INTERNATIONAL STANDARD

ISO/IEC 23006-4

Second edition
2013-02-15

Information technology — Multimedia service platform technologies —

Part 4: Elementary services

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

Partie 4: Services élémentaires

Reference number
ISO/IEC 23006-4:2013(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23006-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 23006-4:2010), which has been technically revised.

ISO/IEC 23006 consists of the following parts, under the general title *Information technology — Multimedia service platform technologies*:

— *Part 1: Architecture*

— *Part 2: MPEG extensible middleware (MXM) API*

— *Part 3: Conformance and reference software*

— *Part 4: Elementary services*

— *Part 5: Service aggregation*

Introduction

ISO/IEC 23006 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 accessible from the middleware.

ISO/IEC 23006 is referred as MPEG Extensible Middleware (MXM) in its first edition, and it specifies an architecture (Part 1), an API (Part 2), a reference software (Part 3) and a set of protocols which MXM Devices had to adhere (Part 4).

ISO/IEC 23006 is referred as Multimedia Service Platform Technologies (MSPT) in its second edition, and it conserves the architecture and design philosophy of the first edition, but stressing the Service Oriented Architecture character. It specifies also how to combine elementary services into aggregated services (Part 5).

This second edition has been specified to address the demand of service specification for an advanced IPTV terminal (AIT). The ISO/IEC 23006 suite of standards also aims at leveraging on advanced technologies to bring into IPTV services the buoyancy of new exciting initiatives – sometimes assembling millions of users in a fortnight – that pop up almost every day with new features such as open APIs and the possibility for third parties to provide applications to those APIs.

This enables the development of a global market of:

— MSPT applications that can run on MSPT devices, like Advanced IPTV Terminals (AITs), thanks to the existence of a standard MSPT application API

— MSPT devices executing MSPT applications thanks to the existence of a standard MSPT architecture

— MSPT engines thanks to the existence of standard MSPT architecture and standard APIs

— Innovative business models because of the ease to design and implement media-handling value chains whose devices interoperate because they are all based on the same set of technologies, especially MPEG technologies.

Information technology — Multimedia service platform technologies —

Part 4: Elementary services

1 Scope

This part of ISO/IEC 23006 specifies a set of Elementary Services and protocols enabling distributed applications to exchange information related to content items and parts thereof, including all the necessary Operations on MPEG-related Entities: Content, Contract, Device, Event, License, Service and User. These operations are defined to be the following: Authenticate, Authorize, Check With, Create, Deliver, Describe, Identify, Install, Interact With, Negotiate, Package, Post, Present, Process, Request, Revoke, Search, Store, Transact, Uninstall and Verify. Elementary Services can be combined in well defined sequences to build Aggregated Services, both of them being called in general Multimedia Services. ISO/IEC 23006 (all parts) will be referred to as MPEG-M for short in the text. The Multimedia Services are provided by and consumed by Multimedia Devices in a MSPT ecosystem, an example of which is the Advanced IPTV Terminal.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-13, *Information technology — Coding of audio-visual objects — Part 13: Intellectual Property Management and Protection (IPMP) extensions*

ISO/IEC 15938-1, *Information technology — Multimedia content description interface — Part 1: Systems*

ISO/IEC 15938-5, *Information technology — Multimedia content description interface — Part 5: Multimedia description schemes*

ISO/IEC 15938-12, *Information technology — Multimedia content description interface — Part 12: Query format*

ISO/IEC 21000-2, *Information technology — Multimedia framework (MPEG-21) — Part 2: Digital Item Declaration*

ISO/IEC 21000-3, *Information technology — Multimedia framework (MPEG-21) — Part 3: Digital Item Identification*

ISO/IEC 21000-4, *Information technology — Multimedia framework (MPEG-21) — Part 4: Intellectual Property Management and Protection Components*

ISO/IEC 21000-5, *Information technology — Multimedia framework (MPEG-21) — Part 5: Rights Expression Language*

ISO/IEC 21000-15, *Information technology — Multimedia framework (MPEG-21) — Part 15: Event Reporting*

ISO/IEC 21000-17, *Information technology — Multimedia framework (MPEG-21) — Part 17: Fragment Identification of MPEG Resources*

ISO/IEC 21000-20, *Information technology — Multimedia framework (MPEG-21) — Part 20: Contract Expression Language*

ISO/IEC 23000-5, *Information technology — Multimedia application format (MPEG-A) — Part 5: Media streaming application format*

ISO/IEC 23001-2, *Information technology — MPEG systems technologies — Part 2: Fragment request units*

ISO/IEC 23001-3, *Information technology — MPEG systems technologies — Part 3: XML IPMP messages*

ISO/IEC 23005-2, *Information technology — Media context and control — Part 2: Control Information*

ISO/IEC 23005-3, *Information technology — Media context and control — Part 3: Sensory Information*

ISO/IEC 23006-1, *Information technology — Multimedia service platform technologies — Part 1: Architecture*

ISO/IEC 23006-2, *Information technology — Multimedia service platform technologies — Part 2: MPEG Extensible Middleware (MXM) API*

ISO/IEC 23006-3, *Information technology — Multimedia service platform technologies — Part 3: Reference software*

IETF RFC 2616, *Hypertext Transfer Protocol — HTTP/1.1*, IETF Request For Comments, June 1999

IETF RFC 3614, *A Uniform Resource Name (URN) Namespace for the Motion Picture Experts Group (MPEG)*, IETF Request For Comments, September 2003

OASIS SAML-CORE-2.0-OS, *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS Standard, 15 March 2005

OMG BPMN 2.0, *Business Process Model and Notation (BPMN) Version 2.0*, Object Management Group, January 2011

W3C SOAP, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, W3C Recommendation, 27 April 2007

W3C WSDL, *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Recommendation, 26 June 2007

W3C XML, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, 29 September 2006

W3C XMLNAMES, *Namespaces in XML 1.0 (Second Edition)*, W3C Recommendation, 16 August 2006

W3C XMLSCHEMA, *XML Schema Part 1: Structures Second Edition and XML Schema Part 2: Datatypes Second Edition*, W3C Recommendations, 28 October 2004

W3C XPATH1, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 16 November 1999

W3C XSL, *XSL Transformations (XSLT) Version 2.0*, W3C Recommendation, 23 January 2007

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

3.1.1
Aggregated Service
service resulting from the combination of **Elementary Service**s

3.1.2
Elementary Service
basic unit of **service**

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

3.1.4
contract
set of **metadata**, **licenses**, promises and signers agreed by **Users** of a multimedia value chain, 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** to play a role in multimedia **value chain**s

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: **content**, **contract**, **device**, **event**, **license**, **service**, and **user**

3.1.8
governance
ability to control, direct or oversee the behavior of each **entity** or operation in an multimedia **value chain**

3.1.9
license
collection of authorizations, conditions and payment terms granted by a **user** to other **user**s

3.1.10
operation
process in which an **Entity** is altered or manipulated

3.1.11
protocol
set of rules and data format used by two **Device**s to communicate

3.1.12
rate
function of expressing the perceived ranking of an **entity** related to a metric

3.1.13
remunerate
function of assigning money to a **user** in exchange of **right**s

3.1.14
reputation
measure of the credibility of or the possibility (e.g., legal) for a **user** to be a party in a transaction

3.1.15
resource
individually identifiable asset or a sequence of assets such as a 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.

3.1.16
right
ability of a **user** to perform an Operation in the multimedia **value chain**

3.1.17
role
ability of a **user** to perform a set of Operations in the multimedia **value chain**

3.1.18
service
operation performed on an **entity** by a **user** on behalf of other **user**s

3.1.19
service definition
specification of a **Service** in terms of interfaces, protocols as well as syntax and semantics of the protocol data formats

3.1.20
service instance
particular implementation of a **Service**

3.1.21
service instance declaration
description of a **Service Instance** in terms of provider, connection end-points, and usage conditions

3.1.22
service provider
user offering **service**s to other **user**s

3.1.23
space shifting
function of consuming content on a **device** (Device Shifting), or at a place (Space Shifting) or at a time (Time Shifting) different than the one intended by the rights holder

3.1.24
tag
free text descriptive information attached to an **entity**

3.1.25
user
any participant in multimedia **value chains**

3.1.26
value chain
collection of **user**s, including Creators, End Users and Service Providers, that conform to this standard

3.2 Abbreviated terms

For the purposes of this document, the abbreviated terms given in the following apply:

AIK Attestation Identity Key
AIT Advanced IPTV Terminal
AS Aggregated Service
ATIS Alliance for Telecommunications Industry Solutions
BBL Bitstream Binding Language
BPMN Business Process Model and Notation
CEL Contract Expression Language
CRID Content Reference Identifier
CS Classification Scheme
DB Database
DI Digital Item
DIA Digital Item Adaptation
DID Digital Item Declaration
DIDL Digital Item Declaration Language
DVB Digital Video Broadcasting
DII Digital Item Identification
EPG Electronic Program Guide
ER Event Report
ERR Event Report Request
ES Elementary Service
ETSI European Telecommunications Standards Institute
FRU Fragment Request Unit
FUU Fragment Update Unit
HTTP Hypertext Transfer Protocol
IIF IPTV Interoperability Forum
IPMP Intellectual Property Management and Protection
IPTV Internet Protocol Television
LASeR Lightweight Application Scene Representation
MDS Multimedia Description Schemes
MIME Multipurpose Internet Mail Extensions
MPEG Moving Picture Experts Group
MPEG-4 Coding of audio-visual objects (see ISO/IEC 14496)
MPEG-7 Multimedia Content Description Interface Standard (see ISO/IEC 15938)
MPEG-21 Multimedia Framework (see ISO/IEC 21000)
MPEG-A Multimedia application format (see ISO/IEC 23000)
MPEG-M Multimedia Service Platform Technologies (see ISO/IEC 23006)
MPEG-V Media Context and Control (see ISO/IEC 23005)

MPQF	MPEG Query Format
MSPT	Multimedia Service Platform Technologies
NER	Named Entity Recognition
OWL	Web Ontology Language
PCR	Platform Configuration Registers
POS	Part of Speech
RDF	Resource Description Framework
REL	Rights Expression Language
RTP	Real-Time Transport Protocol
SAML	Security Assertion Markup Language
SDP	Session Description Protocol
SID	Service Instance Declaration
SLA	Service Level Agreement
SNR	Signal-to-Noise Ratio
SP	Service Provider
SPARQL	SPARQL Protocol and RDF Query Language
SVC	Scalable Video Coding
TCPA	Trusted Computing Platform Alliance
TESP	Transact Entity Service Provider
TPM	Trusted Platform Module
TSS	Trusted Software Stack
UED	Usage Environment Description
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VAT	Value Added Tax
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

4 Namespaces and conventions

4.1 Namespaces

4.1.1 Introduction

URN namespaces defined in this Part of ISO/IEC 23006 conform to RFC 3614. The "standard name" used as prefix of the namespace specific string of URN namespaces defined in this Part of ISO/IEC 23006 is "`mpegM`".

NOTE Although RFC 3614 specifies the "standard name" to be case-insensitive, some tools might be incapable of handling this case insensitivity properly. Thus, alternative capitalizations are discouraged.

4.1.2 Schema namespaces

The value of the namespace URI for W3C XMLSCHEMA Schema definitions of base type used commonly by all Elementary Services in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:01-base-NS:2012
```

The value of the namespace URI for XML Schema definitions of Elementary Service protocol messages in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:02-service-NS:2012
```

The value of the namespace URI for XML Schema definitions of the BPMN 2.0 extension for message flow references in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:04-bpmn-ext-mfr-NS:2012
```

The value of the namespace URI for XML Schema definitions of Service Instance Declarations in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:05-sid-NS:2012
```

The value of the namespace URI for XML Schema definitions of the MPEG-21 DIDL extension in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:06-didl-NS:2012
```

The value of the namespace URI for XML Schema definitions of the MPEG-M IPMP Info Profile in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:07-IPMPINFO-NS:2012
```

The value of the namespace URI for XML Schema definitions of the MPEG-M IPMP DIDL Profile in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:schema:08-IPMPDIDL-NS:2012
```

4.1.3 BPMN namespaces

The value of the namespace URI for BPMN 2.0 XML representations in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:bpmn:01-service-NS:2012
```

4.1.4 Service Type namespaces

The value of the namespace URI for XML Schema definitions of the Service Type "Recognize Speech" in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:service-type:01-recognize-speech-NS:2012
```

The value of the namespace URI for XML Schema definitions of the Service Type "Synthesize Speech" in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:service-type:02-synthesize-speech-NS:2012
```

The value of the namespace URI for XML Schema definitions of the Service Type "Extract Sensory Information" in this Part of ISO/IEC 23006 is:

urn:mpeg:mpegM:service-type:03-extract-sensory-information-NS:2012

The value of the namespace URI for XML Schema definitions of the Service Type "Content Adaptation" in this Part of ISO/IEC 23006 is:

urn:mpeg:mpegM:service-type:04-content-adaptation-NS:2012

The value of the namespace URI for XML Schema definitions of the Service Type "Process Language" in this Part of ISO/IEC 23006 is:

urn:mpeg:mpegM:service-type:05-process-language-NS:2012

The value of the namespace URI for XML Schema definitions of the Service Type "Translate Language" in this Part of ISO/IEC 23006 is:

urn:mpeg:mpegM:service-type:05-translate-language-NS:2012

The value of the namespace URI for XML Schema definitions of the Service Type "Resource Transcoding" in this Part of ISO/IEC 23006 is:

urn:mpeg:mpegM:service-type:05-resource-transcoding-NS:2012

The value of the namespace URI for XML Schema definitions of the Service Type "Stream Repurposing" in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:service-type:06-stream-repurposing-NS:2012
```

4.1.5 Classification Scheme namespaces

The value of the namespace URI for Classification Schemes for IPTV offering discovery sections in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:cs:02-iptvods-NS:2012
```

The value of the namespace URI for Classification Schemes for various MPEG-M Elementary Services in this Part of ISO/IEC 23006 is:

```
urn:mpeg:mpegM:cs:03-es-NS:2012
```

4.2 Namespace convention

For clarity, throughout this Part of ISO/IEC 23006, 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 specification are not normative, e.g., "`mpegm:`", "`dia:`".

In particular, most of the informative examples in this specification 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
xsd	http://www.w3.org/2001/XMLSchema	http://www.w3.org/2001/XMLSchema.xsd
xsi	http://www.w3.org/2001/XMLSchema-instance	
bbl	urn:mpeg:mpeg21:2007:01-BBL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dis/bbl.xsd
bpmn	http://www.omg.org/spec/BPMN/20100524/MODEL	http://www.omg.org/spec/BPMN/20100501/BPMN20.xsd

Prefix	Corresponding namespace	Schema location
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:cel:core:2012	
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	
dia	urn:mpeg:mpeg21:2003:01-DIA-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dia-2nd/UED-2nd.xsd
didl	urn:mpeg:mpeg21:2002:02-DIDL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didl.xsd
didl-msx	urn:mpeg:maf:schema:mediastreaming:DIDLextensions	Defined in ISO/IEC 23000-5:2011
didmodel	urn:mpeg:mpeg21:2002:02-DIDMODEL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didmodel.xsd
dii	urn:mpeg:mpeg21:2002:01-DII-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/dii/dii.xsd
dsig	http://www.w3.org/2000/09/xmldsig#	http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd
ebucore	urn:ebu:metadata-schema:ebuCore_2010	http://www.ebu.ch/metadata/schemas/EBUCore/20100820/EBU_CORE_20100820.xsd
erl	urn:mpeg:mpeg21:2005:01-ERL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/er/er.xsd

Prefix	Corresponding namespace	Schema location
esi	urn:mpeg:mpegM:service-type:03-extract-sensory-information-NS:2012	
etsi	urn:dvb:metadata:iptv:sdns:2008-1	Defined in ETSI TS 102 034 [9]
fru	urn:mpeg:mpegB:schema:FragmentRequestUnits:2007	Defined in ISO/IEC 23001-2:2008
ipmpdidl	urn:mpeg:mpeg21:2004:01-IPMPDIDL-NS	http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/ipmp/ipmpdidl.xsd
ipmpinfo	urn:mpeg:mpeg21:2004:01-IPMPINFO-NS	standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/ipmp/ipmpgeneral.xsd
ipmpinfo-msx	urn:mpeg:maf:Schema:mediastreaming:IPMPINFOextensions:2007	Defined in ISO/IEC 23000-5:2011
ipmpmsg	urn:mpeg:mpegB:schema:IPMP-XML-MESSAGES:2007	Defined in ISO/IEC 23001-3:2008
mpeg4ipmp	urn:mpeg:mpeg4:IPMPSchema:2002	Defined in ISO/IEC 14496-13:2004
mpeg7	urn:mpeg:mpeg7:schema:2004	
mpqf	urn:mpeg:mpqf:schema:2008	Defined in ISO/IEC 15938-12
mpegm	urn:mpeg:mpegM:schema:02-service-NS:2012	
mpegmb	urn:mpeg:mpegM:schema:01-base-NS:2012	
mpegm-didl	urn:mpeg:mpegM:schema:06-didl-NS:2012	
mpegm-ipmpdidl	urn:mpeg:mpegM:schema:08-IPMPDIDL-NS:2012	
mpegm-ipmpinfo	urn:mpeg:mpegM:schema:07-IPMPINFO-NS:2012	
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
rs	urn:mpeg:mpegM:service-type:01-recognize-speech-NS:2012	

Prefix	Corresponding namespace	Schema location
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 [7]
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

NOTE In this Part of ISO/IEC 23006, it is discouraged to link to third-party XML Schema definitions in the `schemaLocation` attribute of `import` statements. Instead, local copies of these XML Schema definitions should be used. This measure increases availability of third-party XML Schema definitions and avoids excessive traffic [11].

Unlike the informative descriptions examples, the normative specification of the syntax of tools in XML Schema follows the namespace binding context defined in the relevant schema declaration.

4.3 Conventions

Fixed-width font is used to indicate literal machine-readable character sequences.

The names of XML Schema attributes appear in fixed-width font in mixed case with an initial lower case letter. The names of XML Schema elements and types appear in fixed-width font in mixed case with an initial upper case letter.

XML Schema definitions are represented with orange amber background color and a double line black border.

XML examples are represented with gray background color and a single line black border.

XML Infosets of MPEG-7 Classification Schemes, XML declarations of Service Types, and BPMN 2.0 XML notations are represented with gray background color and a double line black border.

The protocol specifications for Elementary Services in this Part of ISO/IEC 23006 are represented by protocol tables. The first row of a *protocol table* is the header, which specifies the entities involved in this protocol. Each subsequent row describes a single step of the protocol run. For each step, the first column indicates the

step number, subsequent columns describe the action taken by the corresponding entity. If a step is not attributed to a single entity, then cells in that row are merged accordingly. Steps that are only executed depending on some conditions are marked with the keyword "(optional)" and are represented in light gray background color. The example below illustrates of such a protocol table.

Steps	Entity A	Entity B
1.	Entity A and Entity B perform some common action.	
2.	Entity A sends a message to Entity B.	
3.		Entity B processes the received message.
4.		(optional) Depending on the result of the processing, Entity B may send a response to Entity A.

5 Elementary Services

5.1 Introduction

5.1.1 Overview of Elementary Services

This subclause introduces the concept of Elementary Services, defines relevant terms, and provides guidelines and rules for the specification of Elementary Services. 0 gives an overview of Operations and Entities, as well as the corresponding Elementary Services. 5.1.3 to 5.1.8 provide guidelines and rules for the specification of Elementary Services and the specification of Service Instance Declarations.

An Elementary Service (e.g., *Search Content*) corresponds to an Operation (e.g., *Search*) that is performed for a kind of Entity (e.g., *Content*).

An Elementary Service (ES) is a Service of atomic nature, which cannot be usefully divided into smaller parts and does not normatively mandate other Services to be executed during its workflow.

The terms defined in Table 2 allow for generalizing statements about Elementary Services.

Table 2 — Definition of kinds of Elementary Services

Term	Definition	Example (informative)
abstract Elementary Service	Abstraction of all Elementary Services that correspond to a specific Operation, regardless of their kind of Entity. NOTE An *abstract Elementary Service* specifies the protocol and workflow that all regular Elementary Services derived from it share (unless specified otherwise).	*Search Entity*

Term	Definition	Example (informative)
regular Elementary Service	Corresponds to a specific Operation that is performed for a specific kind of Entity. The terms *regular Elementary Service* and *Elementary Service* are used synonymously. Elementary Services can be further classified by their complexity: — An Elementary Service with a simple request-response protocol is called *atomic Elementary Service*. — An Elementary Service with a protocol that uses more than these two messages is called *compound Elementary Service*. NOTE A *regular Elementary Service* may be derived from an abstract Elementary Service. NOTE A *regular Elementary Service* specifies the protocol, workflow, as well as the syntax and semantics of protocol data formats.	*Deliver Content* (a compound regular Elementary Service not derived from an abstract Elementary Service). *Search Content* (an atomic regular Elementary Service derived from *Search Entity*). *Authenticate Content* (an atomic regular Elementary Service specified according to but not derived by, *Authenticate Entity* abstract Elementary Service)
generic Elementary Service	Elementary Service that requires a specific Service Type in order to be instantiated. NOTE A *generic Elementary Service* corresponds to a specific kind Entity but leaves the Operation generic.	*Process Content* (the only generic Elementary Service specified in this Part of ISO/IEC 23006).
specific Elementary Service	*Generic Elementary Service* that is associated with a Service Type. NOTE The Service Type further specifies the Operation that corresponds to the *specific Elementary Service*.	*Process Content* with Service Type "Recognize Speech".

5.1.2 Overview of Operations and Entities

The Entities defined in this Part of ISO/IEC 23006 are listed in Table 3.

Table 3 — Definition of Entities

Entity	Definition	Representation
Content	An ISO/IEC 21000 Digital Item and its component elements, namely Resources (e.g., media, scripts, executable), Identifiers, Descriptions (e.g., metadata) and Event Reports.	Specified in ISO/IEC 21000-2. An alternative representation is specified in Annex F.
Contract	A set of metadata, Licenses, promises and signers agreed by Users of an MPEG-M value chain, 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.	Specified in ISO/IEC 21000-20.
Device	A hardware/software or simply software apparatus that enables a User to play a role in MPEG-M value chains.	Specified in ISO/IEC 23000-5.
Event	The performance of a specified set of functions or Operations.	Specified in ISO/IEC 21000-15.
License	A collection of authorizations, conditions and payment terms granted by a User to other Users.	Specified in ISO/IEC 21000-5.
Service	A set of Operations performed by a User on behalf of other Users. A Service Instance is a particular implementation of a Service, typically described in terms of provider, connection end-points, and usage conditions.	The representation of Service Instances is specified in Annex C.
User	Any participant in MPEG-M value chains.	Specified in ISO/IEC 15938-5.

The Operations defined in this Part of ISO/IEC 23006 are listed in Table 4.

Table 4 — Definition of Operations

Operation	Definition
Authenticate	To confirm the identity of an Entity.
Authorize	To grant rights to perform certain operations.
Check With	To check if the User is allowed to act on the content based on a Contract or a Licens.
Create	To generate a data structure representing a Content, a Contract, or a License.
Deliver	To ask a Service Provider to transfer a Content or Contract from a Device or User to another Device or User.
Describe	To add descriptive information to an Entity.
Identify	To assign an identifier to an Entity.

Operation	Definition
Negotiate	To assist Users to seek agreement on a Transaction.
Package	To make a Content Item suitable for Delivery.
Post	To make a Content discoverable by other Entities.
Present	To make Contract or License available for consumption by the intended user.
Process	To make change to an Entity or combine Entities into another Entity.
Request	To call for the provision of an Entity.
Revoke	To discontinue the validity of an Entity.
Search	To find the URI or physical location of Entities satisfying given conditions.
Store	To transfer a digital Entity to a Service Provider for storage.
Transact	To grants Rights to an Entity in exchange of a remuneration.
Verify	To check the validity of an Entity.

Table 5 shows the Elementary Services for the combinations of Operations and Entities are defined in this Part of ISO/IEC 23006. The entries in the table refer to the subclause in which the respective Elementary Service is defined. Note that the column labeled "Entity" refers to abstract Elementary Services, from which the regular Elementary Services in that row are derived.

Table 5 — Elementary Services classified by Operations and Entities

	Entity	**Content**	**Contract**	**Device**	**Event**	**License**	**Service**	**User**
Authenticate	*5.5.1*	5.5.2	5.5.3			5.5.4		5.5.5
Authorize								5.6.1
Check With			5.7.1			5.7.2		
Create		5.8.1	5.8.2			5.8.3		
Deliver		5.9.1	5.9.2					
Describe	*5.10.1*	5.10.2		5.10.3			5.10.4	5.10.5
Identify	*5.11.1*	5.11.2	5.11.3	5.11.4		5.11.5		0
Negotiate	*5.12.1*		5.12.2			5.12.3		
Package		5.13.1						
Post		5.14.1						
Present	*5.15.1*		5.15.2			5.15.3		

	Entity	Content	Contract	Device	Event	License	Service	User
Process		5.16.1				5.16.2		
Request		5.17.1	5.17.2	5.17.3	5.17.4	5.17.5		
Revoke	*5.18.1*	5.18.2	5.18.3			5.18.4		
Search	*5.19.1*	5.19.2	5.19.3	5.19.4		5.19.5	5.19.6	5.19.7
Store		5.20.1	5.20.2		5.20.3	5.20.4		
Transact	*5.21.1*	5.21.2				5.21.3		
Verify			5.22.1	5.22.2		5.22.3		

5.1.3 General Service Definition

The following subclauses describe the relation between the *General Service Definition*, *Service Type*, and *Service Instance Declaration*.

A *General Service Definition* denotes the specification of interfaces, protocol specification as well as syntax and semantics of the protocol data formats of a Service. A General Service Definition can either specify an Elementary Service or an Aggregated Services. Examples of General Service Definitions are the Elementary Services specified in 5.5 to 5.22 of this part of ISO/IEC 23006 (called MPEG-M for short) and the Aggregated Services provided in 5.3 of ISO/IEC 23006-5. The terms *General Service Definition* and *Service Definition* are used synonymously throughout this standard.

A General Service Definition must comprise at least all of the following parts:

a) Narrative description of the actions that the Service performs;

b) BPMN 2.0 XML representation of a Collaboration and a Process for the Service Provider as a formal description of Service's workflow;

c) Syntax of input and output messages, represented as XML Schema;

d) Descriptions of semantics of input and output messages and of their parameters;

e) Extension to the Service Instance Declaration, containing syntax and semantics of any additional parameters that should be published by a Service Provider.

The above defined parts are mandatory for the description of both Elementary Services and Aggregated Services.

This standard does not specify the representational format for the narrative description of actions, description of semantics of messages, and description of semantics of extensions to the SID. It is, however, recommended to follow the style of the specification of Elementary Services provided in this Part of ISO/IEC 23006.

