the removal (of practically all versions) of a specific DI from the system.

6.3.17 Search Services APIs

6.3.17.1 Search Content Engine

The Search Content Engine APIs provide interfaces allowing Users to locate Contents satisfying some criteria.

6.3.17.2 Search Contract Engine

The Search Contract Engine APIs provide interfaces allowing Users to retrieve Contract satisfying some specified criteria.

6.3.17.3 Search Device Engine

The Search Device Engine APIs provide interfaces allowing Users to locate Devices satisfying some criteria.

6.3.17.4 Search License Engine

The Search License Engine APIs provide interfaces allowing Users to locate Licenses satisfying some criteria.

6.3.17.5 Search Service Engine

The Search Service Engine APIs provide interfaces allowing Users to locate Services satisfying some criteria.

6.3.17.6 Search User Engine

The Search User Engine APIs provide interfaces allowing Users to locate other Users satisfying some criteria.

6.3.18 Store Services APIs

6.3.18.1 Store Content Engine

The Store Content Engine APIs provide interface allowing Users to save a Content for later use.

6.3.18.2 Store Contract Engine

The Store Contract Engine APIs provide interface allowing Users to save a Contract for later use.

6.3.18.3 Store Event Engine

The Store Event Engine APIs provide interface allowing Users to report an Event that has occurred.

6.3.18.4 Store License Engine

The Store License Engine APIs provide interface allowing Users to save a License for later use.

6.3.19 Transact Services APIs

6.3.19.1 Transact Content Engine

The Transact Content Engine APIs provide interfaces allowing Users to interface with Payment and Cashing systems with regard to Contents.

6.3.19.2 Transact License Engine

The Transact License Engine APIs provide interfaces allowing Users to interface with Payment and Cashing systems with regard to Licenses.

6.3.20 Verify Services APIs

6.3.20.1 Verify Contract Engine

The Verify Contract Engine APIs provide interfaces allowing Users to check the integrity of a Contract.

6.3.20.2 Verify Device Engine

The Verify Device Engine APIs provide interfaces allowing Users to check the integrity of a Device.

6.3.20.3 Verify License Engine

The Verify License Engine APIs allow Users to check the integrity of a License in an AIT value chain.

6.4 MXM Orchestrator APIs

6.4.1 General

The Orchestrator Engines are supposed to deal with many and different MPEG-M Engines, so its APIs should be as much extensible and flexible as possible. The APIs should allow instantiation of the Orchestrator Engine by providing an MXM Object initialized with a configuration file containing a list of all MXM Engines that will be managed by Orchestrator Engine implementation.

A typical extension of Orchestrator Engine

— extends the Orchestrator Engine Interface,

— defines custom methods which involve specific MPEG-M Engine.

Then it will be up to the MPEG-M Application to instantiate and implement the latter Orchestrator Engine implementation. The following methods can be used to initialize the Orchestrator Engine:

— MXMEngineResponse init(MXM mxm) throw MXMException;

— MXM mxm: the object which should be instantiated with the file containing the list of the MXM Engine useful for specific application.

6.4.2 DID Engine Orchestrator APIs

The DID Engine Orchestrator APIs specify useful methods to create, access and update a simple version of a Digital Item. It creates a chain of Digital Item TE, Metadata TE, REL TE hiding to the developer of an MPEG-M Application the way to connect and invoke them in the correct way.

6.4.3 Identify Content Engine Orchestrator APIs

The Identify Content Engine Orchestrator APIs specify methods to identify a generic Digital Item remotely. It exposes useful APIs to permit an easy access to this functionality without taking care of the sequence of MPEG-M Engines needed.

6.4.4 Identify User Engine Orchestrator APIs

The Identify User Engine Orchestrator APIs specify high-level methods to identify a User. It exposes useful APIs to permit an easy access to this functionality without taking care of the sequence of MPEG-M Engines needed.

6.4.5 MF Orchestrator Engine APIs

The MF Orchestrator Engine APIs specify high-level methods to renderize a Content contained in a Digital Item. It permits to easily invoke a rendering task on a specified Java graphics area (java.awt.Canvas).

Annex A
(normative)

MXM Configuration

A.1 General

In order to enable dynamic instantiation of MPEG-M Engines, MXM defines a schema (MXM Configuration Schema) used to create XML Configuration Files that specify which MPEG-M Engines shall be loaded (which properties shall they have and other information) on a particular setup of an MPEG-M Application.

A specific MPEG-M Engine, called Orchestrator Engine, is in charge of creating chains of MPEG-M Engines and using them to execute function calls made by MPEG-M Applications. This is achieved by requesting to MXM the reference to instances of the MPEG-M Engines which are required to satisfy the requests from MPEG-M Applications.

A.2 Syntax of the MXM Configuration Schema

The current version of the schema is available on the MPEG-M svn software repository at the folder http://wg11.sc29.org/mxmsvn/repos/JAVA/trunk/mxm-core/src/main/resources/mxmconfiguration.xsd.

The MXM Configuration Schema is reported below for a detailed description:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="org:iso:mpeg:mxm:configuration:schema"
   xmlns:mxm="org:iso:mpeg:mxm:configuration:schema" xmlns="http://www.w3.org/2001/
XMLSchema"
   attributeFormDefault="unqualified">
   <complexType name="MXMConfigurationBaseType" abstract="true" />
   <!-- **************************************************************** -->
   <!-- MXMConfiguration -->
   <!-- **************************************************************** -->
   <element name="MXMConfiguration" type="mxm:MXMConfigurationType" />
   <complexType name="MXMConfigurationType">
      <complexContent>
         <extension base="mxm:MXMConfigurationBaseType">
            <sequence>
             <element name="MXMParameters" type="mxm:MXMParameterType"
                  minOccurs="0" />
            <element ref="mxm:MXMEngine" minOccurs="0" maxOccurs="unbounded"/>
           </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="MXMParameterType">
```

```
      <complexContent>
         <extension base="mxm:MXMGenericParameterType">
            <sequence>
               <element name="MXMEnginesFolder" type="string" minOccurs="0"
                  maxOccurs="unbounded" />
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="MXMGenericParameterType">
      <complexContent>
         <extension base="mxm:MXMConfigurationBaseType">
            <sequence>
               <element name="entry" type="mxm:KeyValueParameterType"
                  minOccurs="0" maxOccurs="unbounded" />
               <element name="ComplexParameter" type="mxm:ComplexParameterType"
                  minOccurs="0" maxOccurs="unbounded" />
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="KeyValueParameterType">
      <simpleContent>
         <extension base="string">
            <attribute name="key" type="string" use="required" />
         </extension>
      </simpleContent>
   </complexType>
   <complexType name="ComplexParameterType">
      <complexContent>
         <extension base="mxm:MXMConfigurationBaseType">
            <sequence>
               <any namespace="##other" processContents="lax"
                    maxOccurs="unbounded" />
            </sequence>
         </extension>
      </complexContent>
```