The BPMN 2.0 Process and Collaboration instances must at least model the exchange of messages between the caller and the SP as defined in 5.1.8. For the description of Elementary Services, the BPMN Process should be of type BPMN Public Process. The BPMN messages should be mapped to XML data types of input and out messages.

NOTE The BPMN 2.0 Process and Collaboration instances of all Elementary Services specified in this part of ISO/IEC 23006 are collectively provided in Annex H. Since the protocols for most of these Elementary Services only consist of a request message and a corresponding response message, the BPMN representations are only included directly in the Service definitions where it is considered appropriate.

For the definition of Aggregated Services, it is not mandatory to describe the interaction with the Services of which the AS is composed. Optionally, this information can be provided in the definition, e.g., in the BPMN Process.

The BPMN Process instance must be represented as XML serialization. Optionally, an informative BPMN diagram of the BPMN Process instance can be included in the description of a Service. If a diagram is present, it must conform to the XML serialization of the BPMN Process.

For regular Services, the syntax definition of input and output messages comprises the definition of XML elements and any XML complex types necessary for instantiating these elements.

For abstract Services, the syntax definition of input and output messages shall only comprise the XML complex types for the messages, but not the message XML elements themselves. For regular Services based on abstract Services, the syntax definition of input and output messages shall comprise the message XML elements based on the XML complex types of the corresponding abstract Service.

EXAMPLE The Search Entity abstract Elementary Service defines the complex types `mpegm:SearchEntityRequestType` and `mpegm:SearchEntityResponseType`. The Search Content regular Elementary Service, which is based on the abstract ES, defines a message element `mpegm:SearchContentRequest` of type `mpegm:SearchEntityRequestType` and a message element `mpegm:SearchContentResponse` of type `mpegm:SearchEntityResponseType`.

The extension to the SID should be represented as an XML complex type that is derived from the `sid:ServiceInstanceDeclarationType` complex type.

Optionally, the description of a Service can contain examples about the usage of the Service.

The description of Services may contain extensions that are not specified in this International Standard.

5.1.4 Service Type

The Process Content elementary service defined in 5.16.1 provides a generic workflow for the processing of content. In order to specify the concrete task of this processing (e.g., Recognize Speech, Transcoding, etc.), 5.16.1 defines the concept of Service Types. Process Content is called a generic Elementary Service since it needs a Service Type in order to be implemented.

The Service Type of a Service Instance of Process Content must be provided in the Service Instance Declaration.

A non-exhaustive list of Service Types for Process Content is specified in Annex B.

Further examples of Service Types are provided in Annex A of ISO/IEC 23006-5. Note that ISO/IEC 23006-5 also specifies a mechanism for the representation of Service Types at the Service aggregation level.

Other parties may specify additional Service Types. This is done by making the `ServiceType` element, representing the formal specification of the Service Type, available. If the `ServiceType` element contains a `RequestParametersType` element, this specified extension to the `RequestParametersBaseType` must also be made available. If the `ServiceType` element contains a `CompletionParametersType` element, this specified extension to the `CompletionParametersBaseType` must also be made available. The XML Schema definition of these extensions shall contain a description of the purpose of the Service Type and the specification of the semantics of the extension represented as `xsd:annotation/xsd:documentation` elements.

5.1.5 Service Instance Declaration

A *Service Instance Declaration* describes a concrete realization of a Service where the Service is specified through a General Service Definition. A Service Instance Declaration is created by a Service Provider and specifies the concrete parameters that enable a client to invoke that Service Instance. A Service Instance Declaration must identify the General Service Definition (or Service Type) that it implements. Conversely, a General Service Definition (or a Service Type) can have multiple Service Instance Declarations implementing it.

A Service Instance Declaration comprises the following components:

a) Identifier of the Service Instance

b) Name of the Service Instance

c) Identifier of the corresponding General Service Definition

d) Indication of supported formats for metadata representation

e) Provider (i.e., address, contact information, self-description, etc.)

f) Connection end-point (i.e., where the client has to connect to in order to use the Service), e.g., IP address

g) Protocol binding description that specifies how MPEG-M protocol messages are exchanged

h) License and/or Contract that regulates the use of the Service Instance (optional)

i) Description of the Service Instance (optional)

j) Service availability period (optional)

k) Application-specific data, e.g., further configuration (optional)

l) Signature (optional).

If the signature is present, it always signs the entire Service Instance Declaration, including any extensions.

The preferred XML representation of Service Instance Declarations is specified in Annex C.

5.1.6 Message transport

This document does not specify the transport protocol over which the payload is delivered. The transport protocol must provide ordered message delivery. The transport protocol of a Service Instance is indicated by the SID as described in 5.1.5 and Annex C.

The communication for MPEG-M Services relies on mutual authentication between the involved entities as described in 5.2.5.1.

Response messages may be represented by HTTP responses in some cases as specified in Annex D.

5.1.7 Validation

Validating a document against the MPEG-M schema (as specified in W3C XMLSCHEMA) is necessary, but not sufficient, to determine its validity with respect to MPEG-M. After a document is validated against the MPEG-M schema, it shall also be subjected to additional validation rules. These additional rules are given below in the descriptions of the elements to which they pertain.

5.1.8 BPMN 2.0 XML representation of Service workflows

5.1.8.1 Introduction

OMG BPMN 2.0 specifies a notation to represent processes which is used in throughout this document to provide a formal representation Service workflows. BPMN 2.0 specifies both, a graphical notation and an XML representation for processes. The workflow of an Elementary Service is normatively specified through the XML representation of BPMN 2.0 Collaboration and Process instances.

Unless specified otherwise (see 5.1.8.2), the BPMN 2.0 XML representation of an Elementary Service must contain all of the following elements:

a) A `bpmn:collaboration` that identifies the participants (i.e., client and SP) and the message flows between them.

b) A `bpmn:process` representing a BPMN Public Process for the SP's workflow.

c) `bpmn:message` elements for all messages exchanged between client and SP. Each `bpmn:message` references a `bpmn:itemDefinition`.

d) `bpmn:itemDefinition` elements that reference the actual XML messages of the protocol data format.

The following rules apply to the `bpmn:collaboration` element:

The `bpmn:collaboration` shall specify the `id` attribute in order to allow references from the `bpmn:process` of the SP. The `name` attribute shall be the name of the Elementary Service (e.g., "Deliver Content").

The `bpmn:participant` elements shall specify the name attributes (e.g., "Client" for the client and "Deliver Content SP" for the SP). The `bpmn:participant` representing the SP must reference the `bpmn:process` of the Elementary Service through the `processRef` attribute. The `bpmn:participant` representing the client shall specify the `id` attribute in order to enable Service aggregation as explained in ISO/IEC 23006-5.

The `bpmn:messageFlow` elements must reference through their `sourceRef` and `targetRef` attributes the `bpmn:process` for the client and the BPMN Flow Nodes (i.e., Events, Tasks, Gateways) within the `bpmn:process` for the SP. Each The `bpmn:messageFlow` element shall specify the `id` attribute in order to allow Aggregated Services to match message flows of the Services they are composed of (see 6.2 of ISO/IEC 23006-5). Unless specified otherwise (see 5.1.8.2), the `bpmn:messageFlow` elements must reference the corresponding `bpmn:messages` through the `messageRef` attribute.

The following rules apply to the `bpmn:process` element:

The `bpmn:process` must specify the `id` attribute in order to allow references from the `bpmn:participant` of the `bpmn:collaboration` and to enable Service aggregation as explained in ISO/IEC 23006-5. The `processType` attribute shall be set to "Public". The `bpmn:collaboration` of the Elementary Service shall be referenced through the `definitionalCollaborationRef` attribute.

The `bpmn:process` shall contain a `bpmn:startEvent`, which is triggered by the first message flow from the client to the SP. The `bpmn:startEvent` shall contain a `bpmn:messageEventDefinition` child element. Further BPMN Flow Nodes (i.e., Events, Tasks, Gateways, SubProcesses), and the `bpmn:sequenceFlow` elements connecting them, specify a high-level workflow for the SP. The Flow Nodes of the `bpmn:process` shall not reference any BPMN messages themselves; the connection is established by the `bpmn:messageFlow` elements of the `bpmn:collaboration`. Along the normal workflow, outgoing response messages should indicate successful execution of the respective tasks (i.e., they should contain an element derived from `ProtocolSuccessType`). Exception sequence flows should originate from BPMN Error Boundary Events and the response messages originating from BPMN Send Tasks along those sequence flows shall indicate failures (i.e., they should contain an element derived from `ProtocolFailureType`).

The following rules apply to the bpmn:message elements:

The bpmn:message must specify the id attribute in order to allow references from the bpmn:messageFlow. The name attribute shall be the name of the protocol message of the Elementary Service (e.g., "Deliver Content Request"). The corresponding bpmn:itemDefinition must be referenced through the itemRef attribute.

The following rules apply to the bpmn:itemDefinition elements:

The bpmn:itemDefinition must specify the id attribute in order to allow references from the bpmn:message elements. The XML element of the actual protocol message of the Elementary Service (e.g., mpegm:DeliverContentRequest) must be referenced through the structureRef attribute.

The representation of workflows of the Elementary Services defined in this document is provided in Annex H.

5.1.8.2 BPMN 2.0 XML representation of abstract base Elementary Service workflows

The binding of the workflow of abstract base Elementary Services to the derived actual Elementary Service is accomplished through the extensibility mechanism of BPMN 2.0.

In BPMN 2.0, a bpmn:messageFlow element represents the connection between two BPMN 2.0 Interaction Nodes (i.e., Pool/Participant, Activity, or Event), along which a message is sent. BPMN 2.0 envisions that the bpmn:messageFlow element specifies the reference to a bpmn:message element which links to the actual data type of the message. This mechanism is not sufficient for defining a generic workflow (e.g., for an abstract base Elementary Service) and concretizing it with data formats of messages (e.g., to obtain an actual Elementary Service).

The BPMN 2.0 extension for message flow references provides the bpmn:message element with an additional bpmnext1:MessageFlowRef child element in order to bind it to a bpmn:messageFlow element.

The XML Schema definition of the BPMN 2.0 extension for message flow references is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:mpeg:mpegM:schema:04-bpmn-ext-mfr-NS:2012"
xmlns:bpmnext1="urn:mpeg:mpegM:schema:04-bpmn-ext-mfr-NS:2012"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="MessageFlowRef" type="QName"/>
</schema>
```

Semantics of the bpmnext1:MessageFlowRef:

Name	*Definition*
bpmnext1:MessageFlowRef	Provides an association between bpmn:messageFlow and bpmn:message, similar to the messageRef attribute of the bpmn:messageFlow, but specifying it from the other end at the bpmn:message element. The bpmnext1:MessageFlowRef element shall only be used as an extension to bpmn:message elements. The value is a reference to a bpmn:messageFlow element.

If the Elementary Service has an abstract base Elementary Service (e.g., Search Entity is the abstract base Elementary Service for Search Content), then the following additional rules apply:

The surrounding `bpmn:definitions` root element must specify a `bpmn:extension` element. The `mustUnderstand` attribute of the `bpmn:extension` element must be set to "true". The `definition` attribute must reference the `bpmnext1:MessageFlowRef` element.

The BPMN 2.0 XML representation of the abstract base Elementary Service (e.g., Search Entity) must contain the `bpmn:collaboration` and `bpmn:process` elements as specified in 5.1.8. However, the `bpmn:messageFlow` elements must not reference any `bpmn:message` elements. Each `bpmn:messageFlow` element must specify the `id` attribute in order to be referenced by a `bpmnext1:MessageFlowRef` element in the BPMN representation of the actual Elementary Service (e.g., Search Content).

The BPMN 2.0 XML representation of the abstract base Elementary Service (e.g., Search Entity) must not contain any `bpmn:message` or `bpmn:itemDefinition` elements.

The BPMN 2.0 XML representation of the actual Elementary Service derived from the abstract base Elementary Service (e.g., Search Content) must contain the `bpmn:message` elements and `bpmn:itemDefinition` elements as specified in 5.1.8. Each `bpmn:message` element must contain a `bpmn:extensionElements` element which contains one or more `bpmnext1:MessageFlowRef` elements. Each `bpmnext1:MessageFlowRef` element must reference an individual corresponding `bpmn:messageFlow` element (e.g., the message flow for Search Entity Request) from the `bpmn:collaboration` of the corresponding abstract base Elementary Service.

The BPMN 2.0 XML representation of the actual Elementary Service (e.g., Search Content) must not contain any `bpmn:collaboration` or `bpmn:process` elements.

Examples for the BPMN 2.0 XML representation of abstract base Elementary Services are Describe Entity and Search Entity as provided in Annex H.

5.1.8.3 Examples

The example below shows the BPMN 2.0 XML representation for the Elementary Service Create Content (see 5.8.1).

```
<bpmn:definitions targetNamespace="urn:mpeg:mpegM:bpmn:01-service-NS:2012">
  <!-- ********************************************************** -->
  <!--  Collaboration, Process, and Messages for Create Content   -->
  <!-- ********************************************************** -->
  <!-- NOTE: This is an atomic workflow. -->
  <bpmn:collaboration id="create_content_collaboration" name="Create Content">
    <bpmn:participant id="create_content_client" name="Client"/>
    <bpmn:participant name="Create Content SP" processRef="create_content_sp"/>
    <bpmn:messageFlow id="create_content_request_message_flow" name="Create
Content Request MessageFlow" sourceRef="create_content_client"
targetRef="create_content_start" messageRef="create_content_request"/>
    <bpmn:messageFlow id="create_content_reject_message_flow" name="Create
Content Reject MessageFlow" sourceRef="create_content_reject_request"
targetRef="create_content_client" messageRef="create_content_response"/>
    <bpmn:messageFlow id="create_content_response_message_flow" name="Create
Content Response MessageFlow" sourceRef="create_content_respond"
targetRef="create_content_client" messageRef="create_content_response"/>
  </bpmn:collaboration>
  <bpmn:process id="create_content_sp" processType="Public"
definitionalCollaborationRef="create_content_collaboration">
    <bpmn:startEvent id="create_content_start">
      <bpmn:messageEventDefinition/>
    </bpmn:startEvent>
```

```
    <bpmn:sequenceFlow sourceRef="create_content_start"
targetRef="create_content_fulfill_request"/>
    <bpmn:task id="create_content_fulfill_request" name="Fulfill Request"/>
    <bpmn:boundaryEvent id="create_content_fulfill_error_occurrence"
attachedToRef="create_content_fulfill_request">
      <bpmn:eventDefinition xsi:type="bpmn:tErrorEventDefinition"/>
    </bpmn:boundaryEvent>

    <bpmn:sequenceFlow sourceRef="create_content_fulfill_error_occurrence"
targetRef="create_content_reject_request"/>
    <bpmn:sendTask id="create_content_reject_request" name="Reject Request"/>
    <bpmn:sequenceFlow sourceRef="create_content_reject_request"
targetRef="create_content_error_end"/>
    <bpmn:endEvent id="create_content_error_end" name="Error">
      <bpmn:eventDefinition xsi:type="bpmn:tErrorEventDefinition"/>
    </bpmn:endEvent>

    <bpmn:sequenceFlow sourceRef="create_content_fulfill_request"
targetRef="create_content_respond"/>
    <bpmn:sendTask id="create_content_respond" name="Respond"/>
    <bpmn:sequenceFlow sourceRef="create_content_respond"
targetRef="create_content_success_end"/>
    <bpmn:endEvent id="create_content_success_end" name="Success"/>
  </bpmn:process>

  <bpmn:message id="create_content_request" name="CreateContentRequest"
itemRef="create_content_request_itemdef"/>
  <bpmn:message id="create_content_response" name="CreateContentResponse"
itemRef="create_content_response_itemdef"/>

  <bpmn:itemDefinition id="create_content_request_itemdef"
structureRef="mpegm:CreateContentRequest"/><!-- Binding of the bpmn:message to
the actual MPEG-M request message definition. -->
  <bpmn:itemDefinition id="create_content_response_itemdef"
structureRef="mpegm:CreateContentResponse"/><!-- Binding of the bpmn:message to
the actual MPEG-M response message definition. -->
</bpmn:definitions>
```

The example below shows the BPMN 2.0 XML representations of the abstract base Elementary Service Search Entity (see 5.19.1) and of the actual Elementary Service Search Content (see 5.19.2) based on it.

```
<bpmn:definitions targetNamespace="urn:mpeg:mpegM:bpmn:01-service-NS:2012">

  <bpmn:extension definition="bpmnext1:MessageFlowRef" mustUnderstand="true">
    <bpmn:documentation>Used in bpmn:message to reference a
bpmn:messageFlow.</bpmn:documentation>
  </bpmn:extension>
  <!-- ************************************************************** -->
  <!--     Collaboration and Process for Search Entity                -->
  <!-- ************************************************************** -->
  <!-- NOTE: This is an atomic workflow for an abstract base Elementary Service.
-->
  <bpmn:collaboration id="search_entity_collaboration" name="Search Entity">
    <bpmn:participant id="search_entity_client" name="Client"/>
    <bpmn:participant name="Search Entity SP" processRef="search_entity_sp"/>
    <bpmn:messageFlow id="search_entity_request_message_flow" name="Search Entity
Request MessageFlow" sourceRef="search_entity_client"
targetRef="search_entity_start"/>
```

```
    <bpmn:messageFlow id="search_entity_reject_message_flow" name="Search Entity
Reject MessageFlow" sourceRef="search_entity_reject_request"
targetRef="search_entity_client"/>
    <bpmn:messageFlow id="search_entity_response_message_flow" name="Search
Entity Response MessageFlow" sourceRef="search_entity_fulfill_request"
targetRef="search_entity_client"/>
  </bpmn:collaboration>
  <bpmn:process id="search_entity_sp" processType="Public"
definitionalCollaborationRef="search_entity_collaboration">
    <bpmn:startEvent id="search_entity_start">
      <bpmn:messageEventDefinition/>
    </bpmn:startEvent>
    <bpmn:sequenceFlow sourceRef="search_entity_start"
targetRef="search_entity_fulfill_request"/>
    <bpmn:task id="search_entity_fulfill_request" name="Fulfill Request"/>
    <bpmn:boundaryEvent id="search_entity_fulfill_error_occurrence"
attachedToRef="search_entity_fulfill_request">
      <bpmn:eventDefinition xsi:type="bpmn:tErrorEventDefinition"/>
    </bpmn:boundaryEvent>

    <bpmn:sequenceFlow sourceRef="search_entity_fulfill_error_occurrence"
targetRef="search_entity_reject_request"/>
    <bpmn:sendTask id="search_entity_reject_request" name="Reject Request"/>
    <bpmn:sequenceFlow sourceRef="search_entity_reject_request"
targetRef="search_entity_error_end"/>
    <bpmn:endEvent id="search_entity_error_end" name="Error">
      <bpmn:eventDefinition xsi:type="bpmn:tErrorEventDefinition"/>
    </bpmn:endEvent>

    <bpmn:sequenceFlow sourceRef="search_entity_fulfill_request"
targetRef="search_entity_respond"/>
    <bpmn:sendTask id="search_entity_respond" name="Respond"/>
    <bpmn:sequenceFlow sourceRef="search_entity_respond"
targetRef="search_entity_success_end"/>
    <bpmn:endEvent id="search_entity_success_end" name="Success"/>
  </bpmn:process>

  <!-- ********************************************************** -->
  <!--        Messages and ItemDefinitions for Search Content     -->
  <!-- ********************************************************** -->
  <bpmn:message id="search_content_request" name="SearchContentRequest"
itemRef="search_content_request_itemdef">
    <bpmn:extensionElements>

<bpmnext1:MessageFlowRef>search_content_request_message_flow</bpmnext1:MessageFlo
wRef>
    </bpmn:extensionElements>
  </bpmn:message>
  <bpmn:message id="search_content_response" name="SearchContentResponse"
itemRef="search_content_response_itemdef">
    <bpmn:extensionElements>

<bpmnext1:MessageFlowRef>search_content_reject_message_flow</bpmnext1:MessageFlow
Ref>

<bpmnext1:MessageFlowRef>search_content_response_message_flow</bpmnext1:MessageFl
owRef>
    </bpmn:extensionElements>
  </bpmn:message>
```

```
  <bpmn:itemDefinition id="search_content_request_itemdef"
structureRef="mpegm:SearchContentRequest"/><!-- Binding of the bpmn:message to
the actual MPEG-M request message definition. -->
  <bpmn:itemDefinition id="search_content_response_itemdef"
structureRef="mpegm:SearchContentResponse"/><!-- Binding of the bpmn:message to
the actual MPEG-M response message definition. -->
</bpmn:definitions>
```

5.2 Base data types, elements, and common messages

5.2.1 Introduction

This subclause specifies common base data types and elements that are used throughout various MPEG-M Services.

The common base data types and elements are defined in a separate XML Schema definition with the target namespace `urn:mpeg:mpegM:schema:01-base-NS:2012`, facilitating their reuse outside of ISO/IEC 23006.

NOTE The default namespace used in this subclause is `urn:mpeg:mpegM:schema:01-base-NS:2012`.

5.2.2 MPEG-M base Schema wrapper

The syntax of description tools specified in this subclause is provided as a collection of schema components, consisting notably in type definitions and element declarations. In order to form a valid schema document, these schema components are gathered in a same document with the following declaration defining in particular the target namespace and the namespaces prefixes.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:didl="urn:mpeg:mpeg21:2006:07-DIDL-NS"
xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xmlns:cel="urn:mpeg:mpeg21:cel:core:2012"
xmlns:rel-r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:rel-sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:ipmpinfo-msx="urn:mpeg:maf:Schema:mediastreaming:IPMPINFOextensions:2007"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
targetNamespace="urn:mpeg:mpegM:schema:01-base-NS:2012"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="ISO/IEC 23006-4 2nd edition" id="mpeg-m-base.xsd">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="w3c/xmldsig-core-schema.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2006:07-DIDL-NS"
schemaLocation="mpeg/didl.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2002:01-DII-NS"
schemaLocation="http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
21_schema_files/dii/dii.xsd"/>
  <import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="mpeg/mpeg7-
v3.xsd"/>
  <import namespace="urn:mpeg:mpeg21:cel:core:2012" schemaLocation="cel-
core.xsd"/>
```

```
  <import namespace="urn:mpeg:mpeg21:2003:01-REL-R-NS" schemaLocation="mpeg/rel-
r.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2003:01-REL-SX-NS" schemaLocation="mpeg/rel-
sx.xsd"/>
  <import namespace="urn:mpeg:maf:Schema:mediastreaming:IPMPINFOextensions:2007"
schemaLocation="mpeg/ipmpinfo-msx.xsd"/>
  <import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
schemaLocation="oasis/saml-schema-assertion-2.0.xsd"/>
```

Additionally, the following line should be appended to the resulting schema document in order to obtain a well-formed XML document.

```
</schema>
```

5.2.3 Common elements and types

5.2.3.1 Introduction

This Subclause describes elements and complex element types which are used throughout various Services.

5.2.3.2 ServiceTypeType

5.2.3.2.1 Introduction

The `ServiceTypeType` defines an XML representation for the concept of Service Types described in 5.1.4.

5.2.3.2.2 Syntax

```
  <!-- ################################################ -->
  <!--  Service Type                                    -->
  <!-- ################################################ -->
  <complexType name="ServiceTypeType">
    <sequence>
      <element ref="mpegmb:ServiceTypeIdentifier"/>
      <element name="Name" type="string"/>
      <element name="RequestParametersType" type="QName" minOccurs="0"/>
      <element name="CompletionParametersType" type="QName" minOccurs="0"/>
    </sequence>
  </complexType>
```

5.2.3.2.3 Semantics

Semantics of the `ServiceTypeType`:

Name	*Definition*
`ServiceTypeType`	Top-level type for the specification of a Service Type.
`ServiceTypeIdentifier`	Unique identifier of the Service Type.

`Name`	Descriptive name of the Service Type.
`RequestParametersType`	Specifies the complex type for the `RequestParameters` element of the `ProcessContentRequest` message.
`CompletionParametersType`	Specifies the complex type for the `CompletionParameters` element of the `ProcessContentCompletion` message.

5.2.3.2.4 Additional validation rules

5.2.3.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.2.4.2 The value of the `RequestParametersType` element must be an existing complex type that extends `RequestParametersBaseType`.

5.2.3.2.4.3 The value of the `CompletionParametersType` element must be an existing complex type that extends `CompletionParametersBaseType`.

5.2.3.3 ContractIdentifier

5.2.3.3.1 Introduction

Contracts within MPEG-M are identified by the `ContractIdentifier` element.

NOTE This Part of ISO/IEC 23006 does neither specify a particular identification scheme for Contracts nor a mechanism for attaching the `ContractIdentifier` element to a Contract.

5.2.3.3.2 Syntax

```
<!-- Definition of ContractIdentifier -->
<element name="ContractIdentifier" type="mpeg7:UniqueIDType" />
```

5.2.3.3.3 Semantics

Semantics of the `ContractIdentifier`:

Name	*Definition*
`ContractIdentifier`	Describes the identification of a Contract based on the `mpeg7:UniqueIDType`. The contract identifier corresponds to the `cel:contractId`.

5.2.3.4 LicenseIdentifier

5.2.3.4.1 Introduction

Licenses within MPEG-M are identified by the `LicenseIdentifier` element.

NOTE This Part of ISO/IEC 23006 does neither specify a particular identification scheme for Licenses nor a mechanism for attaching the `LicenseIdentifier` element to a License.

5.2.3.4.2 Syntax

```
<!-- Definition of LicenseIdentifier -->
<element name="LicenseIdentifier" type="mpeg7:UniqueIDType" />
```

5.2.3.4.3 Semantics

Semantics of the `LicenseIdentifier`:

Name	*Definition*
`LicenseIdentifier`	Describes the identification of a License. The license identifier corresponds to the `rel-r:licenseId`.

5.2.3.5 ServiceTypeIdentifier

5.2.3.5.1 Introduction

Service Types within MPEG-M are identified by the `ServiceTypeIdentifier` element.

NOTE The namespace URIs for XML Schemas of Service Types specified in 4.1.4 also serve as Service Type Identifiers of the Service Types defined in this Part of ISO/IEC 23006.

5.2.3.5.2 Syntax

```
<!-- Definition of ServiceTypeIdentifier -->
<element name="ServiceTypeIdentifier" type="mpeg7:UniqueIDType" />
```

5.2.3.5.3 Semantics

Semantics of the `ServiceTypeIdentifier`:

Name	*Definition*
`ServiceTypeIdentifier`	Describes the identification of a Service Type.

5.2.3.6 SIDIdentifier

5.2.3.6.1 Introduction

Service Instance Declarations (SIDs) within MPEG-M are identified by the `SIDIdentifier` element.

NOTE This Part of ISO/IEC 23006 does not specify a particular identification scheme for SIDs.

5.2.3.6.2 Syntax

```
<!-- Definition of SIDIdentifier -->
<element name="SIDIdentifier" type="mpeg7:UniqueIDType" />
```

5.2.3.6.3 Semantics

Semantics of the `SIDIdentifier`:

Name	*Definition*
`SIDIdentifier`	Describes the identification of a Service Instance Declaration.

5.2.3.7 UserIdentifier

5.2.3.7.1 Introduction

Users within MPEG-M are identified by the `UserIdentifier` element.

NOTE This Part of ISO/IEC 23006 does not specify a particular identification scheme for Users.

5.2.3.7.2 Syntax

```
<!-- Definition of UserIdentifier -->
<element name="UserIdentifier" type="mpeg7:UniqueIDType" />
```

5.2.3.7.3 Semantics

Semantics of the `UserIdentifier`:

Name	*Definition*
`UserIdentifier`	Describes the identification of a User.

5.2.3.8 EntityBaseType

5.2.3.8.1 Introduction

Throughout MPEG-M, Entities are in general included in messages by one of the following mechanisms:

— **Direct inclusion:** The Entity is typically included as an XML infoset (if applicable).

— **By reference:** One or more locations, represented as URLs, are specified, where the Entity can be found. If multiple URLs are given, they must all link to replications of the same Entity.

— **By identifier:** One or more identifiers specifying the same Entity. The resolution of identifiers is out of the scope of MPEG-M.

NOTE This Part of ISO/IEC 23006 does not mandate the use of particular inclusion mechanisms for Elementary Services. If a particular inclusion mechanism is recommended for an Elementary Service, this is stated in the General Service Definition. For example, for an `IdentifyContentRequest` message, direct inclusion or inclusion by reference should be preferred over inclusion by identifier. On the other hand, for a `RevokeContractRequest` message, inclusion by identifier is preferred over inclusion by reference because all replications of that contract shall be revoked and not just the one referenced instance.

The representations of Entity inclusions are derived from the `EntityBaseType`.

5.2.3.8.2 Syntax

```
  <!-- Definition of EntityBaseType -->
  <complexType name="EntityBaseType" abstract="true">
    <annotation>
      <documentation xml:lang="en">
        Derived types append a choice between direct inclusion, inclusion by
reference, and inclusion by identifier.
      </documentation>
    </annotation>
  </complexType>
```

5.2.3.8.3 Semantics

Semantics of the `EntityBaseType`:

Name	*Definition*
`EntityBaseType`	Base type for containers that include a specific kind of Entity by either direct inclusion, inclusion by reference, or inclusion by identifier.

5.2.3.9 ContentEntityType

5.2.3.9.1 Introduction

Throughout MPEG-M, Content is in general either referenced or included in the `ContentEntityType`. Content is represented as MPEG-21 DIDL. Annex F specifies a Content representation (called MPEG-M DIDL) based on MPEG-21 DIDL. The use of MPEG-M DIDL is optional.

5.2.3.9.2 Syntax

```
  <complexType name="ContentEntityType">
    <complexContent>
      <extension base="mpegmb:EntityBaseType">
        <choice>
          <element ref="didl:DIDL"/>
          <element name="ContentRef" type="anyURI" maxOccurs="unbounded"/>
          <element ref="dii:Identifier" maxOccurs="unbounded"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

5.2.3.9.3 Semantics

Semantics of the `ContentEntityType`:

Name	*Definition*
`ContentEntityType`	Container for a Digital Item, a reference to it, or an identifier of it.

`didl:DIDL`	Digital Item Declaration of the Content. The Content may also contain sensory information as specified in ISO/IEC 23005-3. NOTE Sensory Information is considered as content (not as metadata) and is therefore represented as a `didl:Resource` inside a `didl:Component` of a Digital Item. NOTE Annex F specifies a Content representation (called MPEG-M DIDL) based on MPEG-21 DIDL. The use of MPEG-M DIDL is optional.
`ContentRef`	Reference to the Content. `ContentRef` should specify a location rather than an Identifier. Multiple locations for the same Content can be specified.
`dii:Identifier`	Identifier of the Content as specified in ISO/IEC 21000-3. Multiple identifiers for the same Content can be specified.

5.2.3.9.4 Additional validation rules

5.2.3.9.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.9.4.2 If multiple `ContentRef` elements are specified, they must all point to copies of the same Content.

5.2.3.9.4.3 If multiple `dii:Identifier` elements are specified, they must all identify the same Content.

5.2.3.10 ContractEntityType

5.2.3.10.1 Introduction

Throughout MPEG-M, Contracts are in general either referenced or included in the `ContractEntityType`. Contracts are represented as ISO/IEC 21000-20 `cel:contract`.

5.2.3.10.2 Syntax

```
<complexType name="ContractEntityType">
  <complexContent>
    <extension base="mpegm:EntityBaseType">
      <choice>
        <element ref="cel:contract"/>
        <element name="ContractRef" type="anyURI" maxOccurs="unbounded"/>
        <element ref="mpegmb:ContractIdentifier" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.10.3 Semantics

Semantics of the `ContractEntityType`:

Name	*Definition*
`ContractEntityType`	Container for a Contract or a reference to it.
`cel:contract`	MPEG-21 CEL contract.
`ContractRef`	Reference to the Contract. `ContractRef` specifies a location rather than an Identifier, matching the `cel:contractId` element. Several locations for the same Contract can be specified.
`ContractIdentifier`	Identifier of the Contract. Multiple identifiers for the same Contract can be specified.

5.2.3.10.4 Additional validation rules

5.2.3.10.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.10.4.2 If multiple `ContractRef` elements are specified, they must all point to copies of the same Contract.

5.2.3.10.4.3 If multiple `ContractIdentifier` elements are specified, they must all identify the same Contract.

5.2.3.11 DeviceEntityType

5.2.3.11.1 Introduction

Throughout MPEG-M, Devices are in general either referenced or included in the `DeviceEntityType`. Devices are represented as ISO/IEC 23000-5 `ipmpinfo-msx:ToolBody`.

5.2.3.11.2 Syntax

```
<complexType name="DeviceEntityType">
  <complexContent>
    <extension base="mpegm:EntityBaseType">
      <choice>
        <element ref="ipmpinfo-msx:ToolBody"/>
        <element name="DeviceRef" type="anyURI" maxOccurs="unbounded"/>
        <element ref="ipmpinfo-msx:ToolBodyID" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.11.3 Semantics

Semantics of the `DeviceEntityType`:

Name	*Definition*
`DeviceEntityType`	Container for a Device or a reference to it.
`ipmpinfo-msx:ToolBody`	A software Device.
`DeviceRef`	Reference to the Device. `DeviceRef` should specify a location rather than an Identifier. Several locations for the same Device can be specified.
`ipmpinfo-msx:ToolBodyID`	Identifier of the Device. Multiple identifiers for the same Device can be specified.

5.2.3.11.4 Additional validation rules

5.2.3.11.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.11.4.2 If multiple `DeviceRef` elements are specified, they must all point to the same Device.

5.2.3.11.4.3 If multiple `ipmpinfo-msx:ToolBodyID` elements are specified, they must all identify the same Device.

5.2.3.12 LicenseEntityType

5.2.3.12.1 Introduction

Throughout MPEG-M, Licenses are in general either referenced or included in the `LicenseEntityType`. Licenses are represented as ISO/IEC 21000-5 `rel-r:license`.

5.2.3.12.2 Syntax

```
<complexType name="LicenseEntityType">
  <complexContent>
    <extension base="mpegm:EntityBaseType">
      <choice>
        <element ref="rel-r:license"/>
        <element name="LicenseRef" type="anyURI" maxOccurs="unbounded"/>
        <element ref="mpegmb:LicenseIdentifier" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.12.3 Semantics

Semantics of the `LicenseEntityType`:

Name	*Definition*
`LicenseEntityType`	Container for a License or a reference to it.
`rel-r:license`	MPEG-21 REL license.
`LicenseRef`	Reference to the License. `LicenseRef` should specify a location rather than an Identifier. Several locations for the same License can be specified.
`LicenseIdentifier`	Identifier of the License. Multiple identifiers for the same License can be specified.

5.2.3.12.4 Additional validation rules

5.2.3.12.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.12.4.2 If multiple `LicenseRef` elements are specified, they must all point to copies of the same License.

5.2.3.12.4.3 If multiple `LicenseIdentifier` elements are specified, they must all identify the same License.

5.2.3.13 ServiceTypeEntityType

5.2.3.13.1 Introduction

Throughout MPEG-M, Service Types are in general either referenced or included in the `ServiceTypeEntityType`. Service Types are represented as `ServiceTypeType`, defined in 5.2.3.2.

5.2.3.13.2 Syntax

```
<complexType name="ServiceTypeEntityType">
  <complexContent>
    <extension base="mpegm:EntityBaseType">
      <choice>
        <element name="ServiceType" type="mpegm:ServiceTypeType"/>
        <element name="ServiceTypeRef" type="anyURI" maxOccurs="unbounded"/>
        <element ref="mpegmb:ServiceTypeIdentifier" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.13.3 Semantics

Semantics of the `ServiceTypeEntityType`:

Name	*Definition*
`ServiceTypeEntityType`	Container for a Service Type or a reference to it.
`ServiceType`	Service Type provided as inline definition.
`ServiceTypeRef`	Reference to the Service Type. `ServiceTypeRef` should specify a location rather than an Identifier. Several locations for the same Service Type can be specified.
`ServiceTypeIdentifier`	Identifier of the Service Type. Multiple identifiers for the same Service Type can be specified.

5.2.3.13.4 Additional validation rules

5.2.3.13.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.13.4.2 If multiple `ServiceTypeRef` elements are specified, they must all point to the same Service Type.

5.2.3.13.4.3 If multiple `ServiceTypeIdentifier` elements are specified, they must all identify the same Service Type.

5.2.3.14 UserEntityType

5.2.3.14.1 Introduction

Throughout MPEG-M, Users are in general either referenced or included in the `UserEntityType`. Users are represented as ISO/IEC 15938-5 `mpeg7:AgentType`.

5.2.3.14.2 Syntax

```
<complexType name="UserEntityType">
  <complexContent>
    <extension base="mpegm:EntityBaseType">
      <choice>
        <element name="User" type="mpeg7:AgentType"/>
        <element name="UserRef" type="anyURI" maxOccurs="unbounded"/>
        <element ref="mpegmb:UserIdentifier" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.14.3 Semantics

Semantics of the `UserEntityType`:

Name	*Definition*
`UserEntityType`	Container for a User or a reference to it.
`User`	Description of a User provided as inline definition.
`UserRef`	Reference to the User. `UserRef` should specify a location rather than an Identifier. Several locations for the same User can be specified.
`UserIdentifier`	Identifier of the User. Multiple identifiers for the same User can be specified.

5.2.3.14.4 Additional validation rules

5.2.3.14.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.3.14.4.2 If multiple `UserRef` elements are specified, they must all point to the same User.

5.2.3.14.4.3 If multiple `UserIdentifier` elements are specified, they must all identify the same User.

5.2.3.15 RequestParametersBaseType

5.2.3.15.1 Introduction

The request parameters of all Service Types of the Process Content Elementary Service (see 5.16.1) are based on the `RequestParametersBaseType`.

5.2.3.15.2 Syntax

```
<!-- Definition of RequestParametersBaseType -->
<complexType name="RequestParametersBaseType" abstract="true"/>
```

5.2.3.15.3 Semantics

Semantics of the `RequestParametersBaseType`:

Name	*Definition*
`RequestParametersBaseType`	Base type for all request parameters of the Process Content Elementary Service.

5.2.3.16 CompletionParametersBaseType

5.2.3.16.1 Introduction

The completion parameters of all Service Types of the Process Content Elementary Service (see 5.16.1) are based on the `CompletionParametersBaseType`.

5.2.3.16.2 Syntax

```
<!-- Definition of CompletionParametersBaseType -->
<complexType name="CompletionParametersBaseType" abstract="true"/>
```

5.2.3.16.3 Semantics

Semantics of the `CompletionParametersBaseType`:

Name	*Definition*
`CompletionParametersBaseType`	Base type for all completion parameters of the Process Content Elementary Service.

5.2.3.17 SupportedMetadataSchemaType

5.2.3.17.1 Introduction

Depending on the strictness of metadata support indicated in the SID (see Annex C), various metadata representations can be deployed in MPEG-M. Users can indicate supported metadata schemas and their respective preferences through the `SupportedMetadataSchemaType`.

5.2.3.17.2 Syntax

```
<!-- Definition of SupportedMetadataSchemaType -->
<complexType name="SupportedMetadataSchemaType">
  <simpleContent>
    <extension base="anyURI">
      <attribute name="pref" type="mpeg7:zeroToOneType" use="required"/>
    </extension>
  </simpleContent>
</complexType>
```

5.2.3.17.3 Semantics

Semantics of the `SupportedMetadataSchemaType`:

Name	*Definition*
`SupportedMetadataSchemaType`	The `xsd:targetNamespace` of a supported metadata schema for the Entity description.
`pref`	Indicates the User's preference of this metadata schema. Higher numbers indicate higher preferences.

5.2.3.18 SupportedTransferProtocolType

5.2.3.18.1 Introduction

Users can indicate supported transfer protocols and their respective preferences through the `SupportedTransferProtocolType`.

5.2.3.18.2 Syntax

```
  <!-- Definition of SupportedTransferProtocolType -->
  <complexType name="SupportedTransferProtocolType">
    <sequence>
      <element name="TransferProtocol" type="mpeg7:ControlledTermUseType">
        <annotation>
          <documentation xml:lang="en">
            Proposed Classification Scheme: urn:mpeg:mpegM:cs:03-es-
NS:2012:TransferProtocolCS
          </documentation>
        </annotation>
      </element>
      <element name="Option" type="mpegmb:KeyValueDataType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="pref" type="mpeg7:zeroToOneType"/>
  </complexType>
```

5.2.3.18.3 Semantics

Semantics of the `SupportedTransferProtocolType`:

Name	*Definition*
`SupportedTransferProtocolType`	Indication of a supported transfer protocol and it preference.
`TransferProtocol`	Specifies the transfer protocol using an MPEG-7 Classification Scheme. Terms for the `TransferProtocol` are specified by the TransferProtocolCS (`urn:mpeg:mpegM:cs:03-es-NS:2012:TransferProtocolCS`) in G.4.
`Option`	Application-specific options for this transfer protocol.
`pref`	Indicates the User's preference of this transfer protocol. Higher numbers indicate higher preferences.

5.2.3.19 RevocationReasonType

5.2.3.19.1 Introduction

The `RevocationReasonType` allows for the specification of a reason for a revocation.

5.2.3.19.2 Syntax

```
<!-- Definition of RevocationReasonType -->
<simpleType name="RevocationReasonType">
  <union>
    <simpleType>
      <restriction base="NMTOKEN">
        <enumeration value="TERMINATE"/>
        <enumeration value="CANCEL"/>
        <enumeration value="ABORT"/>
      </restriction>
    </simpleType>
    <simpleType>
      <restriction base="mpeg7:termReferenceType"/>
    </simpleType>
  </union>
</simpleType>
```

5.2.3.19.3 Semantics

Semantics of the `RevocationReasonType`:

Name	*Definition*
`RevocationReasonType`	Reason of a revocation chosen from a predefined set: TERMINATE, CANCEL, or ABORT. The possible values are defined in Table 6. Other values that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.
`OtherReason`	Descriptive information of any other revocation reason.

The reason of revocation can be one of the following:

Table 6 — Possible values of the `RevocationType`

Protocol Code	Definition
TERMINATE	Terminate naturally the Entity (i.e., contract or license). The requester must be entitled to do it, otherwise it would be making an abortion.
CANCEL	Ask for an agreed cancellation of an Entity (i.e., contract or license).
ABORT	Abort the Entity (i.e., contract or license) unilaterally, being then subject to the penalties for abortion foreseen in the breachment clauses.

5.2.3.20 EntityOrderInformationType

5.2.3.20.1 Introduction

The `EntityOrderInformationType` comprises information about an order in a transaction.

5.2.3.20.2 Syntax

```
  <!-- Definition of EntityOrderInformationType -->
  <complexType name="EntityOrderInformationType" abstract="true">
    <sequence>
      <annotation>
        <documentation xml:lang="en">
          Each extension puts here a transacted Entity of the appropriate type
(e.g., mpegmb:ContentEntityType).
        </documentation>
      </annotation>
      <element name="OrderID" type="anyURI"/>
      </element>
      <element name="Buyer" type="mpegmb:UserEntityType"/>
      <element name="TransactionDate" type="dateTime" minOccurs="0"/>
      <sequence minOccurs="0">
        <element name="Amount" type="rel-sx:Rate"/>
        <element name="VAT" type="rel-sx:Rate" minOccurs="0"/>
      </sequence>
    </sequence>
  </complexType>
```

5.2.3.20.3 Semantics

Semantics of the `EntityOrderInformationType`:

Name	*Definition*
`OrderInformationType`	Collection of information about the order in a transaction. NOTE Any extension of this type includes a transacted Entity of the appropriate type (e.g., `mpegmb:ContentEntityType`)
`OrderID`	Identifier of the order.
`Buyer`	Information on the buyer.
`TransactionDate`	Date when the transaction was completed. Only used if the transaction has already been completed.
`Amount`	Amount of the transaction (including any VAT).
`VAT`	Value Added Tax (VAT) of the transaction.

5.2.3.21 ContentOrderInformationType

5.2.3.21.1 Introduction

The `ContentOrderInformationType` comprises information about a content order in a transaction.

5.2.3.21.2 Syntax

```
<!-- Definition of ContentOrderInformationType -->
<complexType name="ContentOrderInformationType">
  <complexContent>
    <extension base="mpegmb:EntityOrderInformationType">
      <sequence>
        <element name="TransactedContent" type="mpegmb:ContentEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.21.3 Semantics

Semantics of the `ContentOrderInformationType`:

Name	*Definition*
`ContentOrderInformationType`	Collection of information about the content order in a transaction.
`TransactedContent`	Content being transacted.

5.2.3.22 LicenseOrderInformationType

5.2.3.22.1 Introduction

The `LicenseOrderInformationType` comprises information about a license order in a transaction.

5.2.3.22.2 Syntax

```
<!-- Definition of LicenseOrderInformationType -->
<complexType name="LicenseOrderInformationType">
  <complexContent>
    <extension base="mpegmb:EntityOrderInformationType">
      <sequence>
        <element name="TransactedLicense" type="mpegmb:LicenseEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.22.3 Semantics

Semantics of the `LicenseOrderInformationType`:

Name	*Definition*
`LicenseOrderInformationType`	Collection of information about the license order in a transaction.
`TransactedLicense`	License being transacted.

5.2.4 Base Protocol

5.2.4.1 Introduction

All the complex types of messages exchanged in protocols for MPEG-M Services extend from the complex type `ProtocolBaseType`. The `ProtocolBaseType` defines elements that are used in all protocol messages.

5.2.4.2 Syntax

```
  <!-- Definition of ProtocolBaseType. Base type for all protocol messages. -->
  <complexType name="ProtocolBaseType" abstract="true">
    <sequence>
      <element name="TransactionIdentifier" type="string"/>
      <element name="Timestamp" type="dateTime" minOccurs="0"/>
      <element name="Entry" type="mpegmb:KeyValueDataType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="ApplicationSpecificData"
type="mpegmb:ApplicationSpecificDataType" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="dsig:Signature" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="KeyValueDataType">
    <simpleContent>
      <extension base="string">
        <attribute name="key" type="mpeg7:termReferenceType" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="ApplicationSpecificDataType">
    <sequence>
      <any namespace="##other" processContents="lax"/><!-- May contain any
application-specific XML element. -->
    </sequence>
  </complexType>
```

5.2.4.3 Semantics

Semantics of the `ProtocolType`:

Name	*Definition*
`ProtocolType`	Base type for all the protocol message types.
`TransactionIdentifier`	Conveys a value which is used to track the relation between exchanged messages. Any message in response to another message shall specify the same `TransactionIdentifier` value contained in the request.
`Timestamp`	The precise date and time when the message is sent.
`Entry`	An arbitrary property-value pair for application-specific data.
`ApplicationSpecificData`	Other application-specific data.

`dsig:Signature`	Signature of the protocol message. If the signature is present, it signs the entire protocol message and not just the elements of the `ProtocolType`.

Semantics of the `KeyValueType`:

Name	*Definition*
`KeyValueType`	Type for an application-specific property-value pair. The value is contained in the text node of the element.
`key`	Identifying the property of this property-value pair. The key references an MPEG-7 term using a URI. The term should be defined in an MPEG-7 Classification Scheme. NOTE Implementations may define their own application-specific Classification Schemes.

Semantics of the `ApplicationSpecificDataType`:

Name	*Definition*
`ApplicationSpecificDataType`	Container for any application-specific XML-based data structures.

5.2.4.4 Examples

EXAMPLE 1 The example below shows an `Entry` element that specifies the start time for a Service session. A client may specify such a start time in the request message in order to instruct the SP to start the processing of the request at a certain time.

```
<Entry key="urn:example:time-cs:start-time">2011-03-16T14:28:00</Entry>
```

EXAMPLE 2 The example below shows an `ApplicationSpecificData` element that specifies a Service Level Agreement (SLA) for the Service session. The client may specify additional requirements such as network properties, transport mechanism, timeout for delivery, size of content to be delivered, etc. for Deliver Content.

```
<ApplicationSpecificData>
  <ns1:SLA>
    <ns1:Timeout>PT3M30S</ns1:Timeout>
    <!-- Specification of further requirements... -->
  </ns1:SLA>
</ApplicationSpecificData>
```

5.2.5 Common messages

5.2.5.1 Introduction

Many of the protocols define a request and a response messages. Some of the elements in this message are common and they can be abstracted.

Response message make use of either a *success acknowledgement* (represented by the `ProtocolSuccessType`) or an *error information* (represented by the `ProtocolFailureType`).

All protocols, except the Authenticate User elementary service defined in 5.5.3.5, require that Client and Service Provider mutually authenticate before the first protocol step. This authentication may be either explicit (e.g., authentication protocol) or implicit (e.g., trusted environment). This Part of ISO/IEC 23006 does not specify the mechanism of the authentication. One possible authentication mechanism is the Authenticate User elementary service as defined in 5.5.3.5. Note that ISO/IEC 23006-2 specifies profiles for the authentication mechanism.

5.2.5.2 Syntax

```
  <!-- Definition of ProtocolRequestType -->
  <complexType name="ProtocolRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolBaseType">
        <sequence>
          <element ref="saml:Assertion" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of ProtocolResponseType -->
  <complexType name="ProtocolResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolBaseType">
        <!-- Each Elementary Service appends here the choice between
[ServiceName]Sucess and [ServiceName]Failure. -->
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ProtocolSuccessType">
    <sequence>
      <element name="SuccessCode" type="mpegmb:SuccessCodeType" minOccurs="0"/>
      <element name="DisplayString" type="string" minOccurs="0"/>
    </sequence>
  </complexType>

  <simpleType name="SuccessCodeType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="OK"/>
          <enumeration value="CREATED"/>
          <enumeration value="ACCEPTED"/>
          <enumeration value="NON-AUTHORITATIVE_CONTENT"/>
          <enumeration value="NO_CONTENT"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>

  <complexType name="ProtocolFailureType">
    <sequence>
      <element name="FailureCode" type="mpegmb:FailureCodeType" minOccurs="0"/>
      <element name="DisplayString" type="string" minOccurs="0"/>
    </sequence>
```

```
  </complexType>

  <simpleType name="FailureCodeType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="PERMISSION_DENIED"/>
          <enumeration value="TIMEOUT"/>
          <enumeration value="BUSY"/>
          <enumeration value="MALFORMED_REQUEST"/>
          <enumeration value="UNABLE_TO_PROCESS"/>
          <enumeration value="VERIFICATION_FAILED"/>
          <enumeration value="OPERATION_NOT_SUPPORTED"/>
          <enumeration value="UNKNOWN_MESSAGE"/>
          <enumeration value="UNKNOWN_ERROR"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>
```

5.2.5.3 Semantics

Semantics of the `ProtocolRequestType`:

Name	*Definition*
`ProtocolRequestType`	Base type for all the request messages issued in the course of the protocols.
`saml:Assertion`	Security token with an authorization statement that proves that the Service invoker is actually authorized to invoke it.

Semantics of the `ProtocolResponseType`:

Name	*Definition*
`ProtocolResponseType`	Base type for all the response messages issued in the course of the protocols.

Semantics of the `ProtocolSuccessType`:

Name	*Definition*
`ProtocolSuccessType`	Base type for the main response message part issued in case of success.
`SuccessCode`	Indicates the type success.
`DisplayString`	Optional text with more information on the result to be displayed to the user.

SuccessCodeType	Success codes for events that can appear in any protocol. The possible values are defined in Table 7. Other values that are datatype-valid with respect to mpeg7:termReferenceType are reserved.

Table 7 below specifies the possible values for the SuccessCodeType.

Table 7 — Possible values for the SuccessCodeType simple type

Success Code	Definition
OK	The operation was performed as requested.
CREATED	The operation was performed as requested and resulted in a new entity being created.
ACCEPTED	The request has been accepted for processing but the actual operation has not yet been completed.
NON-AUTHORITATIVE_INFORMATION	The operation was performed as requested but the returned information may be from another User.
NO_CONTENT	The operation was performed as requested but the response contains no further information, e.g., the client already has the latest information.

Semantics of the ProtocolFailureType:

Name	*Definition*
ProtocolFailureType	Base type for the main response message part issued in case of failure.
FailureCode	Indicates the type of failure.
DisplayString	Text with more information on the result to be displayed to the user.
FailureCodeType	Failure codes for events that can appear in any protocol. The possible values are defined in Table 8. Other values that are datatype-valid with respect to mpeg7:termReferenceType are reserved.

Table 8 below specifies the possible values for the `FailureCodeType`.

Table 8 — Possible values for the `FailureCodeType` simple type

Failure Code	Definition
PERMISSION_DENIED	The client is not allowed to carry out the requested operation.
BUSY	The requested operation cannot be performed because the addressee is busy.
MALFORMED_REQUEST	The request message is malformed or incomplete.
UNABLE_TO_PROCESS	The requested operation is supported by the addressee, however the addressee is not able to process the request for an unknown reason.
OPERATION_NOT_SUPPORTED	The requested operation is not supported by the addressee.
UNKNOWN_MESSAGE	The sent message was not recognized by the addressee.
UNKNOWN_ERROR	An unknown error occurred.

5.2.5.4 Additional validation rules

5.2.5.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.2.5.4.2 Each complex type extending the `ProtocolResponseType` must contain a `choice` between an element of complex type `ProtocolSuccessType` and an element of complex type `ProtocolFailureType` or subtypes of them. These elements are named `<ServiceName>Success` and `<ServiceName>Failure` respectively, where `<ServiceName>` is replaced by the camel case version of the name of the Service (e.g., `ProcessContent` for "Process Content").

5.3 MPEG-M service Schema wrapper

The syntax of description tools specified in this clause is provided as a collection of schema components, consisting notably in type definitions and element declarations. In order to form a valid schema document, these schema components are gathered in a same document with the following declaration defining in particular the target namespace and the namespaces prefixes.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:mpegm="urn:mpeg:mpegM:schema:02-service-NS:2012"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
xmlns:didl="urn:mpeg:mpeg21:2006:07-DIDL-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:erl="urn:mpeg:mpeg21:2005:01-ERL-NS"
xmlns:fru="urn:mpeg:mpegB:schema:FragmentRequestUnits:2007"
xmlns:ipmpinfo="urn:mpeg:mpeg21:2004:01-IPMPINFO-NS"
xmlns:ipmpmsg="urn:mpeg:mpegB:schema:IPMP-XML-MESSAGES:2007"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xmlns:rel-r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:cel="urn:mpeg:mpeg21:cel:core:2012"
```

```
xmlns:bbl="urn:mpeg:mpeg21:2007:01-BBL-NS"
xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:rel-sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:mpeg4ipmp="urn:mpeg:mpeg4:IPMPSchema:2002"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xhtml="http://www.w3.org/1999/xhtml"
targetNamespace="urn:mpeg:mpegM:schema:02-service-NS:2012"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="ISO/IEC 23006-4 2nd edition" id="mpeg-m.xsd">
  <import namespace="urn:mpeg:mpegM:schema:01-base-NS:2012" schemaLocation="mpeg-
m-base.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2006:07-DIDL-NS"
schemaLocation="mpeg/didl.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2005:01-ERL-NS"
schemaLocation="mpeg/erl.xsd"/>
  <import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="mpeg/mpeg7-
v3.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2004:01-IPMPINFO-NS"
schemaLocation="mpeg/ipmpinfo.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2003:01-REL-R-NS" schemaLocation="mpeg/rel-
r.xsd"/>
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="w3c/xmldsig-core-schema.xsd"/>
  <import namespace="urn:mpeg:mpegB:schema:IPMP-XML-MESSAGES:2007"
schemaLocation="mpeg/ipmpmsg.xsd"/>
  <import namespace="urn:mpeg:mpeg21:cel:core:2012" schemaLocation="cel-
core.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2003:01-REL-SX-NS" schemaLocation="mpeg/rel-
sx.xsd"/>
  <import namespace="urn:mpeg:mpeg4:IPMPSchema:2002"
schemaLocation="mpeg/mpeg4ipmp.xsd"/>
  <import namespace="urn:mpeg:mpegB:schema:FragmentRequestUnits:2007"
schemaLocation="mpeg/fru.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2007:01-BBL-NS"
schemaLocation="http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
21_schema_files/dis/bbl.xsd"/>
  <import namespace="urn:mpeg:mpqf:schema:2008" schemaLocation="mpeg/mpqf.xsd"/>
  <import namespace="http://www.w3.org/1999/xhtml"
schemaLocation="w3c/xhtml11.xsd"/>
  <import namespace="http://www.w3.org/1999/XSL/Transform"
schemaLocation="w3c/schema-for-xslt20.xsd"/>
  <import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
schemaLocation="oasis/saml-schema-assertion-2.0.xsd"/>
  <import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
schemaLocation="oasis/saml-schema-protocol-2.0.xsd"/>
  <import namespace="urn:mpeg:mpeg21:2003:01-DIA-NS"
schemaLocation="http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
21_schema_files/dia-2nd/UED-2nd.xsd"/>
```

Additionally, the following line should be appended to the resulting schema document in order to obtain a well-formed XML document.

```
</schema>
```

5.4 Session control and status polling

The following messages can be sent from the client to the SP during a protocol run.

5.4.1 Session control messages

5.4.1.1 Introduction

This part describes a session control protocol that can signal three types of control actions, namely Pause, Resume, and Teardown. This message can be used to control any session of MPEG-M protocols.

5.4.1.2 Interfaces and protocol specification

This protocol is employed by a User to request another User to do a control action (i.e., pause, resume, teardown) to a session between the two Users. The protocol is as follows:

Steps	Client	Service Provider
1.	Client and Service Provider start some Service Instance (e.g., Deliver Content).	
2.	If the client wants to teardown, pause, or resume the session, the client sends a `ControlRequest` message in order to request a control action for the session. The `TransactionRef` of this message references the `TransactionIdentifier` of the session to which the control action should be applied. The message contains the `Control` element indicating the type of control ("PAUSE", "RESUME", or "TEARDOWN"), and optionally a `Reason` element indicating the reason for that control request. In addition, the `responseRequired` attribute indicates whether an acknowledgement from the Service Provider is required ("true") or not required ("false").	
3.		Upon receipt, the Service Provider evaluates, whether the request can be met. If yes, the Service Provider carries out the requested control action.
4.		If the `responseRequired` attribute in the request message is "true", the Service Provider sends a `ControlResponse` message, in which the `ControlResult` element conveys the result (success, failure) of the control action.

5.4.1.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Control Request (Teardown/Pause/Resume)         -->
<!-- ################################################ -->

<!-- Definition of ControlRequest -->
<element name="ControlRequest" type="mpegm:ControlRequestType"/>
<complexType name="ControlRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
         <element name="TransactionRef" type="string"/>
         <element name="Control" type="mpegm:ControlType"/>
         <element name="Reason" type="string" minOccurs="0"/>
      </sequence>
      <attribute name="responseRequired" type="boolean" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="ControlType">
  <union>
    <simpleType>
      <restriction base="NMTOKEN">
        <enumeration value="PAUSE"/>
        <enumeration value="RESUME"/>
        <enumeration value="TEARDOWN"/>
      </restriction>
    </simpleType>
    <simpleType>
      <restriction base="mpeg7:termReferenceType"/>
    </simpleType>
  </union>
</simpleType>

<element name="ControlResponse" type="mpegm:ControlResponseType"/>
<complexType name="ControlResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
      <choice>
        <element name="ControlSuccess" type="mpegmb:ProtocolSuccessType"/>
        <element name="ControlFailure" type="mpegmb:ProtocolFailureType"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.4.1.4 Semantics of protocol data format

Semantics of the `ControlRequest`:

Name	*Definition*
`ControlRequest`	Protocol message sent from a Client to a Service Provider to request a control action to the transaction.
`ControlRequestType`	Top-level type for `ControlRequest`. `ControlRequestType` extends `ProtocolRequestType`.
`Control`	Indicates a requested control action.
`Reason`	Indicates the reason for the requested control action.
`responseRequired`	Indicates whether a response message is required.

Semantics of the `ControlType`:

Name	*Definition*
`ControlType`	Top-level type for identifying a control action It can have one of the following values. — PAUSE: pause a session. — RESUME: resume a previously paused a session — TEARDOWN: teardown a session. The session cannot be resumed later. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

Semantics of the `ControlResponse`:

Name	*Definition*
`ControlResponse`	Protocol message sent from a User to another User in response to a control request.
`ControlResponseType`	Top-level type for `ControlResponse`. `ControlResponseType` extends `ProtocolResponseType`.
`ControlSuccess`	Response in case of success.
`ControlFailure`	Response in case of failure.

5.4.1.5 Example

This example shows a control message to teardown a session. Suppose that the client has started a session of Deliver Content, which is identified by the `TransactionIdentifier` set to "5". Due to some emergency reason, the client has to stop the content delivery. For this, the client sends a `ControlRequest` message to the Service Provider, with the `TransactionRef` set to "5" and the `Control` element set to "TEARDOWN". The `responseRequired` attribute is set to "true", meaning that the client expects a response message from the Service Provider.

```
<ControlRequest responseRequired="true">
  <mpegmb:TransactionIdentifier>666</mpegmb:TransactionIdentifier>
  <TransactionRef>5</TransactionRef>
  <Control>TEARDOWN</Control>
</ControlRequest>
```

The Service Provider stops the referenced session of Deliver Content and responds with a ControlResponse message.

```
<ControlResponse>
  <mpegmb:TransactionIdentifier>666</mpegmb:TransactionIdentifier>
  <ControlSuccess>
    <DisplayString>
      TEARDOWN for Deliver Content (TransactionIdentifier "5") executed
succesfully.
    </DisplayString>
  </ControlSuccess>
</ControlResponse>
```

5.4.2 Session status polling messages

5.4.2.1 Introduction

This part describes a session status polling protocol that allows a Client to repeatedly gather the status of an MSPT protocol session. Status polling is especially useful for long-running Services (e.g., Deliver Content, Process Content, and Store Content), but can be applied to any Service.

5.4.2.2 Interfaces and protocol specification

The protocol is as follows:

Steps	Client	Service Provider
1.	Client and SP start some Service Instance (e.g., Deliver Content).	
2.	If the Client wants to know the status of the Service session, it sends a `StatusRequest` message to the SP.	
3.		Upon receipt, the SP evaluates, whether the request can be met and sends a `StatusResponse` message to the client, indicating the status of the referenced Service session.

5.4.2.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Status Polling                                  -->
<!-- ################################################ -->

<!-- Definition of StatusRequest -->
<element name="StatusRequest" type="mpegm:StatusRequestType"/>
<complexType name="StatusRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="TransactionRef" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>


<!-- Definition of StatusResponse -->
<element name="StatusResponse" type="mpegm:StatusResponseType"/>
<complexType name="StatusResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
      <sequence>
        <choice>
          <element name="StatusSuccess" type="mpegm:StatusSuccessType"/>
          <element name="StatusFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Definition of StatusSuccessType -->
<complexType name="StatusSuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolSuccessType">
      <sequence>
        <element name="ProgressStatus" type="mpegm:ProgressStatusType"/>
      </sequence>
      <attribute name="completion" type="mpeg7:zeroToOneType" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="ProgressStatusType">
  <union>
    <simpleType>
      <restriction base="NMTOKEN">
        <enumeration value="TO_BE_PERFORMED"/>
        <enumeration value="CURRENTLY_IN_PROGRESS"/>
        <enumeration value="PROGRESS_PAUSED"/>
        <enumeration value="SUCCESSFULLY_PERFORMED"/>
        <enumeration value="UNSUCCESSFULLY_TERMINATED"/>
        <enumeration value="SESSION_UNKNOWN"/>
      </restriction>
    </simpleType>
    <simpleType>
```

```
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>
```

5.4.2.4 Semantics of protocol data format

Semantics of the `StatusRequest`:

Name	*Definition*
`StatusRequest`	Protocol message sent from the client to the SP to repeatedly poll for the status of a Service session.
`StatusRequestType`	Top-level type for `StatusRequest`. `StatusRequestType` extends `ProtocolRequestType`.
`TransactionRef`	Conveys the value of the `TransactionIdentifier` of the message for which the status is polled.

Semantics of the `StatusResponse`:

Name	*Definition*
`StatusResponse`	Protocol message sent from the SP to the client in response to a `StatusRequest` message to indicate the current status of the specified session.
`StatusResponseType`	Top-level type for `StatusResponse`. `ResponseType` extends `ProtocolResponseType`.
`StatusSuccess`	Response in case of success.
`StatusFailure`	Response in case of failure.

Semantics of the `StatusSuccessType`:

Name	*Definition*
`StatusSuccessType`	Type of the response message part that is provided in case of success.
`ProgressStatus`	States the current status of the Service session.
`completion`	A `float` value indicating the fraction of the operation that has already been performed. The attribute is only used if the `ProgressStatus` is set to "CURRENTLY_IN_PROGRESS" or "PROGRESS_PAUSED". The value "1.0" indicates that the operation has been performed completely.

Semantics of the `ProgressStatusType`:

Name	*Definition*
`ProgressStatusType`	Describes the status of the Service session. The possible values are defined in Table 9. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

The table below specifies the possible values for the `ProgressStatusType`.

Table 9 — Possible values of the `ProgressStatusType` simple type

Protocol Code	Definition
TO_BE_PERFORMED	The SP is waiting to perform the specified Service session, which has not been initiated yet.
CURRENTLY_IN_PROGRESS	The specified operation is currently being performed.
PROGRESS_PAUSED	The specified Service session is currently paused.
SUCCESSFULLY_PERFORMED	The specified Service session has been successfully performed.
UNSUCCESSFULLY_TERMINATED	The specified Service session has been terminated with an error.
SESSION_UNKNOWN	There is no record for the specified `TransactionRef` at the SP.

5.4.2.5 Additional validation rules

5.4.2.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.4.2.5.2 The `completion` attribute may only be present if the `ProgressStatus` is set to "CURRENTLY_IN_PROGRESS" or "PROGRESS_PAUSED".

5.4.2.6 Example

This example shows the messages for polling the status of a session. Suppose that the client has started a Deliver Content session with the `TransactionIdentifier` set to "123". The client wants to know the current status of the content delivery and sends a `StatusRequest` message to the SP.

```
<StatusRequest>
  <mpegmb:TransactionIdentifier>124</mpegmb:TransactionIdentifier>
  <TransactionRef>123</TransactionRef>
</StatusRequest>
```

The SP has already finished 30 % of the content delivery and responds with a `StatusResponse` message.

```
<StatusResponse>
  <mpegmb:TransactionIdentifier>124</mpegmb:TransactionIdentifier>
  <StatusSuccess completion="0.3">
    <ProgressStatus>CURRENTLY_IN_PROGRESS</ProgressStatus>
    <DisplayString>
      3072 of 10240 KB sent.
    </DisplayString>
  </StatusSuccess>
</StatusResponse>
```

5.5 The Authenticate Services

5.5.1 Authenticate Entity

5.5.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Authenticate Entity abstract elementary service. The protocol of this Elementary Service has to be used as a guide for any extra Authenticate Service that may be specified, apart from the ones included in the following Sections.

This abstract Elementary Service enables a User to authenticate an Entity other than User. That is, the client can request the confirmation of the identity and signers of an Entity in a multimedia content value chain.

NOTE The Authenticate User elementary service is not based on the Authenticate Entity abstract elementary service.

5.5.1.2 Interfaces and protocol specification

The Authenticate Entity Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a message of type `AuthenticateEntityRequestType` in order to get or check signature of a given Entity.	
2.		The SP sends a message of type `AuthenticateEntityResponseType` with the result of the authentication. If the authentication has been completed successfully, the message contains an `AuthenticateSuccess` element. The message may provide a signature associated with the Entity.

5.5.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Authenticate Entity                             -->
  <!-- ################################################ -->

  <!-- Definition of AuthenticateEntityRequestType -->
  <complexType name="AuthenticateEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
```

```
        <sequence>
          <element name="AuthenticationEntity" type="mpegmb:EntityBaseType">
            <annotation>
              <documentation xml:lang="en">
                Each regular Elementary Service puts here an Authentication
Entity of the appropriate type (e.g., mpegmb:ContentEntityType).
              </documentation>
            </annotation>
          </element>
          <element name="EntitySignature" type="dsig:SignatureType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>


  <!-- Definition of AuthenticateEntityResponseType -->
  <complexType name="AuthenticateEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AuthenticateEntitySuccess"
type="mpegm:AuthenticateEntitySuccessType"/>
          <element name="AuthenticateEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="AuthenticateEntitySuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="EntitySignature" type="dsig:SignatureType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.5.1.4 Semantics of protocol data format

Semantics of the `AuthenticateEntityRequestType`:

Name	*Definition*
`AuthenticateEntityRequestType`	Base type for Authenticate Entity request messages. `AuthenticateEntityRequestType` extends `ProtocolRequestType`.
`AuthenticationEntity`	The entity to be authenticated.
`EntitySignature`	Signature associated with the Entity in cases where the signature is not already included in the `AuthenticationEntity`.

Semantics of the `AuthenticateEntityResponseType`:

Name	*Definition*
`AuthenticateEntityResponseType`	Top-level type for Authenticate Entity response messages. `AuthenticateResponseType` extends `ProtocolResponseType`.
`AuthenticateEntitySuccess`	Response in case of success.
`AuthenticateEntityFailure`	Response in case of failure.

Semantics of the `AuthenticateEntitySuccessType`:

Name	*Definition*
`EntitySignature`	Optional signature associated with the requested item.

5.5.1.5 Additional validation rules

5.5.1.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.5.1.5.2 Each complex type extending the `AuthenticateEntityRequestType` must contain an authentication entity of the appropriate type (e.g., `mpegmb:ContentEntityType` for Authenticate Content).

5.5.1.6 Extension to SID

For the representation of Service Instance Declarations of *Authenticate Entity*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="AuthenticateEntitySIDType" abstract="true">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.5.2 Authenticate Content

5.5.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Authenticate Content Protocol. The elementary service extends the Authenticate Entity abstract elementary service defined in 5.5.1.

This Elementary Service enables a User to authenticate a Digital Item or any of its components and to retrieve a signature associated with the Digital Item (the signature can be used to check if a content is changed in the mean time).

5.5.2.2 Interfaces and protocol specification

The Authenticate Content Protocol extends the abstract Authenticate Entity Protocol defined in 5.5.1.2, from which it inherits the protocol specification.

5.5.2.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Authenticate Content                            -->
  <!-- ################################################ -->

  <!-- Definition of AuthenticateContentRequest -->
  <element name="AuthenticateContentRequest"
type="mpegm:AuthenticateContentRequestType"/>
  <complexType name="AuthenticateContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="AuthenticationContent" type="mpegmb:ContentEntityType"/>
          <element name="ContentSignature" type="dsig:SignatureType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of AuthenticateContentResponse -->
  <element name="AuthenticateContentResponse"
type="mpegm:AuthenticateEntityResponseType"/>
```

5.5.2.4 Semantics of protocol data format

Semantics of the `AuthenticateContentRequest`:

Name	*Definition*
`AuthenticateContentRequest`	The message for creating an Authenticate Content request.
`AuthenticationContent`	Content to be authenticated.
`ContentSignature`	Signature associated with the Content in cases where the signature is not already included in the `AuthenticationContent`.

Semantics of the `AuthenticateContentResponse`:

Name	*Definition*
`AuthenticateContentResponse`	The response message for Authenticate Content.

5.5.2.5 Extension to SID

For the representation of Service Instance Declarations of *Authenticate Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="AuthenticateContentSIDType">
    <complexContent>
      <extension base="sid:AuthenticateEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.5.3 Authenticate Contract

5.5.3.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Authenticate Contract elementary service. The elementary service extends the Authenticate Entity abstract elementary service defined in 5.5.1.

Authenticate Contract allows Users to confirm the identity and signers of a Contract in a multimedia content value chain. Given that a Contract includes digital signatures, Authenticate a Contract is authenticating the signatures. Note that a Contract in force requires two signatures, an offered contract requires only one, a speculative contract none.

The Protocol to Authenticate Contract is employed by any Entity wishing to verify the authenticity of an MPEG-21 Part 20 Contract.

5.5.3.2 Interfaces and protocol specification

The Authenticate Contract Protocol extends the abstract Authenticate Entity Protocol defined in 5.5.1.2, from which it inherits the protocol specification.

5.5.3.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Authenticate Contract                           -->
  <!-- ################################################ -->

  <!-- Definition of AuthenticateContractRequest -->
  <element name="AuthenticateContractRequest"
type="mpegm:AuthenticateContractRequestType"/>
  <complexType name="AuthenticateContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="AuthenticationContract"
type="mpegmb:ContractEntityType"/>
          <element name="ContractSignature" type="dsig:SignatureType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
```

```
  </complexType>

  <!-- Definition of AuthenticateContractResponse -->
  <element name="AuthenticateContractResponse"
type="mpegm:AuthenticateEntityResponseType"/>
```

5.5.3.4 Semantics of protocol data format

Semantics of the `AuthenticateContractRequest`:

Name	*Definition*
`AuthenticateContractRequest`	The message for creating an Authenticate Contract request.
`AuthenticationContract`	Contract to be authenticated.
`ContractSignature`	Signatures associated with the Contract in cases where the signatures are not already included in the `AuthenticationContract`.

Semantics of the `AuthenticateContractResponse`:

Name	*Definition*
`AuthenticateContractResponse`	The response message for Authenticate Contract.

5.5.3.5 Extension to SID

For the representation of Service Instance Declarations of *Authenticate Contract*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="AuthenticateContractSIDType">
    <complexContent>
      <extension base="sid:AuthenticateEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```


5.5.4 Authenticate License

5.5.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Authenticate License Protocol. The elementary service extends the Authenticate Entity abstract elementary service defined in 5.5.1.

This Elementary Service enables a User to authenticate a License or any of its components and to retrieve a signature associated with the License (the signature can be used to check if a license is changed in the mean time).

5.5.4.2 Interfaces and protocol specification

The Authenticate License Protocol extends the abstract Authenticate Entity Protocol defined in 5.5.1.2, from which it inherits the protocol specification.

5.5.4.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Authenticate License                            -->
  <!-- ################################################ -->

  <!-- Definition of AuthenticateLicenseRequest -->
  <element name="AuthenticateLicenseRequest"
type="mpegm:AuthenticateLicenseRequestType"/>
  <complexType name="AuthenticateLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="AuthenticationLicense" type="mpegmb:LicenseEntityType"/>
          <element name="LicenseSignature" type="dsig:SignatureType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of AuthenticateLicenseResponse -->
  <element name="AuthenticateLicenseResponse"
type="mpegm:AuthenticateEntityResponseType"/>
```

5.5.4.4 Semantics of protocol data format

Semantics of the `AuthenticateLicenseRequest`:

Name	*Definition*
`AuthenticateLicenseRequest`	The message for creating an Authenticate License request.
`AuthenticationLicense`	License to be authenticated.
`LicenseSignature`	Signature associated with the License in cases where the signature is not already included in the `AuthenticationLicense`.

Semantics of the `AuthenticateLicenseResponse`:

Name	*Definition*
`AuthenticateLicenseResponse`	The response message for Authenticate License.

5.5.4.5 Extension to SID

For the representation of Service Instance Declarations of *Authenticate License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="AuthenticateLicenseSIDType">
  <complexContent>
    <extension base="sid:AuthenticateEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.5.5 Authenticate User

5.5.5.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Authenticate User Protocol.

The Authenticate User elementary service enables a client to confirm its identity towards a Service Provider (i.e., to log in at the SP). The "Authenticate User" Service Provider offers a common authentication service for User entities belonging to different domains (Single Sign-on). This protocol is based on the OASIS Security Assertion Markup Language (SAML) V2.0.

The relationship between Authenticate User and Identify User ESs (described in 5.11.6) is as follows. When a User requires authentication, the credentials indicated by him/her in the registration phase are checked to provide the corresponding SAML assertion. In this way, during authentication phase, only Authenticate User ES is required (asking for credentials of the user, as SAML V2.0 describes). Prior to requiring authentication, the User should have registered using Identify User ES.

NOTE The Authenticate User elementary service is not based on the Authenticate Entity abstract elementary service.

5.5.5.2 Interfaces and protocol specification

The Authenticate User Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends an `AuthenticateUserRequest` message in order to log in at the SP. The message contains a `samlp:AuthnRequest` element as specified in 3.4 of SAML-CORE-2.0-OS.	
2.		The SP sends an `AuthenticateUserResponse` message. In case of success, the message contains a `samlp:Response` element with security assertions. The `samlp:Response` contains `saml:Assertion` elements that can be used in subsequent calls to any other Service as explained in 5.2.5.

5.5.5.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Authenticate User                               -->
  <!-- ################################################ -->

  <!-- Definition of AuthenticateUserRequest -->
  <element name="AuthenticateUserRequest"
type="mpegm:AuthenticateUserRequestType"/>

  <complexType name="AuthenticateUserRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element ref="samlp:AuthnRequest"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of AuthenticateUserResponse -->
  <element name="AuthenticateUserResponse"
type="mpegm:AuthenticateUserResponseType"/>
  <complexType name="AuthenticateUserResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <element ref="samlp:Response"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.5.5.4 Semantics of protocol data format

Semantics of the `AuthenticateUserRequest`:

Name	*Definition*
`AuthenticateUserRequest`	The message for creating an Authenticate User request.
`AuthenticateUserRequestType`	Top-level type for `AuthenticateUserRequest`. `AuthenticateUserRequestType` extends `ProtocolRequestType`.
`samlp:AuthnRequest`	Authentication Request defined in SAML.

Semantics of the `AuthenticateUserResponse`:

Name	*Definition*
`AuthenticateUserResponse`	The answer to an Authenticate User message.

`AuthenticateUserResponseType`	Top-level type for `AuthenticateUserResponse`. `AuthenticateUserResponseType` extends `ProtocolResponseType`.
`samlp:Response`	Message response defined by SAML. Contains `saml:Assertion` elements which can be used for the invocation of other Services.

5.5.5.5 Extension to SID

For the representation of Service Instance Declarations of *Authenticate User*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="AuthenticateUserSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.6 The Authorize Services

5.6.1 Authorize User

5.6.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Authorize User elementary service.

Authorize User allows Users to perform certain operations acted on a certain content by triggering the authorization as described in Clause 5 of ISO/IEC 21000-5.

5.6.1.2 Interfaces and protocol specification

The Authorize User Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends an `AuthorizeUserRequest` in order to request the execution of one action (a REL right exercise) over certain content (a REL resource) by a certain User (a REL principal).	
2.		The SP sends an `AuthorizeUserResponse` message answering whether the operation is allowed or not.

5.6.1.3 Syntax of protocol data format

```
<!-- Definition of AuthorizeUserRequest -->
<element name="AuthorizeUserRequest" type="mpegm:AuthorizeUserRequestType"/>
  <complexType name="AuthorizeUserRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
          <element name="Principal" type="rel-r:Principal" minOccurs="0"/>
          <element name="Right" type="rel-r:Right" maxOccurs="unbounded"/>
          <element name="Resource" type="rel-r:Resource" maxOccurs="unbounded"/>
          <element name="Condition" type="rel-r:Condition" maxOccurs="unbounded"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
<!-- Definition of AuthorizeUserResponse -->
<element name="AuthorizeUserResponse" type="mpegm:AuthorizeUserResponseType"/>
  <complexType name="AuthorizeUserResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AuthorizeUserSuccess"
type="mpegm:AuthorizeUserSuccessType"/>
          <element name="AuthorizeUserFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AuthorizeUserSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="Key" type="dsig:KeyInfoType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.6.1.4 Semantics of protocol data format

Semantics of the `AuthorizeUserRequest`:

Name	*Definition*
`AuthorizeUserRequest`	Message sent to start the User authorization.
`LicenseEntity`	A MPEG-21 REL license against which to perform the check. The license may be included either directly, by reference, or by license identifier as specified in 5.2.3.12. NOTE If a licensing service, which is known by the SP, stores the

	licenses for performing user authorization, then the `LicenseEntity` can be omitted.
`Principal`	A MPEG-21 REL principal desiring to exercise a right.
`Right`	An MPEG-21 REL right to be exercised.
`Resource`	An MPEG-21 REL resource over which the right is exercised.
`Condition`	An MPEG-21 REL condition stating the conditions that should be considered for the evaluation of the request.

Semantics of the `AuthorizeUserResponse`:

Name	*Definition*
`AuthorizeUserResponse`	The response message for Authorize User, which determines whether the operation was allowed or not.
`AuthorizeUserSuccess`	Response in case of success.
`AuthorizeUserFailure`	Response in case of failure.

Semantics of the `AuthorizeUserSuccessType`:

Name	*Definition*
`Key`	Optional key to decrypt a certain resource or perform other operations.

5.6.1.5 Extension to SID

For the representation of Service Instance Declarations of *Authorize User*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="AuthorizeUserSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.7 The Check With Services

5.7.1 Check With Contract

5.7.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Check With Contract elementary service.

Check With Contract allows Users to verify if a usage request matches with the content (e.g., obligations, prohibitions) expressed in a Contract.

The evaluation of the contract clauses takes place in the SP. The SP must not change its state by any means as a result of a Check With ES (e.g., even if there is a clause where the consumption of a right is limited to a certain number of times, the account will not be decreased).

The "Check With Contract" Protocol takes place between the SP and one of the contract signers, which always initiates the dialog. The operation will not be performed if the Service invoker is different from one of the signers.

5.7.1.2 Interfaces and protocol specification

The Check With Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `CheckWithContractRequest` message to check a given contract.	
2.		The SP sends a `CheckWithContractResponse` with the result of the check with operation. It can also provide a signature associated with the item requested.

5.7.1.3 Syntax of protocol data format

```
<element name="CheckWithContractRequest"
type="mpegm:CheckWithContractRequestType"/>
  <complexType name="CheckWithContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType"/>
          <element name="Action" type="mpegm:ActionType"/>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <simpleType name="ActionType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
```

```
          <enumeration value="AUDIOVISUAL_DISTRIBUTE"/>
          <enumeration value="MAKE_ADAPTATION"/>
          <enumeration value="MAKE_EXCERPT"/>
          <enumeration value="REMIX"/>
          <enumeration value="MAKE_INSTANCE"/>
          <enumeration value="DUPLICATE"/>
          <enumeration value="PRODUCE"/>
          <enumeration value="PUBLIC_COMMUNICATION"/>
          <enumeration value="COMMUNICATION_TO_THE_PUBLIC"/>
          <enumeration value="FIXATE"/>
          <enumeration value="PUBLIC_PERFORMANCE"/>
          <enumeration value="BROADCAST"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>

<element name="CheckWithContractResponse"
type="mpegm:CheckWithContractResponseType"/>
<complexType name="CheckWithContractResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
      <sequence>
        <choice>
          <element name="CheckWithContractSuccess"
type="mpegm:CheckWithContractSuccessType"/>
          <element name="CheckWithContractFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>

  <complexType name="CheckWithContractSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="CheckWithContractResultCode"
type="mpegm:CheckWithContractResultCodeType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <simpleType name="CheckWithContractResultCodeType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="OK"/>
          <enumeration value="USER_NOT_FOUND"/>
          <enumeration value="DEONTIC_EXPRESSION_NOT_FOUND"/>
          <enumeration value="CONDITION_NOT_SATISFIED"/>
          <enumeration value="UNKNOWN_ERROR"/>
        </restriction>
      </simpleType>
      <simpleType>
```

```
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>
```

5.7.1.4 Semantics of protocol data format

Semantics of the `CheckWithContractRequest`:

Name	*Definition*
`CheckWithContractRequestType`	Message sent to start the Check With service.
`ContractEntity`	A MPEG-21 CEL contract as described in 5.2.3.10.
`ContentEntity`	Digital Item on which the requested operation should take place.
`Action`	One of the actions defined in Table 11.
`ActionType`	Enumeration of possible Actions. The definitions of the Actions are given in Table 11. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

Semantics of the `CheckWithContractResponse`:

Name	*Definition*
`CheckWithContractResponseType`	Message sent as a result of the Check With service invocation.
`CheckWithContractSuccess`	Response in case of success.
`CheckWithContractFailure`	Response in case of failure.

Semantics of the `CheckWithContractSuccessType`:

Name	*Definition*
`CheckWithContractSuccessType`	Type of the response message part that is provided in case of success. NOTE The `CheckWithContractSuccess` element only means that the protocol was carried out successfully. The result of the Check With operation is indicated in the `CheckWithContractResult` element.
`CheckWithContractResult`	The result of the operation.
`CheckWithContractResultCodeType`	Type to convey the specific information about the result of this operation. Possible result codes are provided in Table 10. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

The possible values of the `CheckWithContractResultCodeType` are given in Table 10.

Table 10 — Possible values of the `CheckWithContractResultCodeType`

Result Code	Definition
OK	The User is allowed to act on the content according to the request.
USER_NOT_FOUND	The User is not a party of the Contract
DEONTIC_EXPRESSION_NOT_FOUND	A deontic expression is missing or incomplete.
CONDITION_NOT_SATISFIED	The conditions for a deontic expression to hold are not met.
UNKNOWN_ERROR	An unknown error occurred.

The possible values of the `ActionType` are given in Table 11.

Table 11 — Possible values of the `ActionType`

Action Code	Definition
AUDIOVISUAL_DISTRIBUTE	The Action of selling, renting and lending audiovisual material
MAKE_ADAPTATION	The Action of making an Adaptation (e.g., creating derivative works etc.)
MAKE_EXCERPT	The Action of making an Excerpt (e.g., to recombine (audio tracks or channels from a recording) to produce a new or modified audio recording.)
REMIX	The Action of recombining (audio tracks or channels from a recording) to produce a new or modified audio recording.
MAKE_INSTANCE	The Action of making an Instance from a Manifestation.
DUPLICATE	The Action of reproducing content in any manner or form (i.e., reproduction covers all methods of reproduction for instance drawing, lithography, offset and other printing processes, photocopying, recording).
PRODUCE	The Action of making Products
PUBLIC_COMMUNICATION	The Action of publicly displaying/performing, e.g., live performance, radio, television, internet streaming, multicast of Instances and Manifestations, and download
COMMUNICATION_TO_THE_PUBLIC	The Action of making a public exhibition (performing, showing and playing the work in public), being the provider and the public in the same place, i.e., Theater and Cinema exhibition. To present or execute a work in a place open to the public or at a place where a substantial number of persons outside of a normal circle of a family are gathered a indoor or outdoor environment to which members of the public are given access either for an admission fee or free of charge.

FIXATE	The Action of fixing the oral works and/or performances on a material support.
PUBLIC_PERFORMANCE	The Action of making a public performance, being a performance considered "public" when the work is presented or executed in a place open to the public or at a place where a substantial number of persons outside of a normal circle of a family and its social acquaintances are gathered.
BROADCAST	The Action of sending out or communicating, especially by radio or television.

5.7.1.5 Extension to SID

For the representation of Service Instance Declarations of *Check With Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="CheckWithContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.7.2 Check With License

5.7.2.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Check With License elementary service.

Check With License allows Users to verify facts against the content of a License. This Elementary Service may trigger an authorization as described in Clause 5 in ISO/IEC 21000-5 but the SP must not change its state by any means as a result of a Check With ES (e.g., even if there is a clause where the consumption of a right is limited to a certain number of times, the account will not be decreased).

5.7.2.2 Interfaces and protocol specification

The Check With License Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `CheckWithLicenseRequest` message to verify a given License.	
2.		The SP sends a `CheckWithLicenseResponse` with the result of the check.

5.7.2.3 Syntax of protocol data format

```
<element name="CheckWithLicenseRequest"
type="mpegm:CheckWithLicenseRequestType"/>
  <complexType name="CheckWithLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
          <element name="Principal" type="rel-r:Principal" minOccurs="0"/>
          <element name="Right" type="rel-r:Right" maxOccurs="unbounded"/>
          <element name="Resource" type="rel-r:Resource" maxOccurs="unbounded"/>
          <element name="Condition" type="rel-r:Condition" maxOccurs="unbounded"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name="CheckWithLicenseResponse"
type="mpegm:CheckWithLicenseResponseType"/>
  <complexType name="CheckWithLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
          <element name="CheckWithLicenseSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="CheckWithLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.7.2.4 Semantics of protocol data format

Semantics of the `CheckWithLicenseRequest`:

Name	*Definition*
`CheckWithLicenseRequestType`	Message sent to start the Check With service.
`LicenseEntity`	A MPEG-21 REL license against which to perform the check. The license may be included either directly, by reference, or by license identifier as specified in 5.2.3.12. NOTE If a licensing service, which is known by the SP, stores the licenses for performing the license checking, then the `LicenseEntity` can be omitted.
`Principal`	A MPEG-21 REL principal desiring to exercise a right.
`Right`	An MPEG-21 REL right to be exercised.

`Resource`	An MPEG-21 REL resource over which the right is exercised.
`Condition`	An MPEG-21 REL condition stating the conditions that should be considered for the evaluation of the request.

Semantics of the `CheckWithLicenseResponse`:

Name	*Definition*
`CheckWithLicenseResponseType`	Message sent as a result of the Check With service invocation.
`CheckWithLicenseSuccess`	Response in case of success.
`CheckWithLicenseFailure`	Response in case of failure.

5.7.2.5 Extension to SID

For the representation of Service Instance Declarations of *Check With Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="CheckWithLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.8 The Create Services

5.8.1 Create Content

5.8.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Create Content elementary service.

This Elementary Service enables a User to create a Digital Item. In order to achieve that, several data should be provided.

5.8.1.2 Interfaces and protocol specification

The Create Content Protocol is as follows:

Steps	Client	Service Provider
1.	The Client sends a `CreateContentRequest` message. This message may contain a set of elements to create a DI (resources, licenses, metadata, ERR). At least a resource should be provided.	

Steps	Client	Service Provider
2.		The SP sends back a `CreateContentResponse` message containing a DI or a reference to it.

5.8.1.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Create Content                                  -->
<!-- ################################################ -->

  <!-- Definition of CreateContentRequest -->
  <element name="CreateContentRequest" type="mpegm:CreateContentRequestType"/>
  <complexType name="CreateContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="Metadata" type="didmodel:DescriptorType" minOccurs="0"/>
          <element ref="erl:ERR" minOccurs="0"/>
          <element name="Asset" type="mpegm:AssetType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AssetType">
    <sequence>
      <choice minOccurs="0">
        <element name="IPMPToolInfo" type="mpegm:IPMPToolInfoType"/>
        <element ref="ipmpinfo:IPMPGeneralInfoDescriptor"/>
      </choice>
      <element name="Metadata" type="didmodel:DescriptorType" minOccurs="0"/>
      <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
      <element name="Resource" type="didmodel:ResourceType"/>
   <element ref="erl:ERR" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="IPMPToolInfoType">
    <sequence>
      <element ref="ipmpmsg:IPMPToolID"/>
      <element ref="ipmpmsg:ParametricDescription" minOccurs="0"/>
    </sequence>
  </complexType>

  <!-- Definition of CreateContentResponse -->
  <element name="CreateContentResponse" type="mpegm:CreateContentResponseType"/>
  <complexType name="CreateContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="CreateContentSuccess"
type="mpegm:CreateContentSuccessType"/>
            <element name="CreateContentFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="CreateContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.8.1.4 Semantics of protocol data format

Semantics of the `CreateContentRequest`:

Name	*Definition*
`CreateContentRequest`	The message for requesting to create Content.
`CreateContentRequestType`	Top-level type for `CreateContentRequest`. `CreateContentRequestType` extends `ProtocolRequestType`.
`Metadata`	Metadata describing the Content. The Metadata element has a `didmodel:Statement` child element that can contain various metadata schemas. EXAMPLE In a music album, `Metadata` would describe the overall album while `Asset/Metadata` would describe the individual tracks.
`erl:ERR`	The Event Report Request for the Content.
`Asset`	A list of one or more items that should generate the content.

Semantics of the `AssetType`:

Name	*Definition*
`IPMPToolInfo`	DRM Tool, which could be provided by Request Device.
`ipmpinfo:IPMPGeneralInfoDescription`	Container for a list of required IPMP tools and a collection of licenses.
`Metadata`	Metadata describing this individual asset of the Content. The Metadata element has a `didmodel:Statement` child element that can contain various metadata schemas.
`LicenseEntity`	REL license for the content, preferably included directly.
`Resource`	URL referring to a resource or the resource itself.
`erl:ERR`	The list of Event Report Requests provided for this

individual asset of the Content.

Semantics of the `IPMPToolInfoType`:

Name	*Definition*
`ipmpmsg:IPMPToolID`	Identifier of the DRM tool.
`ipmpmsg:ParametricDescription`	Parametric description of the IPMP tool, as defined in ISO/IEC 23001-3.

Semantics of the `CreateContentResponse`:

Name	*Definition*
`CreateContentResponse`	The response contains the result of the Create Content Service. In the first case the Digital Item created is provided.
`CreateContentResponseType`	Top-level type for `CreateContentResponse`. `CreateContentResponseType` extends `ProtocolRequestType`.
`CreateContentSuccess`	Response in case of success.
`CreateContentFailure`	Response in case of failure.

Semantics of the `CreateContentSuccessType`:

Name	*Definition*
`CreateContentSuccessType`	Type of the response message part that is provided in case of success. If a `SuccessCode` element is present, it should be set to "CREATED".
`ContentEntity`	A Digital Item generated according to the request.

5.8.1.5 Extension to SID

For the representation of Service Instance Declarations of *Create Content*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="CreateContentSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.8.2 Create Contract

5.8.2.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Create Contract elementary service.

This Elementary Service enables a User to create a Contract. The Create Contract Protocol allows creating simple contracts, composed of unrelated Licenses, Text and other Permissions.

5.8.2.2 Interfaces and protocol specification

The Create Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The Client sends a `CreateContractRequest` message. This message may contain a set of elements to create a Contract.	
2.		The SP sends back a `CreateContractResponse` message containing a Contract.

5.8.2.3 Syntax of protocol data format

```
  <!-- ############################################ -->
  <!--  Create Contract                             -->
  <!-- ############################################ -->
  <!-- Definition of CreateContractRequest -->
  <element name="CreateContractRequest" type="mpegm:CreateContractRequestType"/>
<complexType name="CreateContractRequestType">    <complexContent>
<extension base="mpegmb:ProtocolRequestType">        <sequence>
          <element name="Contract" type="cel:ContractType" minOccurs="0"/>
          <element name="TextClause" type="cel:TextClauseType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="DeonticStructuredClause"
type="cel:DeonticStructuredClauseType" minOccurs="0" maxOccurs="unbounded"/>
          <element name="EncryptedClause" type="cel:EncryptedClauseType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>      </extension>    </complexContent>  </complexType>

  <!-- Definition of CreateContractResponse -->
  <element name="CreateContractResponse"
type="mpegm:CreateContractResponseType"/>  <complexType
name="CreateContractResponseType">    <complexContent>      <extension
base="mpegmb:ProtocolResponseType">        <choice>
          <element name="CreateContractSuccess"
type="mpegm:CreateConractSuccessType"/>
          <element name="CreateContractFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>      </extension>    </complexContent>  </complexType>

  <complexType name="CreateContractSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
```

```
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.8.2.4 Semantics of protocol data format

Semantics of the `CreateContractRequest`:

Name	*Definition*
`CreateContractRequest`	The message for creating a request for Create Contract.
`CreateContractRequestType`	Top-level type for `CreateContractRequest`. `CreateLicenseRequestType` extends `ProtocolRequestType`.
`Contract`	A complete contract element, of type `cel:ContractType` if the contract is already fully created.
`TextClause`	Contractual clause of type `cel:TextClauseType`. Each one will be added into the MPEG-21 contract structure.
`DeonticStructuredClause`	Contractual clause of type `cel:DeonticStructuredClauseType`. Each one will be added into the MPEG-21 contract structure.
`EncryptedClause`	Encrypted contractual clause of type `cel:EncryptedClauseType`. Each one will be added into the MPEG-21 contract structure.

Semantics of the `CreateContractResponse`:

Name	*Definition*
`CreateContractResponse`	The response message for creating a contract, which contains either a newly created Contract or a failure message.
`CreateContractResponseType`	Top-level type for `CreateContractResponse`. `CreateContractResponseType` extends `ProtocolRequestType`.
`CreateContractSuccess`	Response in case of success.
`CreateContractFailure`	Response in case of failure.

Semantics of the `CreateContractSuccessType`:

Name	*Definition*
`CreateContractSuccessType`	Type of the response message part that is provided in case of success. If a `SuccessCode` element is present, it should be set to "CREATED".
`ContractEntity`	A contract generated according to the request.

5.8.2.5 Extension to SID

For the representation of Service Instance Declarations of *Create Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="CreateContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.8.3 Create License

5.8.3.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Create License elementary service.

This Elementary Service enables a User/Actor to create a License. User may provide a minimum set of input data to create a simple license, but also all data needed to create a complex element.

Since a license can be created after a transaction related to the purchase of a digital item, the SP may request information from a User, whether the transaction had been performed successfully, before answering to a request for creating a license.

5.8.3.2 Interfaces and protocol specification

The Create License Protocol when no transaction is involved is as follows:

Steps	Client	Service Provider
1.	The client sends a `CreateLicenseRequest` message. The message needs at least a `Resource` and a `Right`. Other information may be: `Principal`, `Issuer`, `ForAll`...	

Steps	Client	Service Provider
2.		The SP sends a `CreateLicenseResponse`, if the operation ends successfully, it will be sent a License reference.

The Create License Protocol when a transaction is involved is as follows:

Steps	Buyer (Client)	Service Provider	Seller
1.	The buyer has successfully purchased an item and he/she sends a `CreateLicenseRequest` message. The message needs at least a `Resource` and a `Right`. Other information may be: `Principal`, `Issuer`, `ForAll`...		
2.		SP may require confirmation of a successful transaction regarding the requested license to the user or to another User (the Seller) with a `TransactionConfirmationRequest`.	
3.			The seller, since the transaction was successful, acknowledges with a `TransactionConfirmationResponse` message containing the result of the transaction.
4.		The SP sends a `CreateLicenseResponse`, if the operation ends successfully, the message contains a License reference.	

5.8.3.3 Syntax of protocol data format

```
<!-- ############################################################ -->
<!--  Create License                                              -->
<!-- ############################################################ -->

<element name="CreateLicenseRequest" type="mpegm:CreateLicenseRequestType"/>
  <complexType name="CreateLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <choice>
          <element name="CustomLicense" type="mpegm:CustomLicenseType"/>
          <element name="TemplateLicense" type="mpegm:TemplateLicenseType"/>
```

```
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="CustomLicenseType">
    <sequence>
      <element ref="mpegmb:LicenseIdentifier"/>
      <choice>
        <element name="Principal" type="rel-r:Principal" minOccurs="0"/>
        <element ref="rel-r:forAll" minOccurs="0"/>
      </choice>
      <element name="RightGroup" type="mpegm:RightGroupType"
maxOccurs="unbounded"/>
      <element ref="rel-r:resource"/>
      <element ref="rel-r:issuer" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="RightGroupType">
    <sequence>
      <element ref="rel-r:right"/>
      <element name="Condition" type="rel-r:Condition" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="TemplateLicenseType">
    <sequence>
      <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
      <element name="Principal" type="rel-r:Principal" minOccurs="0"/>
    </sequence>
  </complexType>

<!-- Definition of CreateLicenseResponse -->
<element name="CreateLicenseResponse" type="mpegm:CreateLicenseResponseType"/>
  <complexType name="CreateLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="CreateLicenseSuccess"
type="mpegm:CreateLicenseSuccessType"/>
            <element name="CreateLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of CreateLicenseSuccessType -->
  <complexType name="CreateLicenseSuccessType">    <complexContent>
<extension base="mpegmb:ProtocolSuccessType">        <sequence>          <element
name="LicenseEntity" type="mpegmb:LicenseEntityType"/>        </sequence>
</extension>    </complexContent>  </complexType>
<element name="TransactionConfirmationRequest"
type="mpegm:TransactionConfirmationRequestType"/>
  <complexType name="TransactionConfirmationRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
```

```
          <element name="TargetLicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<element name="TransactionConfirmationResponse"
type="mpegm:TransactionConfirmationResponseType"/>
  <complexType name="TransactionConfirmationResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="TransactionConfirmationSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="TransactionConfirmationFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

5.8.3.4 Semantics of protocol data format

Semantics of the `CreateLicenseRequest`:

Name	*Definition*
`CreateLicenseRequest`	The message for creating a request for CreateLicense.
`CreateLicenseRequestType`	Top-level type for `CreateLicenseRequest`. `CreateLicenseRequestType` extends `mpegm:ProtocolRequestType`.
`CustomLicense`	A `CustomLicense` is a `CustomLicenseType` where all the fields of the license need to be specified.
`TemplateLicense`	A `TemplateLicense` is a `TemplateLicenseType` where all the fields of the license are already specified except for the Principal to whom the license is granted (i.e., the licensee).

Semantics of the `CustomLicenseType`:

Name	*Definition*
`Principal`	The licensee to whom the license is being granted.
`forAll`	Provides a set of variables within the scope of the license. If the license contains multiple principals, these variables apply to all principals.
`RightGroup`	This element combines a right with the corresponding conditions.

Resource	The resource on which the issuer grants the rights to the principals specified above.
LicenseIdentifier	The identifier of the license.
Issuer	The issuer of the license.

Semantics of the RightGroupType:

Name	*Definition*
RightGroupType	This element combines a right with the corresponding conditions.
Right	The Right corresponding to a Grant of a REL license.
Condition	Condition corresponding to the right.

Semantics of the TemplateLicenseType:

Name	*Definition*
LicenseEntity	A license or a reference to it, having all the fields already present except for Principal.
Principal	Licensee to whom the license is being granted.

Semantics of the CreateLicenseResponse:

Name	*Definition*
CreateLicenseResponse	The response message for creating a license, which contains either a newly created license or a failure message.
CreateLicenseResponseType	Top-level type for CreateLicenseRequest. CreateLicenseRequestType extends ProtocolResponseType.
CreateLicenseSuccess	Response in case of success.
CreateLicenseFailure	Response in case of failure.

Semantics of the CreateLicenseSuccessType:

Name	*Definition*
CreateLicenseSuccessType	CreateLicenseSuccessType extends ProtocolSuccessType. If a SuccessCode element is present, it should be set to "CREATED".
LicenseEntity	A reference to the created license or the created license itself.

Semantics of the `TransactionConfirmationRequest`:

Name	*Definition*
`TransactionConfirmationRequest`	The message for creating a `TransactionConfirmationRequest`.
`TransactionConfirmationRequestType`	Top-level type for `TransactionConfirmationRequest`. `TransactionConfirmationRequestType` extends `ProtocolRequestType`.
`TargetLicenseEntity`	The template of the license that will be generated.
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22.

Semantics of the `TransactionConfirmationResponse`:

Name	*Definition*
`TransactionConfirmationResponse`	This message is sent in response to a `TransactionConfirmationRequest` message.
`TransactionConfirmationResponseType`	Top-level type for `TransactionConfirmationResponse`. `TransactionConfirmationResponseType` extends `ProtocolResponseType`.
`TransactionConfirmationSuccess`	The result of the transaction occurred for the content item purchased in case of success.
`TransactionConfirmationFailure`	The result of the transaction occurred for the content item purchased in case of failure.

5.8.3.5 Extension to SID

For the representation of Service Instance Declarations of *Create License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->
<complexType name="CreateLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.9 The Deliver Services

5.9.1 Deliver Content

5.9.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Deliver Content elementary service.

With the Deliver Content Protocol a client can request a Service Provider (SP) to initiate the delivery of content from a Sender to a Receiver according to specified terms and conditions. The client also specifies in the request, how the notification about the delivery should be performed.

This document does not specify which entities perform the protocol communication. In particular, the entity issuing a `DeliverContentRequest` message is not necessarily involved in the actual delivery.

5.9.1.2 Interfaces and protocol specification

The Deliver Content Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `DeliverContentRequest` message to the SP.	
2.		Upon receipt, the SP evaluates, whether the content delivery can be performed. If the delivery can be performed, the SP initiates the delivery. If the client has requested an immediate response message, the SP continues with step 3. Otherwise, if the client has not requested an immediate response message, the SP continues with step 4.
3.		(optional) The SP sends a `DeliverContentResponse` message to the client, indicating whether the delivery will be performed. If the delivery cannot be performed, the protocol ends here.
4.		After the delivery has finished, the SP sends a `DeliverContentCompletion` message to the client, indicating whether the delivery has been performed successfully.

The `TransactionIdentifier` remains the same over the entire protocol run.

In addition to the specified protocol, the client may also apply the session control messages defined in 5.4.1 to pause/resume or stop the processing.

The client may also apply the session status polling message defined in 5.4.2 to repeatedly poll for the status of the delivery.

5.9.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Deliver Content                                 -->
  <!-- ################################################ -->


  <!-- Definition of DeliverContentRequest -->
  <element name="DeliverContentRequest" type="mpegm:DeliverContentRequestType"/>
  <complexType name="DeliverContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
          <choice>
            <element name="SenderUser" type="mpegmb:UserEntityType"/>
            <element name="SenderDevice" type="mpegmb:DeviceEntityType"/>
          </choice>
          <choice>
            <element name="RecipientUser" type="mpegmb:UserEntityType"/>
            <element name="RecipientDevice" type="mpegmb:DeviceEntityType"/>
          </choice>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
        </sequence>
        <attribute name="immediateResponse" type="boolean" use="optional"
default="true"/>
      </extension>
    </complexContent>
  </complexType>


  <!-- Definition of DeliverContentResponse -->
  <element name="DeliverContentResponse"
type="mpegm:DeliverContentResponseType"/>
  <complexType name="DeliverContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="DeliverContentSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="DeliverContentFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>


  <!-- Definition of DeliverContentCompletion -->
  <element name="DeliverContentCompletion"
type="mpegm:DeliverContentCompletionType"/>
  <complexType name="DeliverContentCompletionType">
```

```
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="DeliverContentCompletionSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="DeliverContentCompletionFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

5.9.1.4 Semantics of protocol data format

Semantics of the `DeliverContentRequest`:

Name	*Definition*
`DeliverContentRequest`	Protocol message sent from the client to the SP to request that certain content is delivered from the Sender to the Receiver under provided terms and conditions.
`DeliverContentRequestType`	Top-level type for `DeliverContentRequest`. `DeliverContentRequestType` extends `ProtocolRequestType`.
`ContentEntity`	The content to be delivered.
`SenderUser`	The sending User, typically specified via reference.
`SenderDevice`	The sending Device, typically specified via reference.
`RecipientUser`	The receiving User, typically specified via reference. If no `Recipient` is specified, the Content will be sent to the client.
`RecipientDevice`	The receiving Device, typically specified via reference. If no Recipient is specified, the Content will be sent to the client.
`LicenseEntity`	An optional license that may be associated with the content.
`immediateResponse`	Indicates whether the SP has to respond with a `DeliverContentResponse` message. If set to "false", the SP will only send the `DeliverContentCompletion` message after the delivery has finished.

Optionally, application-specific information such as Service start time or network properties for delivery can be provided in the `Entry` and `ApplicationSpecificData` elements of the request message.

Semantics of the DeliverContentResponse:

Name	*Definition*
DeliverContentResponse	Protocol message sent from the SP to the client to indicate whether the delivery will be performed. NOTE DeliverContentResponse only indicates, whether the SP will attempt to carry out the delivery process.
DeliverContentResponseType	Top-level type for DeliverContentResponse. DeliverContentResponseType extends ProtocolResponseType.
DeliverContentSuccess	Response in case of success. If a SuccessCode element is present, it should be set to "ACCEPTED".
DeliverContentFailure	Response in case of failure.

Semantics of the DeliverContentCompletion:

Name	*Definition*
DeliverContentCompletion	Protocol message sent from the SP to the client after finishing the delivery of the specified content.
DeliverContentCompletionType	Top-level type for DeliverContentCompletion. DeliverContentCompletionType extends ProtocolResponseType.
DeliverContentCompletionSuccess	Response in case of success.
DeliverContentCompletionFailure	Response in case of failure.

5.9.1.5 Extension to SID

For the representation of Service Instance Declarations of *Deliver Content*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->
<complexType name="DeliverContentSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.9.1.6 Example

EXAMPLE This example shows the run of the Deliver Content Protocol. The client sends a `DeliverContentRequest` message to the SP, requesting for a video to be delivered between two Users.

```
<DeliverContentRequest immediateResponse="true">
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!—Signature of the entire message goes here... -->
  </dsig:Signature>
  <ContentEntity>
    <didl:DIDL>
      <didl:Item>
        <didl:Descriptor>
          <didl:Statement mimeType="text/xml">
            <dii:Identifier>urn:mpegRA:mpeg21:dii:cid:1702.F109%2F0000011
</dii:Identifier>
          </didl:Statement>
        </didl:Descriptor>
      </didl:Item>
    </didl:DIDL>
  </ContentEntity>
  <Sender>
    <mpegmb:UserRef>http://example.com/user/john.doe</mpegmb:UserRef>
  </Sender
  <Recipient>
    <mpegmb:UserRef>http://example.com/user/max.mustermann</mpegmb:UserRef>
  </Recipient>
  <LicenseEntity>
    <!-- License goes here... -->
  </LicenseEntity>
</DeliverContentRequest>
```

The content to be delivered is identified as by the DII "urn:mpegRA:mpeg21:dii:cid:1702.F109%2F0000011". The Sender is identified as the User "John Doe". The Receiver is identified as the User "Max Mustermann". The `rel-r:license` element provides the license for the content. The `immediateResponse` attribute indicates that the SP must respond with a `DeliverContentResponse` message. The additional `dsig:Signature` element contains an XML Signature of the `DeliverContentRequest` message.

The SP checks whether the content can be delivered to the Receiver and responds with an affirmative `DeliverContentResponse` message to the client.

```
<DeliverContentResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
  <DeliverContentSuccess/>
</DeliverContentResponse>
```

The client may than poll for the status of the delivery. The `TransactionIdentifier` element of the `DeliverContentResponse` message is referenced through the `TransactionRef` element in the `StatusRequest` message.

```
<StatusRequest>
  <mpegmb:TransactionIdentifier>43</mpegmb:TransactionIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
  <TransactionRef>42</TransactionRef>
</StatusRequest>
```

The SP has already begun the content delivery and responds to the `StatusRequest` message with a `StatusResponse` message with the Progress`Status` set to "CURRENTLY_IN_PROGRESS".

```
<StatusResponse completion="0.2">
  <mpegmb:TransactionIdentifier>43</mpegmb:TransactionIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
  <StatusSuccess>
    <ProgressStatus>CURRENTLY_IN_PROGRESS</ProgressStatus>
    <DisplayString>2048 of 10240 KB sent</DisplayString>
  </StatusSuccess>
</StatusResponse>
```

After the successful delivery of the content, the SP sends a `DeliverContentCompletion` message to the client.

```
<DeliverContentCompletion>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
  <DeliverContentCompletionSuccess/>
</StatusResponse>
```


5.9.2 Deliver Contract

5.9.2.1 Introduction

The Deliver Contract Service allows Users to transfer a Contract between Users of a multimedia content value chain.

The "Deliver Contract" operation takes place between two peers negotiating a contract among other situations, and it simply consists of the delivery of a contract. Given that Contracts are not large documents, there is no need for the delivery mechanisms described in 5.9.1 for Content.

5.9.2.2 Interfaces and protocol specification

The Deliver Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The Client sends a `DeliverContractRequest` message.	

Steps	Client	Service Provider
2.		Upon receipt, the SP evaluates, whether the contract delivery can be performed. If the delivery can be performed, the SP initiates the contract delivery.
3.		After the delivery has finished, the SP sends a `DeliverContractResponse` message to the client, indicating whether the delivery has been performed successfully.

5.9.2.3 Syntax of protocol data format

This is the message sent to request the operation Deliver, being possible to specify origin and destination of the delivery. It includes simply one contract, plus the sender and the recipient.

```
 <element name="DeliverContractRequest" type="mpegm:DeliverContractRequestType"/>
  <complexType name="DeliverContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType" />
          <choice>
            <element name="SenderUser" type="mpegmb:UserEntityType"/>
            <element name="SenderDevice" type="mpegmb:DeviceEntityType"/>
          </choice>
          <choice>
            <element name="RecipientUser" type="mpegmb:UserEntityType"/>
            <element name="RecipientDevice" type="mpegmb:DeviceEntityType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of DeliverContractResponse -->
  <element name="DeliverContractResponse"
type="mpegm:DeliverContractResponseType"/>
  <complexType name="DeliverContractResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="DeliverContractSuccess"
type="mpegmb:ProtocolSuccessType"/>
            <element name="DeliverContractFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.9.2.4 Semantics of protocol data format

Semantics of the `DeliverContractRequest`:

Name	*Definition*
`DeliverContractRequest`	Protocol message sent from the client to the SP to request that certain contract is delivered from the Sender to the Receiver under provided terms and conditions.
`DeliverContractRequestType`	Top-level type for `DeliverContractRequest`. `DeliverContractRequestType` extends `ProtocolRequestType`.
`ContractEntity`	A MPEG-21 CEL contract as described in 5.2.3.10.
`SenderUser`	The sending User, typically specified via reference.
`SenderDevice`	The sending Device, typically specified via reference.
`RecipientUser`	The receiving User, typically specified via reference. If no `Recipient` is specified, the Content will be sent to the client.
`RecipientDevice`	The receiving Device, typically specified via reference. If no Recipient is specified, the Content will be sent to the client.

Semantics of the `DeliverContractResponse`:

Name	*Definition*
`DeliverContractResponse`	Protocol message sent from the SP to the client to indicate whether the delivery has been performed. NOTE In contrast to Deliver Content, the `DeliverContractResponse` is sent after the delivery of the contract.
`DeliverContractResponseType`	Top-level type for `DeliverContractResponse`. `DeliverContractResponseType` extends `ProtocolResponseType`.
`DeliverContractSuccess`	Response in case of success.
`DeliverContractFailure`	Response in case of failure.

5.9.2.5 Extension to SID

For the representation of Service Instance Declarations of *Deliver Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="DeliverContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
```

```
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.10 The Describe Services

5.10.1 Describe Entity

5.10.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the abstract Describe Entity Protocols.

The Describe Entity Protocols comprise two generic *getter* and *setter* protocols that enable access to either entire XML documents or parts thereof. Access to fragments of a description may reduce network traffic for large descriptions. The protocols are independent of the data format used in these XML documents to represent metadata.

The Set Describe Entity Protocol is specified in 5.10.1.2 and the Get Describe Entity Protocol is specified in 5.10.1.3.

5.10.1.2 Set Describe Entity Protocol

5.10.1.2.1 Interfaces and protocol specification

The Set Describe Entity Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a message of type `SetDescribeEntityRequestType` to the server.	
2.		Upon receipt, the SP processes the `AccessUnit` as specified in ISO/IEC 15938-1, i.e., performs the necessary updates to the description.
3.		The SP sends a message of type `SetDescribeEntityResponseType` to the client, signaling whether the operation was carried out successfully.

The request contains a Fragment Update Unit (FUU) as defined in ISO/IEC 15938-1. This FUU conveys either the entire XML document of the description of the Entity, or individual parts thereof that shall be updated.

Schema validity of description

The current description tree resulting after composition must be schema valid after all Fragment Update Units have been processed, but intermediate results need not be schema valid.

5.10.1.2.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  SetDescribeEntityProtocol                       -->
  <!-- ################################################ -->

  <!-- Definition of SetDescribeEntityRequestType -->
  <complexType name="SetDescribeEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="EntityIdentifier">
            <simpleType>
              <union memberTypes="string anyURI"/>
            </simpleType>
          </element>
          <element name="AccessUnit" type="mpeg7:AccessUnitType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of SetDescribeEntityResponseType -->
  <complexType name="SetDescribeEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="SetDescribeEntitySuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="SetDescribeEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

5.10.1.2.3 Semantics of protocol data format

Semantics of the `SetDescribeEntityRequestType`:

Name	*Definition*
`SetDescribeEntityRequestType`	Top-level type for Set Describe Entity Protocol request messages. `SetDescribeEntityRequestType` extends `ProtocolRequestType`.
`EntityIdentifier`	Identification of the Entity to be described (i.e.., Content Identifier, Device Identifier, SID Identifier, or User Identifier).
`AccessUnit`	Conveys the description of the Entity identified by the `EntityIdentifier` or parts of the description. The `AccessUnit` element contains one or more `mpeg7:FragmentUpdateUnit` elements.

Semantics of the `SetDescribeEntityResponseType`:

Name	*Definition*
`SetDescribeEntityResponseType`	Top-level type for Set Describe Entity Protocol response messages. `SetDescribeEntityResponseType` extends `ProtocolResponseType`.
`SetDescribeEntitySuccess`	Response in case of success.
`SetDescribeEntityFailure`	Response in case of failure.

5.10.1.2.4 Additional validation rules

5.10.1.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.1.2.4.2 The metadata schema of the description conveyed in the `AccessUnit` element of the request must conform to one of the metadata schemas specified by the `sid:SupportedMetadataSchema` elements of the SID.

5.10.1.3 Get Describe Entity Protocol

5.10.1.3.1 Interfaces and protocol specification

5.10.1.3.1.1 General

The Get Describe Entity Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a message of type `GetDescribeEntityRequestType` to the server.	
2.		Upon receipt, the SP determines, whether it has the requested description.
3.		If the request contains a `fru:FRU` element, the SP creates an appropriate FRU Response (i.e., an `AccessUnit` element containing the appropriate Fragment Update Units).
4.		Otherwise if the entire description was requested (i.e., the request contains no `fru:FRU` element) then the server creates an FRU response (i.e., `AccessUnit`) containing the entire description.

Steps	Client	Service Provider
5.		The SP sends a `GetDescribeEntityResponse` message to the client. If the request was processed successfully, the message contains the FRU Response.

The Get Describe Entity Protocol uses Fragment Request Units (FRU) as defined in ISO/IEC 23001-2 to enable the retrieval of individual parts of the description of an Entity (e.g., the title or creation date of a Content). The usage of FRU presumes that the client knows the structure of the metadata representation before the request. The FRU contains a W3C XPATH1 query into the metadata XML document.

The response from the SP contains a Fragment Update Unit (FUU) as defined in ISO/IEC 15938-1. This FUU conveys either the entire XML document of the description of an Entity, or individual parts as requested through the FRU.

5.10.1.3.1.2 Processing of FRU

The `fru:Src` element of the `fru:FRU` element must be omitted because the description to be requested is already identified through the `ItemIdentifier` element. If the `navMode` attribute of a `fru:Query` element is present, it must be set to "false".

NOTE The use of the `fru:Query` elements prohibits any `fru:XMLPull` elements.

The XPATH1 query of a `fru:Query` element is an XPATH1 AbsoluteLocationPath. The XPATH1 query may use any of the namespace prefixes defined within the scope of the entire Get Describe Entity Protocol request message of type `GetDescribeEntityRequestType`. The server must register all namespace and corresponding namespace prefixes defined within the scope of the Get Describe Entity Protocol request message at the XPATH1 engine before evaluating the XPATH1 query.

The `fru:FRU` element should be treated as if it was in the *Src* operation mode "closed", i.e., the current context information is not needed to be maintained at the server side.

If the entire description of an Entity is requested, the `fru:FRU` element should be omitted.

5.10.1.3.1.3 Conformance to ISO/IEC 23001-2

Total conformance to ISO/IEC 23001-2 with respect to the `fru:Src` element can be re-established by the server by inserting an additional `fru:FRU` element with only a `fru:Src` element into the FRU Request Processor. An example of such an additional `fru:FRU` element is shown below. The content of the `fru:Src` element ("uri/to/description") is replaced by the appropriate URI to the description which can be inferred from the `EntityIdentifier` element. This additional `fru:FRU` element is inserted into the FRU Request Processor prior to the `fru:FRU` element from the Get Describe Entity Protocol request message.

```
<fru:FRU>
  <fru:Src mode="closed">uri/to/description</fru:Src>
</fru:FRU>
```

5.10.1.3.1.4 Processing of FUU

The `AccessUnit` contains one or more `mpeg7:FragmentUpdateUnit` elements. If the entire description of an Entity was requested, exactly one `mpeg7:FragmenUpdateUnit` element should be present.

The mpeg7:FUContext element of an mpeg7:FragmentUpdateUnit element represents the navigation path of the context node. The content of the mpeg7:FUContext element is not needed to be the same as the XPATH1 query from the Get Describe Entity Protocol request message.

NOTE The XPATH1 query from the Get Describe Entity Protocol request message defines, which nodes are requested. The mpeg7:FUContext element specifies the navigation path of a matching node.

Similar to the XPATH1 query of the Get Describe Entity Protocol request message, the navigation path of the mpeg7:FUContext may use any of the namespace prefixes defined within the scope of the entire Get Describe Protocol response message of type GetDescribeEntityResponseType.

If the entire description of an Entity was requested, the mpeg7:FUContext element should be set to "/" to represent the document root.

The mpeg7:FUPayload element contains the actual description or the requested parts of the description. Since the mpeg7:FragmentUpdatePayloadType complex type does not allow character data as content, XPATH1 queries from Get Describe Entity Protocol request messages must be handled as follows: If a result of the XPATH1 query of the Get Describe Entity Protocol request message yields a text node (i.e., character data), then the mpeg7:FUPayload element must contain the parent node of that text node. In this case, the mpeg7:FUContext element must be adjusted accordingly (e.g., by appending "/.." to the value to indicate the parent node).

5.10.1.3.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  GetDescribeEntityProtocol                       -->
  <!-- ################################################ -->

  <!-- Definition of GetDescribeEntityRequestType -->
  <complexType name="GetDescribeEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="EntityIdentifier">
            <simpleType>
              <union memberTypes="string anyURI"/>
            </simpleType>
          </element>
          <element name="SupportedMetadataSchema"
type="mpegmb:SupportedMetadataSchemaType" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="fru:FRU" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of GetDescribeEntityResponseType -->
  <complexType name="GetDescribeEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="GetDescribeEntitySuccess"
type="mpegm:GetDescribeEntitySuccessType"/>
          <element name="GetDescribeEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

```
<complexType name="GetDescribeEntitySuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolSuccessType">
      <sequence>
        <element name="AccessUnit" type="mpeg7:AccessUnitType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.10.1.3.3 Semantics of protocol data format

Semantics of the `GetDescribeEntityRequestType`:

Name	*Definition*
`GetDescribeEntityRequestType`	Top-level type for Get Describe Entity Protocol request messages. `GetDescribeEntityRequestType` extends `ProtocolRequestType`.
`EntityIdentifier`	Identification of the Entity about which to get the description (i.e., Content Identifier, Device Identifier, SID Identifier, or User Identifier).
`SupportedMetadataSchema`	Indicates the `xsd:targetNamespace` of the metadata schemas, in which the returned description must be represented. NOTE If the `strictMetadataSupport` attribute of the SID (see C.3) is set to "true", the `SupportedMetadataSchema` element is redundant and shall not be used. If the `SupportedMetadataSchema` is omitted, the SP may return the description in any metadata schema listed by the `sid:SupportedMetadataSchema` elements in the SID (preferably the one with the highest `pref` attribute).
`fru:FRU`	Conveys one or more XPATH1 queries to request specific parts of the description.

Semantics of the `GetDescribeEntityResponseType`:

Name	*Definition*
`GetDescribeEntityResponseType`	Top-level type for Get Describe Protocol response messages. `GetDescribeEntityResponseType` extends `ProtocolResponseType`.
`GetDescribeEntitySuccess`	Response in case of success.
`GetDescribeEntityFailure`	Response in case of failure.

Semantics of the `GetDescribeEntitySuccessType`:

Name	*Definition*
`GetDescribeEntitySuccessType`	Type of the response message part that is provided in case of success.
`AccessUnit`	Conveys the description (or parts thereof) of the Entity requested through a preceding message of type `GetDescribeEntityRequestType`. The `AccessUnit` element contains one or more `mpeg7:FragmentUpdateUnit` elements.

5.10.1.3.4 Additional validation rules

5.10.1.3.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.1.3.4.2 If present, the `SupportedMetadataSchema` elements must be a subset of the `sid:SupportedMetadataSchema` elements in the SID of the Service Instance. However, the client sets its own preferences through the `pref` attribute.

5.10.1.3.4.3 The metadata schema of the description conveyed in the `AccessUnit` element of the response must conform to one of the metadata schemas specified by the `SupportedMetadataSchema` elements of the request message.

5.10.1.4 Extension to SID

For the representation of Service Instance Declarations of *Describe Entity*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="DescribeEntitySIDType" abstract="true">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="SupportedMetadataSchema"
type="mpegmb:SupportedMetadataSchemaType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:DescribeEntitySIDType`:

Name	*Definition*
`sid:DescribeEntitySIDType`	Top-level type for SIDs of the *Describe Entity* abstract ES. `sid:DescribeEntitySIDType` extends `sid:ServiceInstanceDeclarationType`.

`sid:SupportedMetadataSchema`	Lists metadata schemas in which the Entity description can be provided. NOTE If the `strictMetadataSupport` attribute of the SID (see C.3) is set to "true", the `SupportedMetadataSchema` element is redundant and shall not be used.

5.10.2 Describe Content

5.10.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Describe Content elementary service.

The Describe Content Protocols allow access to descriptions (or parts thereof) of Content entities. The protocols extend the Describe Entity Protocols defined in 5.10.1.

The Content to be described is identified through its Digital Item Identifier (DII).

Content metadata formats are specified by the SID (see Annex C).

5.10.2.2 Set Describe Content Protocol

5.10.2.2.1 Interfaces and protocol specification

The Set Describe Content Protocol specifies how to set the description (or parts thereof) of a Content entity. The protocol extends the Set Describe Entity Protocol defined in 5.10.1.2.1, from which it inherits the protocol specification.

5.10.2.2.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  SetDescribeContentProtocol                      -->
  <!-- ################################################ -->

  <!-- Definition of SetDescribeContentRequest -->
  <element name="SetDescribeContentRequest"
type="mpegm:SetDescribeEntityRequestType"/>

  <!-- Definition of SetDescribeContentResponse -->
  <element name="SetDescribeContentResponse"
type="mpegm:SetDescribeEntityResponseType"/>
```


5.10.2.2.3 Semantics of protocol data format

Semantics of the `SetDescribeContentRequest`:

Name	*Definition*
`SetDescribeContentRequest`	Protocol message sent from the client to the server in order to set the description (or parts thereof) of a Content entity.
`EntityIdentifier`	A Digital Item Identifier. (See semantics of the `SetDescribeEntityRequestType` in 5.10.1.2.3.)

Semantics of the `SetDescribeContentResponse`:

Name	*Definition*
`SetDescribeContentResponse`	Protocol message sent from the server to the client to indicate whether the protocol was carried out with success or otherwise.

5.10.2.2.4 Additional validation rules

5.10.2.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.2.2.4.2 The `EntityIdentifier` must be a Content Identifier represented as `dii:Identifier`.

5.10.2.2.5 Example

EXAMPLE This example shows the run of the Set Describe Content Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "false", allowing any representation format of Content metadata. For this example, the Content description is assumed to be represented in the TV-Anytime format. The client sends a `SetDescribeContentRequest` message to the server, containing a full description of the Content.

```
<SetDescribeContentRequest>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>
  <EntityIdentifier>crid://bbc.co.uk/272927586</EntityIdentifier>
  <AccessUnit>
    <mpeg7:FragmentUpdateUnit>
      <mpeg7:FUCommand>replaceNode</mpeg7:FUCommand>
      <mpeg7:FUContext>/.</mpeg7:FUContext>
      <mpeg7:FUPayload>
        <tva:TVAMain xml:lang="en" xmlns:tva="urn:tva:metadata:2010">
          <tva:ProgramDescription>
            <tva:ProgramInformationTable>
              <tva:ProgramInformation programId="crid://bbc.co.uk/272927586">
                <!-- Program Description goes here... -->
              </tva:ProgramInformation>
            </tva:ProgramInformationTable>
          </tva:ProgramDescription>
          <tva:ProgramLocationTable>
            <!-- Description of schedule goes here... -->
          </tva:ProgramLocationTable>
        </tva:TVAMain>
      </mpeg7:FUPayload>
    </mpeg7:FragmentUpdateUnit>
  </AccessUnit>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeContentRequest>
```

The server processes the request (i.e., sets the description for the Content) and responds with a `SetDescribeContentResponse` message. The `TransactionIdentifier` stays the same over an entire session.

```
<SetDescribeContentResponse>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>
  <SetDescribeEntitySuccess>
    <!-- The DisplayString is optional -->
    <DisplayString>OK</DisplayString>
  </SetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeContentResponse>
```

5.10.2.3 Get Describe Content Protocol

5.10.2.3.1 Interfaces and protocol specification

The Get Describe Content Protocol specifies how to get the description (or parts thereof) of a Content entity. The protocol extends the Get Describe Entity Protocol defined in 5.10.1.3.2, from which it inherits the protocol specification.

5.10.2.3.2 Syntax of protocol data format

```
  <!-- ############################################ -->
  <!--  GetDescribeContentProtocol                  -->
  <!-- ############################################ -->

  <!-- Definition of GetDescribeContentRequest -->
  <element name="GetDescribeContentRequest"
type="mpegm:GetDescribeEntityRequestType"/>

  <!-- Definition of GetDescribeContentResponse -->
  <element name="GetDescribeContentResponse"
type="mpegm:GetDescribeEntityResponseType"/>
```

5.10.2.3.3 Semantics of protocol data format

Semantics of the `GetDescribeContentRequest`:

Name	*Definition*
`GetDescribeContentRequest`	Protocol message sent from the client to the server in order to get the description (or parts thereof) of a Content entity. The `EntityIdentifier` element contains a Digital Item Identifier.
`EntityIdentifier`	A Digital Item Identifier. (See semantic of `GetDescribeEntityRequestType` in 5.10.1.3.3.)

Semantics of the `GetDescribeContentResponse`:

Name	*Definition*
`GetDescribeContentResponse`	Protocol message sent from the server to the client to convey the description (or parts thereof) of the Content entity requested through the preceding `GetDescribeContentRequest` message.

5.10.2.3.4 Additional validation rules

5.10.2.3.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.2.3.4.2 The `EntityIdentifier` must be a Content Identifier represented as `dii:Identifier`.

5.10.2.3.5 Examples

EXAMPLE 1 This example shows the run of the Get Describe Content Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "false", allowing any representation format of Content metadata. For this example, the Content description is assumed to be represented in the EBU Core format [6]. The client sends a `GetDescribeContentRequest` message to the server, requesting the full description of the Content.

```
<GetDescribeContentRequest>
  <mpegmb:TransactionIdentifier>41</mpegmb:TransactionIdentifier>
  <EntityIdentifier>crid://zak.ch/272927586</EntityIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeContentRequest>
```

The server processes the request (i.e. packs the description into an `mpeg7:FragmentUpdateUnit` element) and responds with a `GetDescribeContentResponse` message containing the `AccessUnit` element with a nested `mpeg7:FragementUpdateUnit` element. The `TransactionIdentifier` stays the same over an entire session.

NOTE Unlike the `SetDescribeContentRequest` message in 5.10.2.2.5, the `mpeg7:FUCommand` is set to "addNode" because the client is expected to have no prior description of the Content.

```
<GetDescribeContentResponse>
  <mpegmb:TransactionIdentifier>41</mpegmb:TransactionIdentifier>
  <GetDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>
        <mpeg7:FUContext>/.</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <ebucore:ebuCoreMain version="1.1" dateLastModified="2006-05-04"
documentId="documentId0">
            <ebucore:coreMetadata>
              <!-- Full Description goes here ... -->
            </ebucore:coreMetadata>
          </ebucore:ebuCoreMain>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
```

```
    </AccessUnit>
  </GetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeContentResponse>
```

EXAMPLE 2 The following example shows the run of the Get Describe Content Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of Content metadata to MPEG-7 MDS. The client sends a `GetDescribeContentRequest` message to the server, requesting the title and the file format of the Content.

```
<GetDescribeContentRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <EntityIdentifier>http://example.com/content/some.video</EntityIdentifier>
  <fru:FRU>
    <fru:Query>//mpeg7:CreationInformation/mpeg7:Creation/mpeg7:Title
</fru:Query>

<fru:Query>//mpeg7:MediaInformation/mpeg7:MediaProfile/mpeg7:MediaFormat/mpeg7:Fi
leFormat</fru:Query>
  </fru:FRU>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeContentRequest>
```

The server processes the request (i.e., passes the `fru:FRU` element to the FRU Request Processor) and responds with a `GetDescribeContentResponse` message containing the `AccessUnit` element with nested `mpeg7:FragementUpdateUnit` elements. Each `mpeg7:FragementUpdateUnit` element conveys a matching node for the XPATH1 queries from the `GetDescribeContentRequest` message. The `TransactionIdentifier` stays the same over an entire session.

```
<GetDescribeContentResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <GetDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>

<mpeg7:FUContext>/mpeg7:Mpeg7/mpeg7:Description[1]/mpeg7:MultimediaContent/mpeg7:
AudioVisual/mpeg7:CreationInformation/mpeg7:Creation
</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <mpeg7:Title xml:lang="en">Some Video</mpeg7:Title>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>

<mpeg7:FUContext>/mpeg7:Mpeg7/mpeg7:Description[1]/mpeg7:MultimediaContent/mpeg7:
AudioVisual/mpeg7:CreationInformation/mpeg7:Creation
</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <mpeg7:Title xml:lang="de">Irgendein Video</mpeg7:Title>
```

```
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>

<mpeg7:FUContext>/mpeg7:Mpeg7/mpeg7:Description[1]/mpeg7:MultimediaContent/mpeg7:
AudioVisual/mpeg7:MediaInformation/mpeg7:MediaProfile/mpeg7:MediaFormat/mpeg7:Fil
eFormat</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <mpeg7:Name>mp4</mpeg7:Name>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
    </AccessUnit>
  </GetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeContentResponse>
```

5.10.2.4 Extension to SID

For the representation of Service Instance Declarations of *Describe Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="DescribeContentSIDType">
    <complexContent>
      <extension base="sid:DescribeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.10.3 Describe Device

5.10.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Describe Device elementary service.

The Describe Device Protocols allow access to descriptions (or parts thereof) of Device entities. The protocols extend the Describe Entity Protocols defined in 5.10.1.

The Device to be described is identified through its `ipmpinfo-msx:ToolBodyID`.

Device metadata formats are specified by the SID (see Annex C).

5.10.3.2 Set Describe Device Protocol

5.10.3.2.1 Interfaces and protocol specification

The Set Describe Device Protocol specifies how to set the description (or parts thereof) of a Device entity. The protocol extends the Set Describe Entity Protocol defined in 5.10.1.2.1, from which it inherits the protocol specification.

5.10.3.2.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  SetDescribeDeviceProtocol                       -->
  <!-- ################################################ -->

  <!-- Definition of SetDescribeDeviceRequest -->
  <element name="SetDescribeDeviceRequest"
type="mpegm:SetDescribeEntityRequestType"/>

  <!-- Definition of SetDescribeDeviceResponse -->
  <element name="SetDescribeDeviceResponse"
type="mpegm:SetDescribeEntityResponseType"/>
```

5.10.3.2.3 Semantics of protocol data format

Semantics of the `SetDescribeDeviceRequest`:

Name	*Definition*
`SetDescribeDeviceRequest`	Protocol message sent from the client to the server in order to set the description (or parts thereof) of a Device entity.
`EntityIdentifier`	A Device Identifier. (See semantics of the `SetDescribeEntityRequestType` in 5.10.1.2.3.)

Semantics of the `SetDescribeDeviceResponse`:

Name	*Definition*
`SetDescribeDeviceResponse`	Protocol message sent from the server to the client to indicate whether the protocol was carried out with success or otherwise.

5.10.3.2.4 Additional validation rules

5.10.3.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.3.2.4.2 The `EntityIdentifier` must be a Device Identifier represented as `ipmpinfo-msx:ToolBodyID`.

5.10.3.2.5 Example

EXAMPLE This example shows the run of the Set Describe Device Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of Device metadata to MPEG-21 DIA UED. The client sends a `SetDescribeDeviceRequest` message to the server, containing a full description of the Device.

```
<SetDescribeDeviceRequest>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:device:some.device</EntityIdentifier>
  <AccessUnit>
    <mpeg7:FragmentUpdateUnit>
      <mpeg7:FUCommand>replaceNode</mpeg7:FUCommand>
      <mpeg7:FUContext>/.</mpeg7:FUContext>
      <mpeg7:FUPayload>
        <dia:DIA>
          <!-- Full Description goes here... -->
        </dia:DIA>
      </mpeg7:FUPayload>
    </mpeg7:FragmentUpdateUnit>
  </AccessUnit>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeDeviceRequest>
```

The server processes the request (i.e., sets the description for the Device) and responds with a `SetDescribeDeviceResponse` message. The `TransactionIdentifier` stays the same over an entire session.

```
<SetDescribeDeviceResponse>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>
  <SetDescribeEntitySuccess/>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeDeviceResponse>
```

5.10.3.3 Get Describe Device Protocol

5.10.3.3.1 Interfaces and protocol specification

The Get Describe Device Protocol specifies how to get the description (or parts thereof) of a Device entity. The protocol extends the Get Describe Entity Protocol defined in 5.10.1.3.2, from which it inherits the protocol specification.

5.10.3.3.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  GetDescribeDeviceProtocol                       -->
  <!-- ################################################ -->

  <!-- Definition of GetDescribeDeviceRequest -->
  <element name="GetDescribeDeviceRequest"
type="mpegm:GetDescribeEntityRequestType"/>

  <!-- Definition of GetDescribeDeviceResponse -->
  <element name="GetDescribeDeviceResponse"
type="mpegm:GetDescribeEntityResponseType"/>
```

5.10.3.3.3 Semantics of protocol data format

Semantics of the `GetDescribeDeviceRequest`:

Name	*Definition*
`GetDescribeDeviceRequest`	Protocol message sent from the client to the server in order to get the description (or parts thereof) of a Device entity.
`EntityIdentifier`	A Device Identifier. (See semantics of the `GetDescribeEntityRequestType` in 5.10.1.3.3.)

Semantics of the `GetDescribeDeviceResponse`:

Name	*Definition*
`GetDescribeDeviceResponse`	Protocol message sent from the server to the client to convey the description (or parts thereof) of the Device entity requested through the preceding `GetDescribeDeviceRequest` message.

5.10.3.3.4 Additional validation rules

5.10.3.3.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.3.3.4.2 The `EntityIdentifier` must be a Device Identifier represented as `ipmpinfo-msx:ToolBodyID`.

5.10.3.3.5 Example

EXAMPLE This example shows the run of the Get Describe Device Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of Device metadata to MPEG-21 DIA UED. The client sends a `GetDescribeDeviceRequest` message to the server, requesting the full description of the Device.

```
<GetDescribeDeviceRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:device:some.device</EntityIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeDeviceRequest>
```

The server processes the request (i.e., packs the description into an `mpeg7:FragmentUpdateUnit` element) and responds with a `GetDescribeDeviceResponse` message containing the `AccessUnit` element with a nested `mpeg7:FragementUpdateUnit` element. The `TransactionIdentifier` stays the same over an entire session.

NOTE Unlike the `SetDescribeContentRequest` message in 5.10.3.2.5, the `mpeg7:FUCommand` is set to "addNode" because the client is expected to have no prior description of the Device.

```
<GetDescribeDeviceResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <GetDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>
        <mpeg7:FUContext>/.</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <dia:DIA>
            <!-- Full Description goes here ... -->
          </dia:DIA>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
    </AccessUnit>
  </GetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeDeviceResponse>
```


5.10.3.4 Extension to SID

For the representation of Service Instance Declarations of *Describe Device*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="DescribeDeviceSIDType">
    <complexContent>
      <extension base="sid:DescribeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.10.4 Describe Service

5.10.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Describe Service elementary service.

The Describe Service Protocols allow access to descriptions (or parts thereof) of Service Instance Declarations (SIDs). The protocols extend the Describe Entity Protocols defined in 5.10.1.

The Service Instance Declaration to be described is identified through its `SIDIdentifier`.

Service Instance Declaration metadata formats are specified by the SID (see Annex C).

5.10.4.2 Set Describe Service Protocol

5.10.4.2.1 Interfaces and protocol specification

The Set Describe Service Protocol specifies how to set the description (or parts thereof) of a Service entity. The protocol extends the Set Describe Entity Protocol defined in 5.10.1.2.1, from which it inherits the protocol specification.

5.10.4.2.2 Syntax of protocol data format

```
  <!-- ############################################## -->
  <!--  SetDescribeServiceProtocol                    -->
  <!-- ############################################## -->

  <!-- Definition of SetDescribeServiceRequest -->
  <element name="SetDescribeServiceRequest"
type="mpegm:SetDescribeEntityRequestType"/>

  <!-- Definition of SetDescribeServiceResponse -->
  <element name="SetDescribeServiceResponse"
type="mpegm:SetDescribeEntityResponseType"/>
```

5.10.4.2.3 Semantics of protocol data format

Semantics of the `SetDescribeServiceRequest`:

Name	*Definition*
`SetDescribeServiceRequest`	Protocol message sent from the client to the server in order to set the description (or parts thereof) of a Service entity.
`EntityIdentifier`	A Service Instance Declaration (SID) Identifier. (See semantics of the `SetDescribeEntityRequestType` in 5.10.1.2.3.)

Semantics of the `SetDescribeServiceResponse`:

Name	*Definition*
`SetDescribeServiceResponse`	Protocol message sent from the server to the client to indicate whether the protocol was carried out with success or otherwise.

5.10.4.2.4 Additional validation rules

5.10.4.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.4.2.4.2 The `EntityIdentifier` must be a Service Instance Identifier represented as `mpegmb:SIDIdentifier`.

5.10.4.2.5 Example

EXAMPLE This example shows the run of the Set Describe Service Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of Service Instance Declaration to the XML Schema definition for Service Instance Declaration provided in Annex C. The client sends a `SetDescribeServiceRequest` message to the server, containing a full description of the Service Instance Declaration.

```
<SetDescribeServiceRequest>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>

<EntityIdentifier>urn:example:service:instances:some.service:example.implementati
on</EntityIdentifier>
  <AccessUnit>
    <mpeg7:FragmentUpdateUnit>
      <mpeg7:FUCommand>replaceNode</mpeg7:FUCommand>
      <mpeg7:FUContext>/.</mpeg7:FUContext>
      <mpeg7:FUPayload>
        <sid:ServiceInstanceDeclaration>
          <sid:Name>Example Implementation of Some Service</sid:Name>

<sid:GeneralServiceDefinition>urn:example:service:some.service</sid:GeneralServic
eDefinition>
          <!-- Full Description goes here ... -->
        </sid:ServiceInstanceDeclaration>
      </mpeg7:FUPayload>
    </mpeg7:FragmentUpdateUnit>
  </AccessUnit>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeServiceRequest>
```

The server processes the request (i.e., sets the description for the Service) and responds with a `SetDescribeServiceResponse` message. The `TransactionIdentifier` stays the same over an entire session.

```
<SetDescribeServiceResponse>
  <mpegmb:TransactionIdentifier>36</mpegmb:TransactionIdentifier>
  <SetDescribeEntitySuccess/>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeServiceResponse>
```

5.10.4.3 Get Describe Service Protocol

5.10.4.3.1 Interfaces and protocol specification

The Get Describe Service Protocol specifies how to get the description (or parts thereof) of a Service entity. The protocol extends the Get Describe Entity Protocol defined in 5.10.1.3.2, from which it inherits the protocol specification.

5.10.4.3.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  GetDescribeServiceProtocol                       -->
  <!-- ################################################ -->

  <!-- Definition of GetDescribeServiceRequest -->
  <element name="GetDescribeServiceRequest"
type="mpegm:GetDescribeEntityRequestType"/>

  <!-- Definition of GetDescribeServiceResponse -->
  <element name="GetDescribeServiceResponse"
type="mpegm:GetDescribeEntityResponseType"/>
```

5.10.4.3.3 Semantics of protocol data format

Semantics of the `GetDescribeServiceRequest`:

Name	*Definition*
`GetDescribeServiceRequest`	Protocol message sent from the client to the server in order to get the description (or parts thereof) of a Service Instance.
`EntityIdentifier`	A Service Instance Declaration (SID) Identifier. (See semantics of the `GetDescribeEntityRequestType` in 5.10.1.3.3.)

Semantics of the `GetDescribeServiceResponse`:

Name	*Definition*
`GetDescribeServiceResponse`	Protocol message sent from the server to the client to convey the description (or parts thereof) of the Service entity requested through the preceding `GetDescribeServiceRequest` message.

5.10.4.3.4 Additional validation rules

5.10.4.3.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.4.3.4.2 The `EntityIdentifier` must be an SID Identifier represented as `mpegmb:SIDIdentifier`.

5.10.4.3.5 Example

EXAMPLE This example shows the run of the Get Describe Service Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of Service Instance Declaration to the XML Schema definition for Service Instance Declaration provided in Annex C. The client sends a `GetDescribeServiceRequest` message to the server, requesting the full description of the Service Instance Declaration.

```
<GetDescribeServiceRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:service:instances:some.service</EntityIdentifier>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeServiceRequest>
```

The server processes the request (i.e., packs the description into an `mpeg7:FragmentUpdateUnit` element) and responds with a `GetDescribeServiceResponse` message containing the `AccessUnit` element with a nested `mpeg7:FragementUpdateUnit` element. The `TransactionIdentifier` stays the same over an entire session.

NOTE Unlike the `SetDescribeContentRequest` message in 5.10.3.2.5, the `mpeg7:FUCommand` is set to "addNode" because the client is expected to have no prior description of the Service Instance Declaration.

```
<GetDescribeServiceResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <GetDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>
        <mpeg7:FUContext>/.</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <sid:ServiceInstanceDeclaration>
            <!-- Full Description goes here ... -->
          </sid:ServiceInstanceDeclaration>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
    </AccessUnit>
  </GetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeServiceResponse>
```

5.10.4.4 Extension to SID

For the representation of Service Instance Declarations of *Describe Service*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="DescribeServiceSIDType">
    <complexContent>
      <extension base="sid:DescribeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.10.5 Describe User

5.10.5.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Describe User elementary service.

The Describe User Protocols allow access to descriptions (or parts thereof) of User entities. The protocols extend the Describe Entity Protocols defined in 5.10.1.

The User to be described is identified through its `UserIdentifier`.

User metadata formats are specified by the SID (see Annex C).

5.10.5.2 Set Describe User Protocol

5.10.5.2.1 Interfaces and protocol specification

The Set Describe User Protocol specifies how to set the description (or parts thereof) of a User entity. The protocol extends the Set Describe Entity Protocol defined in 5.10.1.2.1, from which it inherits the protocol specification.

5.10.5.2.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  SetDescribeUserProtocol                         -->
  <!-- ################################################ -->

  <!-- Definition of SetDescribeUserRequest -->
  <element name="SetDescribeUserRequest"
type="mpegm:SetDescribeEntityRequestType"/>

  <!-- Definition of SetDescribeUserResponse -->
  <element name="SetDescribeUserResponse"
type="mpegm:SetDescribeEntityResponseType"/>
```

5.10.5.2.3 Semantics of protocol data format

Semantics of the `SetDescribeUserRequest`:

Name	*Definition*
`SetDescribeUserRequest`	Protocol message sent from the client to the server in order to set the description (or parts thereof) of a User entity.
`EntityIdentifier`	A User Identifier. (See semantics of the `SetDescribeEntityRequestType` in 5.10.1.2.3.)

Semantics of the `SetDescribeUserResponse`:

Name	*Definition*
`SetDescribeUserResponse`	Protocol message sent from the server to the client to indicate whether the protocol was carried out with success or otherwise.

5.10.5.2.4 Additional validation rules

5.10.5.2.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.5.2.4.2 The `EntityIdentifier` must be a User Identifier represented as `mpegmb:UserIdentifier`.

5.10.5.2.5 Example

EXAMPLE This example shows the run of the Set Describe User Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of User metadata to MPEG-7 MDS with a user profile extension. The client sends a `SetDescribeUserRequest` message to the server, containing an update to the already existing User description. The `mpeg7:FragmentUpdateUnit` of the message specifies that the genre "Thriller" shall be added to the User's preferences.

```
<SetDescribeUserRequest>
  <mpegmb:TransactionIdentifier>54</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:user:john.doe</EntityIdentifier>
  <AccessUnit>
    <mpeg7:FragmentUpdateUnit>
      <mpeg7:FUCommand>addNode</mpeg7:FUCommand>
      <mpeg7:FUContext>
/dia:DIA/dia:Description/dia:UsageEnvironmentProperty/dia:User/dia:UserCharacteri
stic[3]/dia:UsagePreferences/mpeg7:FilteringAndSearchPreferences/mpeg7:Classifica
tionPreferences</mpeg7:FUContext>
      <mpeg7:FUPayload>
        <mpeg7:Genre href="urn:mpeg:mpeg7:cs:GenreCS:2001:6.8">
          <mpeg7:Name>Thriller</mpeg7:Name>
        </mpeg7:Genre>
      </mpeg7:FUPayload>
    </mpeg7:FragmentUpdateUnit>
  </AccessUnit>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeUserRequest>
```

The server processes the request (i.e., updates the description of the User) and responds with a `SetDescribeUserResponse` message. The `TransactionIdentifier` stays the same over an entire session.

```
<SetDescribeUserResponse>
  <mpegmb:TransactionIdentifier>54</mpegmb:TransactionIdentifier>
  <SetDescribeEntitySuccess/>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</SetDescribeUserResponse>
```

5.10.5.3 Get Describe User Protocol

5.10.5.3.1 Interfaces and protocol specification

The Get Describe User Protocol specifies how to get the description (or parts thereof) of a User entity. The protocol extends the Get Describe Entity Protocol defined in 5.10.1.3.1, from which it inherits the protocol specification.

5.10.5.3.2 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  GetDescribeUserProtocol                         -->
  <!-- ################################################ -->

  <!-- Definition of GetDescribeUserRequest -->
  <element name="GetDescribeUserRequest"
type="mpegm:GetDescribeEntityRequestType"/>

  <!-- Definition of GetDescribeUserResponse -->
  <element name="GetDescribeUserResponse"
type="mpegm:GetDescribeEntityResponseType"/>
```

5.10.5.3.3 Semantics of protocol data format

Semantics of the `GetDescribeUserRequest`:

Name	*Definition*
`GetDescribeUserRequest`	Protocol message sent from the client to the server in order to get the description (or parts thereof) of a User entity.
`EntityIdentifier`	A User Identifier. (See semantics of the `GetDescribeEntityRequestType` in 5.10.1.3.3.)

Semantics of the `GetDescribeUserResponse`:

Name	*Definition*
`GetDescribeUserResponse`	Protocol message sent from the server to the client to convey the description (or parts thereof) of the User entity requested through the preceding `GetDescribeUserRequest` message.

5.10.5.3.4 Additional validation rules

5.10.5.3.4.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.10.5.3.4.2 The `EntityIdentifier` must be a User Identifier represented as `mpegmb:UserIdentifier`.

5.10.5.3.5 Examples

EXAMPLE 1 This example shows the run of the Get Describe User Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of User metadata to MPEG-7 MDS with a given User Profile extension and a Payment extension. The client sends a `GetDescribeUserRequest` message to the server, requesting the usage history of another user.

NOTE The namespace prefix "`dia`" is registered for the `GetDescribeUserRequest` element although it is never used by any XML elements. However, it is required in order to correctly interpret the XPATH1 query of the `fru:FRU` element.

```
<GetDescribeUserRequest
xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS">
  <mpegmb:TransactionIdentifier>8128</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:user:john.doe</EntityIdentifier>
  <fru:FRU>
    <fru:Query fru:levelDepth="-1">
//dia:User/dia:UserCharacteristic/dia:UsageHistory</fru:Query>
  </fru:FRU>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeUserRequest>
```

The server processes the request (i.e., passes the `fru:FRU` element to the FRU Request Processor) and responds with a `GetDescribeUserResponse` message containing the `AccessUnit` element with a nested `mpeg7:FragementUpdateUnit` element. The `mpeg7:FragementUpdateUnit` element conveys the matching node for the XPATH1 query from the `GetDescribeUserRequest` message. The `TransactionIdentifier` stays the same over an entire session.

NOTE Again, the "`dia`" namespace prefix is registered for the message although it is only needed to correctly interpret the navigation path of the `mpeg7:FUContext` element.

```
<GetDescribeUserResponse
xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS">
  <mpegmb:TransactionIdentifier>8128</mpegmb:TransactionIdentifier>
  <getDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>

<mpeg7:FUContext>/dia:DIA/dia:Description/dia:UsageEnvironmentProperty/dia:User/d
ia:UserCharacteristic[4]/dia:UsageHistory</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <mpeg7:UserActionHistory>
            <mpeg7:ObservationPeriod>
              <mpeg7:TimePoint>2010-02-21T00:00:00+01:00</mpeg7:TimePoint>
              <mpeg7:Duration>P1D</mpeg7:Duration>
            </mpeg7:ObservationPeriod>
            <mpeg7:UserActionList numOfInstances="1" totalDuration="PT1H31M">
              <mpeg7:ActionType href="urn:mpeg:mpeg7:cs:ActionTypeCS:2001:1.2">
                <mpeg7:Name>PlayStream</mpeg7:Name>
              </mpeg7:ActionType>
            </mpeg7:UserActionList>
          </mpeg7:UserActionHistory>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
    </AccessUnit>
```

```
  </getDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeUserResponse>
```

EXAMPLE 2 This example shows the run of the Get Describe User Protocol. The `strictMetadataSupport` attribute of the `sid:ServiceInstanceDeclaration` is set to "true", constraining the representation format of User metadata to MPEG-7 MDS with a User Profile extension and a Payment extension. The client sends a `GetDescribeUserRequest` message to the server, requesting the reputation of another user.

NOTE The namespace prefix "`dia`" is registered for the `GetDescribeUserRequest` element although it is never used by any XML elements. However, it is required in order to correctly interpret the XPATH1 query of the `fru:FRU` element.

```
<GetDescribeUserRequest
xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS">
  <mpegmb:TransactionIdentifier>8129</mpegmb:TransactionIdentifier>
  <EntityIdentifier>urn:example:user:john.doe</EntityIdentifier>
  <fru:FRU>
    <fru:Query fru:levelDepth="-1">
//dia:User/dia:UserCharacteristic/DefaultRecordList</fru:Query>
  </fru:FRU>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeUserRequest>
```

The server processes the request (i.e., passes the `fru:FRU` element to the FRU Request Processor) and responds with a `GetDescribeUserResponse` message containing the `AccessUnit` element with a nested `mpeg7:FragementUpdateUnit` element. The `mpeg7:FragementUpdateUnit` element conveys the matching node for the XPATH1 query from the `GetDescribeUserRequest` message. The `TransactionIdentifier` stays the same over an entire session.

NOTE Again, the "`dia`" namespace prefix is registered for the message although it is only needed to correctly interpret the navigation path of the `mpeg7:FUContext` element.

```
<GetDescribeUserResponse
xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS">
  <mpegmb:TransactionIdentifier>8129</mpegmb:TransactionIdentifier>
  <GetDescribeEntitySuccess>
    <AccessUnit>
      <mpeg7:FragmentUpdateUnit>
        <mpeg7:FUCommand>addNode</mpeg7:FUCommand>

<mpeg7:FUContext>/dia:DIA/dia:Description/dia:UsageEnvironmentProperty/dia:User/d
ia:UserCharacteristic[5]/DefaultRecordList</mpeg7:FUContext>
        <mpeg7:FUPayload>
          <DefaultRecord type="custom:ExtendedDefaultRecord">
            <IssuerID>urn:example:payment-SP:some.bank
</IssuerID>
            <UserID>urn:example:user:john.doe</UserID>
            <!-- Custom data on payment default of the user... -->
            <dsig:Signature>
              <!-- Signature goes here ... -->
            </dsig:Signature>
```

```
          </DefaultRecord>
        </mpeg7:FUPayload>
      </mpeg7:FragmentUpdateUnit>
    </AccessUnit>
  </GetDescribeEntitySuccess>
  <!-- Signature of this message: -->
  <dsig:Signature>
    <!-- Signature goes here ... -->
  </dsig:Signature>
</GetDescribeUserResponse>
```

5.10.5.4 Extension to SID

For the representation of Service Instance Declarations of *Describe User*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="DescribeUserSIDType">
    <complexContent>
      <extension base="sid:DescribeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.11 The Identify Services

5.11.1 Identify Entity

5.11.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify Entity elementary service. The protocol of this Elementary Service has to be used as a guide for any extra Identify Service that may be specified, apart from the ones included in the following Sections.

This Elementary Service enables a User to obtain an Identifier to an Entity. On success, its response contains that Entity containing the identifier assigned to it.

5.11.1.2 Interfaces and protocol specification

The Identify Entity Protocol is as follows:

Steps	Client	Service Provider
1.	A User sends an `IdentifyEntityRequest` message. This message should contain an Entity to be identified.	
2.		The SP sends back a message containing an identifier or a hash of the identified data or an identified Entity or a reference to it using an `IdentifyEntityResponse`.

5.11.1.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Identify Entity                                 -->
<!-- ################################################ -->

<!-- Definition of IdentifyEntityRequest -->
  <complexType name="IdentifyEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="IdentificationEntity" type="mpegmb:EntityBaseType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<!-- Definition of IdentifyEntityResponse -->
  <complexType name="IdentifyEntityResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="IdentifyEntitySuccess"
type="mpegm:IdentifyEntitySuccessType"/>
          <element name="IdentifyEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IdentifyEntitySuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="IdentifiedEntity" type="mpegmb:EntityBaseType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.11.1.4 Semantics of protocol data format

Semantics of the `IdentifyEntityRequest`:

Name	*Definition*
`IdentifyEntityRequest`	The message for creating a request for `IdentifyEntity`.
`IdentifyEntityRequestType`	Top-level type for `IdentifyEntityRequest`. `IdentifyEntityRequestType` extends `ProtocolRequestType`.

Semantics of the `IdentifyEntityRequestType`:

Name	*Definition*
`IdentificationEntity`	An Entity to be identified or a reference to it.

Semantics of the `IdentifyEntityResponse`:

Name	*Definition*
`IdentifyEntityResponse`	The response contains the result of the `IdentifyEntity` Service.
`IdentifyEntityResponseType`	Top-level type for `IdentifyEntityResponse`. `IdentifyEntityResponseType` extends `ProtocolSuccessType`. If successful it contains the identified Entity or the identifier of the identified data.
`IdentifyEntitySuccess`	Response in case of success.
`IdentifyEntityFailure`	Response in case of failure.

Semantics of the `IdentifyEntitySuccessType`:

Name	*Definition*
`IdentifiedEntity`	The Entity or a reference to it.

5.11.1.5 Extension to SID

For the representation of Service Instance Declarations of Identify Entity, the following complex type is defined in the SID XML Schema:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="IdentifyEntitySIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.11.2 Identify Content

5.11.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify Content elementary service.

This Elementary Service enables a User to obtain an Identifier to a Digital Item or any of its component elements. The response should be either an identifier or an identified DI or a reference to it.

The Service Provider providing the Identify Content Service keeps a store of Digital Items, after their proper registration done by the Identify operation. The Identify operation may be used in two different situations: to register new content, or to obtain the already existing identification.

5.11.2.2 Interfaces and protocol specification

The Identify Content Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends an `IdentifyContentRequest` message. This message should contain a DI to be identified.	
2.		The SP sends back a message containing an identifier or a hash of the identified data or an identified DI or a reference to it using an `IdentifyContentResponse`.

5.11.2.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Identify Content                                -->
<!-- ################################################ -->

<!-- Definition of IdentifyContentRequest -->
<element name="IdentifyContentRequest" type="mpegm:IdentifyContentRequestType"/>
  <complexType name="IdentifyContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="IdentificationContent" type="mpegmb:ContentEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of IdentifyContentResponse -->
<element name="IdentifyContentResponse"
type="mpegm:IdentifyContentResponseType"/>
  <complexType name="IdentifyContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="IdentifyContentSuccess"
type="mpegm:IdentifyContentSuccessType"/>
          <element name="IdentifyContentFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IdentifyContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="IdentifiedContent" type="mpegmb:ContentEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.11.2.4 Semantics of protocol data format

Semantics of the `IdentifyContentRequest`:

Name	*Definition*
`IdentifyContentRequest`	The message for creating a request for IdentifyContent
`IdentifyContentRequestType`	Top-level type for `IdentifyContentRequest`. `IdentifyContentRequestType` extends `ProtocolRequestType`.

Semantics of the `IdentifyContentRequestType`:

Name	*Definition*
`IdentificationContent`	A Digital Item to be identified or a reference to it.

Semantics of the `IdentifyContentResponse`:

Name	*Definition*
`IdentifyContentResponse`	The response contains the result of the IdentifyContent Service.
`IdentifyContentResponseType`	Top-level type for `IdentifyContentResponse`. `IdentifyContentResponseType` extends `ProtocolSuccessType`. If successful it contains the identified digital item or the identifier of the identified data.
`IdentifyContentSuccess`	Response in case of success.
`IdentifyContentFailure`	Response in case of failure.

Semantics of the `IdentifyContentSuccessType`:

Name	*Definition*
`IdentifiedContent`	The identified Digital Item or a reference to it.

5.11.2.5 Extension to SID

For the representation of Service Instance Declarations of *Identify Content*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="IdentifyContentSIDType">
  <complexContent>
```

```
      <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.11.3 Identify Contract

5.11.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify Contract Protocol.

This Elementary Service enables a User to obtain an Identifier to a Contract. The response should be the identifier or the general failure message.

The Service Provider providing the Identify Contract Service keeps a store of Contracts, after their proper registration done by the Identify operation. The Identify operation may be used in two different function situations: to register a new contract, or to obtain their already existing identification.

Identify Contract allows Users to assign Identifiers to Contracts in a multimedia content value chain.

5.11.3.2 Interfaces and protocol specification

The Identify Content Protocol is as follows:

Steps	Client	Service Provider
1.	A User sends an `IdentifyContractRequest` message. This message should contain a MPEG-21 Part 20 Contract to be identified.	
2.		The SP sends back a message containing an identifier or a hash of the identified data or an identified Contract or a reference to it using an `IdentifyContractResponse`.

5.11.3.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Identify Contract                               -->
<!-- ################################################ -->

<!-- Definition of IdentifyContractRequest -->
<element name="IdentifyContractRequest"
type="mpegm:IdentifyContractRequestType"/>
<complexType name="IdentifyContractRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="IdentificationContract" type="mpegmb:ContractEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<!-- Definition of IdentifyContractResponse -->
<element name="IdentifyContractResponse"
type="mpegm:IdentifyContractResponseType"/>
  <complexType name="IdentifyContractResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="IdentifyContractSuccess"
type="mpegm:IdentifyContractSuccessType"/>
            <element name="IdentifyContractFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<complexType name="IdentifyContractSuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="IdentifiedContract" type="mpegmb:ContractEntityType"
minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```


5.11.3.4 Semantics of protocol data format

Semantics of the `IdentifyContractRequest`:

Name	*Definition*
`IdentifyContractRequest`	The Request to Identify a Contract.
`IdentificationContract`	A MPEG-21 CEL contract as described in 5.2.3.10.

Semantics of the `IdentifyContractResponse`:

Name	*Definition*
`IdentifyContractResponse`	The response to identify a Contract.
`IdentifyContractSuccess`	Sent in case of success.
`IdentifyContractFailure`	Sent in case of failure.
`IdentifyContractSuccessType`	Container of the contract identifier.
`IdentifiedContract`	The contract containing the requested identifier or the identifier itself.

5.11.3.5 Extension to SID

For the representation of Service Instance Declarations of *Identify Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="IdentifyContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.11.4 Identify Device

5.11.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify Device elementary service.

This Elementary Service enables a User to obtain a unique certificate (related to the Device in use) from an Identify Device Service Provider (e.g., a Certification Authority).

NOTE The semantics of Identify Device differ from the semantics of Identify Content, Identify Contract, and Identify User. Identify Content, Identify Contract, and Identify User have the purpose to register a new content/contract/user, or to obtain their already existing identification. However, Identify Device enables a User to obtain a certificate related to the Device.

5.11.4.2 Interfaces and protocol specification

The Identify Device Protocol is as follows:

Steps	Client	Service Provider
1.	The client generates an `IdentifyDeviceRequest` message containing the description of its hardware and software characteristics and its own certificate or public key and sends the message to the Service Provider.	
2.		The SP verifies the digital signature of the message if present.
3.		The SP extracts the information from the message and decides whether to accept the request. In both cases generates an `IdentifyDeviceResponse` message containing either the `IdentifyDeviceResponseSuccess` element indicating the successful completion of the Protocol, or the `IdentifyDeviceResponseFailure` element indicating the reason of failure.

Steps	Client	Service Provider
		For success response, it returns the device's certificate signed by the server.

5.11.4.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Identify Device                                 -->
<!-- ################################################ -->

<!-- Definition of IdentifyDeviceRequest -->
<element name="IdentifyDeviceRequest" type="mpegm:IdentifyDeviceRequestType"/>
  <complexType name="IdentifyDeviceRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="DeviceKey" type="dsig:KeyInfoType" minOccurs="0"/>
          <element ref="mpeg4ipmp:TerminalID" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<!-- Definition of IdentifyDeviceResponse -->
<element name="IdentifyDeviceResponse" type="mpegm:IdentifyDeviceResponseType"/>
  <complexType name="IdentifyDeviceResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="IdentifyDeviceSuccess"
type="mpegm:IdentifyDeviceSuccessType"/>
          <element name="IdentifyDeviceFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IdentifyDeviceSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="DeviceKey" type="dsig:KeyInfoType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.11.4.4 Semantics of protocol data format

Semantics of the `IdentifyDeviceRequest`:

Name	*Definition*
`IdentifyDeviceRequest`	The message for creating a request for IdentifyDevice
`IdentifyDeviceRequestType`	Top-level type for `IdentifyDeviceRequest`. `IdentifyDeviceRequestType` extends `ProtocolRequestType`.
`DeviceKey`	The digital certificate or public key of the device requesting an identifier.
`mpeg4ipmp:TerminalID`	Element describing the hardware and software characteristics of the device as defined in ISO/IEC 14496-13.

Semantics of the `IdentifyDeviceResponse`:

Name	*Definition*
`IdentifyDeviceResponse`	The message conveys the Result attribute indicating whether the Protocol was successful or not.
`IdentifyDeviceResponseType`	Top-level type for `IdentifyDeviceResponse`. `IdentifyDeviceResponseType` extends `ProtocolResponseType`.
`IdentifyDeviceResponseSuccess`	Response in case of success.
`IdentifyDeviceResponseFailure`	Response in case of failure.

Semantics of the `IdentifyDeviceResponseSuccessType`:

Name	*Definition*
`IdentifyDeviceResponseSuccessType`	`IdentifyDeviceResponseSuccessType` extends `ProtocolSuccessType`.
`DeviceKey`	The device's certificate signed by the device identification service provider.

5.11.4.5 Extension to SID

For the representation of Service Instance Declarations of *Identify Device*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="IdentifyDeviceSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
```

```
    </complexContent>
  </complexType>
```

5.11.5 Identify License

5.11.5.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify License Protocol.

This Elementary Service enables a User to obtain an Identifier to a License. The response should be the identifier or the general failure message.

The Service Provider providing the Identify License Service keeps a store of Licenses, after their proper registration done by the Identify operation. The Identify operation may be used in two different function situations: to register a new license, or to obtain their already existing identification.

Identify License allows Users to assign Identifiers to Licenses in a multimedia content value chain.

5.11.5.2 Interfaces and protocol specification

The Identify License Protocol is as follows:

Steps	Client	Service Provider
1.	A User sends an `IdentifyLicenseRequest` message. This message should contain a MPEG-21 Part 5 License to be identified.	
2.		The SP sends back a message containing an identifier or a hash of the identified data or an identified License or a reference to it using an `IdentifyLicenseResponse`.

5.11.5.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Identify License                                -->
<!-- ################################################ -->

<!-- Definition of IdentifyLicenseRequest -->
<element name="IdentifyLicenseRequest" type="mpegm:IdentifyLicenseRequestType"/>
<complexType name="IdentifyLicenseRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="IdentificationLicense" type="mpegmb:LicenseEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<!-- Definition of IdentifyLicenseResponse -->
<element name="IdentifyLicenseResponse"
type="mpegm:IdentifyLicenseResponseType"/>
  <complexType name="IdentifyLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="IdentifyLicenseSuccess"
type="mpegm:IdentifyLicenseSuccessType"/>
            <element name="IdentifyLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<complexType name="IdentifyLicenseSuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolSuccessType">
      <sequence>
        <element name="IdentifiedLicense" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.11.5.4 Semantics of protocol data format

Semantics of the `IdentifyLicenseRequest`:

Name	*Definition*
`IdentifyLicenseRequest`	The Request to Identify a License.
`IdentificationLicense`	A MPEG-21 REL license as described in 5.2.3.12.

Semantics of the `IdentifyLicenseResponse`:

Name	*Definition*
`IdentifyLicenseResponse`	The response to identify a License.
`IdentifyLicenseSuccess`	Sent in case of success.
`IdentifyLicenseFailure`	Sent in case of failure.
`IdentifyLicenseSuccessType`	Container of the license identifier.
`IdentifiedLicense`	The license containing the requested identifier or the identifier itself.

5.11.5.5 Extension to SID

For the representation of Service Instance Declarations of Identify License, the following complex type is defined in the SID XML Schema:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->
  <complexType name="IdentifyLicenseSIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.11.6 Identify User

5.11.6.1 Introduction

This clause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Identify User Protocol.

The Identify User elementary service enables a client to perform different operations related to user identification, like user registration or obtaining a user identifier. The relationship of this ES with Authenticate User ES is described in 5.5.5.

5.11.6.2 Interfaces and protocol specification

The Identify User Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends an `IdentifyUserRequest` message to the Service Provider in order to obtain a new identifier. The message may contain a `UserIdentificationAssertion` or `saml:Assertion` element as specified in 3.4 of SAML-CORE-2.0-OS that verifies that the user is authenticated. The client must send a `ContentEntity` which proves the user identity (e.g., fingerprint, iris scan, or username and password).	
2.		The SP sends an `IdentifyUserResponse` message to the client. In case of success, the message contains a new user identifier.

5.11.6.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Identify User                                   -->
  <!-- ################################################ -->
  <!-- Definition of IdentifyUserRequest -->
  <element name="IdentifyUserRequest" type="mpegm:IdentifyUserRequestType"/>
  <complexType name="IdentifyUserRequestType">
```

```
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="UserIdentificationAssertion" type="saml:AssertionType"
minOccurs="0"/>
          <element name="IdentificationUserEntity" type="mpegmb:UserEntityType"
minOccurs="0"/>
          <element name="IdentifyingMaterial" type="mpegmb:ContentEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of IdentifyUserResponse -->
  <element name="IdentifyUserResponse" type="mpegm:IdentifyUserResponseType"/>
  <complexType name="IdentifyUserResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="IdentifyUserSuccess"
type="mpegm:IdentifyUserSuccessType"/>
          <element name="IdentifyUserFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IdentifyUserSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="IdentifiedUser" type="mpegmb:UserEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.11.6.4 Semantics of protocol data format

Semantics of the `IdentifyUserRequest`:

Name	*Definition*
`IdentifyUserRequest`	The message for creating an Identify User request.
`IdentifyUserRequestType`	Top-level type for `IdentifyUserRequest`. `IdentifyUserRequestType` extends `ProtocolRequestType`.
`UserIdentificationAssertion`	Assertion resulting from a user authentication.
`IdentificationUser`	Information about the user (e.g., full name) in case this information is not already provided in the `UserIdentificationAssertion` or `saml:Assertion` element.
`IdentifyingMaterial`	Information (e.g., X.501 certificate [13], fingerprint, iris scan, or username and password) that proves the user's identity.

Semantics of the `IdentifyUserResponse`:

Name	*Definition*
`IdentifyUserResponse`	The response message Identify User.
`IdentifyUserResponseType`	Top-level type for `IdentifyUserResponse`. `IdentifyUserResponseType` extends `ProtocolResponseType`.
`IdentifyUserSucess`	Response in case of success.
`IdentifyUserFailure`	Response in case of failure.

Semantics of the `IdentifyUserSuccessType`:

Name	*Definition*
`IdentifyUserSuccessType`	The response in case of success.
`IdentifiedUser`	User profile containing the requested identifier (e.g., via the `mpeg7:UserProfileType`) or the identifier itself.

5.11.6.5 Additional validation rules

5.11.6.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.11.6.5.2 The `UserIdentificationAssertion` shall only be used if the User to be identified is not the same as the requesting client. If the client requests a user identifier for himself/herself, then the `saml:Assertion` element of the `mpegmb:ProtocolRequestType` shall be used.

5.11.6.6 Extension to SID

For the representation of Service Instance Declarations of *Identify User*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="IdentifyUserSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.12 The Negotiate Services

5.12.1 Negotiate Entity

5.12.1.1 Introduction

The negotiation protocol allows two parties to reach up an agreement on a given item. One of the party will start a negotiation defining which item is the object of the negotiation and which will be the negotiable elements related to the item.

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the abstract Negotiate Entity Protocol.

The Negotiate Entity Protocol assumes that there are in general 3 actors:

a) **Owner** of the Negotiation is the party that first makes recourse to Negotiation Services. This could be anyone willing to buy or sell rights to an item. Owner can be anyone wishing to acquire rights to that item. Every negotiation can have only one Owner.

b) **Participant** in the Negotiation is any party willing to participate in the negotiation. It's important to consider that we can have several simultaneous participants willing to apply for the same call.

c) **Negotiate Entity Service Provider** handling negotiations between Owner and Participants.

The actors involved in this reference model are represented in Figure 1 (note that an Owner can assume either the Seller role or a Buyer role. In the first case the participants will result to be Buyers, in the second case Sellers).

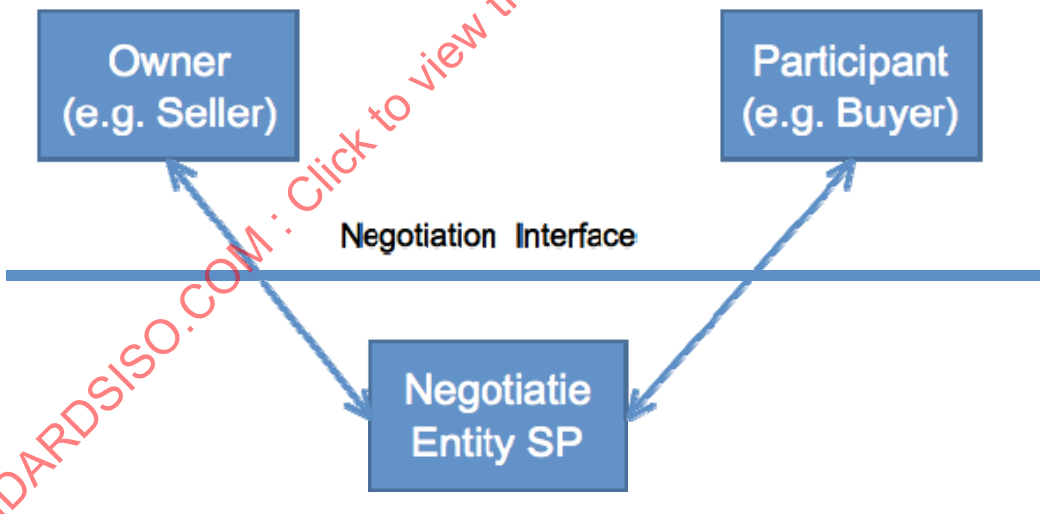

Figure 1 — Actors in a negotiation system

5.12.1.2 General Considerations

During the negotiation process there are several common steps that will take place in order to implement a successful negotiation. The main steps are the following:

Owner decides to set up a Negotiation: This phase contains all activities related to setting-up a negotiation. Owner can either be willing to buy certain goods/services or content or can even decide to sell something that is of interest for a certain group of participants. In particular this means to select a negotiation model template from the repository and to configure the negotiation according to the specific business requirements of the

negotiation owner. Owner can decide:

— To use one of the several negotiation model templates provided by the system: English Auction, Dutch Auction, Reverse Auction and more as defined in G.2.

— To add information that Owner expects will appear in the License or Contract template that will eventually be agreed between Owner and Participant.

— Whether the negotiation is public (open to all) or private (restricted to a subgroup specified in the request).

The Negotiation evolves through the following phases:

a) **Setup Negotiation:** The Owner creates a message to be sent to the SP in order to start a negotiation. This message will indicate whether the negotiation will be public or private, which item will be negotiated and which parameters related to the item will be negotiable (so-called *Issues*) and under which range of values (Conditions).

b) **Admission to Negotiation:** This phase allows participants to join the negotiation advertised by SP. The SP will acknowledge participants whether they are admitted or not to the negotiation.

c) **Offer:** Participants and Owner exchange a set of messages where one of the parties sends an offer or a counter offer for all or for some negotiable parameters. Moreover a response to this message will contain either the acceptance of a proposed value or a different proposal for a given Issues. A consistent amount of messages can be exchanged at this stage. When the parties reach an agreement on all the issues it is possible to pass to the final stage where the negotiation is finalized.

d) **Agreement:** This phase ends the negotiation when owner receives a satisfying offer from a participant.

5.12.1.3 Interfaces and protocol specification

The Negotiate Entity Protocol is as follows:

Steps	Owner	Service Provider	Participants
1.	The owner initiates a new multiparty negotiation license by sending a message of type `SetupNegotiationEntityRequestType`.		
2.		If the message of type `SetupNegotiationEntityRequestType` specifies any participants, than the SP shall forward the message to those participants.	
3.		The SP releases a `NegotiationID` to the Owner sending him/her a message of type `SetupNegotiationEntityResponseType`.	
4.			Participants request to join negotiation by sending a

Steps	Owner	Service Provider	Participants
			message of type `AdmissionEntityRequestType`.
5.		The SP decides whether to accept or not the request for admission to negotiation with a message of type `AdmissionEntityResponseType`.	
6.			A Participant sends a message of type `OfferEntityRequestType` to provide an offer for item.
7.		The SP forwards the request for offers to the owner using a message of type `OfferEntityRequestType`.	
8.	The Owner accepts the negotiation by sending a message of type `OfferEntityResponseType`.		
9.		The SP forwards the message with the decision of the owner with a message of type `OfferEntityResponseType`.	
10.			The Participant receiving a successful response may terminate negotiation by sending a message of type `AgreementEntityRequestType` asking to finalize the negotiation.
11.		The SP forwards to the Owner the agreement request with a message of type `AgreementEntityRequestType`.	
12.	The Owner answers to the final agreement with a message of type		

Steps	Owner	Service Provider	Participants
	`AgreementEntityResponseType`.		
13.		The SP notifies the Participants about the agreement requested to the Owner with a message of type `AgreementEntityResponseType`.	

5.12.1.4 SetupNegotiationEntity

These messages are exchanged to define a negotiation and all the parameters that will be involved in the negotiation, including:

— an Owner, who is starting the negotiation;

— an optional set of Participants, specified whether:

  — only specific actors are allowed to be involved in a negotiation by the SP or

  — the negotiation will be public to anyone who wish to participate;

— an item, the object of the negotiation (e.g., a license);

— a set of issues, the elements that are really negotiable;

— one or more proposed values for each Issue.

5.12.1.4.1 Syntax of protocol data format

```
<!-- Definition of SetupNegotiationEntityRequest -->
  <complexType name="SetupNegotiationEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="NegotiationName" type="string" minOccurs="0"/>
          <element name="NegotiationModel" type="mpeg7:ControlledTermUseType">
            <annotation>
              <documentation xml:lang="en">
                Proposed Classification Scheme: urn:mpeg:mpegM:cs:03-es-
NS:2012:AuctionModelCS
              </documentation>
            </annotation>
          </element>
          <element name="Description" type="mpeg7:TextAnnotationType"
minOccurs="0"/>
          <element name="Owner" type="mpegmb:UserEntityType"/>
```

```
          <element name="Participant" type="mpegmb:UserEntityType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="NegotiationItemTemplate" type="mpegmb:EntityBaseType">
            <annotation>
              <documentation xml:lang="en">
                Each regular Elementary Service puts here an Entity of the
appropriate type (e.g., mpegmb:ContractEntityType).
              </documentation>
            </annotation>
            </element>
          <element name="IssueOffer" type="mpegm:IssueOfferType"
maxOccurs="unbounded"/>
        </sequence>
        <attribute name="publish" type="boolean" use="optional" default="true"/>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IssueOfferType">
    <sequence>
      <element name="Issue" type="mpegm:IssueType"/>
      <element name="ProposedValue" type="anyType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="Description" type="mpeg7:TextAnnotationType" minOccurs="0"/>
    </sequence>
    <attribute name="accepted" type="boolean" use="optional" default="false"/>
  </complexType>
  <complexType name="IssueType">
    <choice>
      <element name="XPath" type="anyURI"/>
      <element name="Term" type="mpegm:ControlledTermUseType">
        <annotation>
          <documentation xml:lang="en">
            Proposed Classification Scheme: urn:mpeg:mpegM:cs:03-es-
NS:2012:IssueTypeCS
          </documentation>
        </annotation>
      </element>
    </choice>
  </complexType>


<!-- Definition of SetupNegotiationEntityResponse -->
<complexType name="SetupNegotiationEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="SetupNegotiationSuccess"
type="mpegm:SetupNegotiationSuccessType"/>
          <element name="SetupNegotiationFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="SetupNegotiationSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="NegotiationIdentifier" type="anyURI"/>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.12.1.4.2 Semantics of protocol data format

Semantics of the `SetupNegotiationEntityRequestType`:

Name	*Definition*
`NegotiationName`	The name given to the negotiation.
`NegotiationModel`	The negotiation model that is used for the negotiation. An example of a Classification Scheme is the AuctionModelCS (`urn:mpeg:mpegM:cs:03-es-NS:2012:AuctionModelCS`) defined in G.2.
`Description`	A description of the negotiation.
`Owner`	The negotiation owner. This is typically the party initiating the negotiation.
`Participant`	One or more participants that can be involved exclusively by the Owner in the negotiation.
`NegotiationItemTemplate`	The Entity under negotiation.
`IssueOffer`	The `NegotiationIssueType` that describes the issues related with the negotiation. All issues that are left blank in the `NegotiationItemTemplate` must be listed.
`publish`	Attribute indicating whether the negotiation shall be *public* (e.g., the SP will be able to advertise it) or not. If the negotiation is *public*, other Users may find and join this negotiation. NOTE The mechanisms for publication and discovery of negotiation is out of the scope of ISO/IEC 23006-4. The default value is *true*, indicating a public negotiation.

Semantics of the `IssueOfferType`:

Name	*Definition*
`Issue`	The issue that is involved in the negotiation. This actually represents which parameters are really negotiable.
`ProposedValue`	One or more proposed values for the issue. This represents the real offer related to a given issue.

`Argument`	Further description of the issue offer (e.g., a rationale).
`accepted`	Attribute indicating whether the parties have agreed on a `ProposedValue`.

Semantics of the `IssueType`:

Name	*Definition*
`XPath`	An issue represented as an XPath into the `NegotiationItemTemplate` XML infoset (e.g., a REL license Grant)
`Term`	Semantic specification of the issue. An example of a Classification Scheme is the `IssueTypeCS` (`urn:mpeg:mpegM:cs:03-es-NS:2012:IssueTypeCS`) defined in G.3.

Semantics of the `SetupNegotiationEntityResponseType`:

Name	*Definition*
`SetupNegotiationSuccess`	Response in case of success.
`SetupNegotiationFailure`	Response in case of failure.

Semantics of the `SetupNegotiationSuccessType`:

Name	*Definition*
`SetupNegotiationSuccessType`	The `SetupNegotiationSuccessType` extends `ProtocolSuccessType`.
`NegotiationIdentifier`	The identifier assigned to the negotiation.

5.12.1.4.3 Additional validation rules

5.12.1.4.3.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.12.1.4.3.2 If at least one `Participant` is specified, the `publish` attribute of the `SetupNegotiationEntityRequestType` is typically set to `false`.

5.12.1.4.3.3 The `accepted` attribute of all `IssueOffer` elements must be set to `false` in the `SetupNegotiationEntityRequestType`.

5.12.1.4.3.4 If the `publish` attribute of the `SetupNegotiationEntityRequestType` is set to `false`, at least one `Participant` must be specified.

5.12.1.4.3.5 The `accepted` attribute of the `IssueType` may only be set to `true` if exactly one `ProposedValue` element is provided (i.e., the accepted value).

5.12.1.5 AdmissionEntity

These messages are exchanged between a User an SP, indicating that the User is willing to join the specified negotiation as a Participant. The SP can accept or reject this request.

5.12.1.5.1 Syntax of protocol data format

```
  <!-- Definition of AdmissionEntityRequest -->
  <complexType name="AdmissionEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="NegotiationIdentifier" type="anyURI"/>
          <element name="Receiver" type="mpegmb:UserEntityType"
maxOccurs="unbounded"/>
          <element name="Sender" type="mpegmb:UserEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of AdmissionEntityResponse -->
  <complexType name="AdmissionEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AdmissionSuccess" type="mpegmb:ProtocolSuccessType"/>
          <element name="AdmissionFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

5.12.1.5.2 Semantics of protocol data format

Semantics of the `AdmissionEntityRequestType`:

Name	*Definition*
`AdmissionEntityRequestType`	Top-level type for `AdmissionEntityRequest`. `AdmissionEntityRequestType` extends `ProtocolRequestType`.
`Receiver`	Receiver of the message.
`Sender`	Sender of the message.

Semantics of the AdmissionEntityResponseType:

Name	*Definition*
AdmissionEntityResponseType	Top-level type for AdmissionEntityResponse. AdmissionEntityResponseType extends ProtocolResponseType.
AdmissionSuccess	Response in case of success.
AdmissionFailure	Response in case of failure.

5.12.1.6 OfferEntity

This will be the messages exchanged among Owner and Participants via SP to define offers and counter-offers that will be involved in a negotiation process.

5.12.1.6.1 Syntax of protocol data format

```
  <!-- Definition of OfferEntityRequest -->
  <complexType name="OfferEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="NegotiationIdentifier" type="anyURI"/>
          <element name="IssueOffer" type="mpegm:IssueOfferType"
maxOccurs="unbounded"/>
          <element name="Sender" type="mpegmb:UserEntityType" minOccurs="0"/>
          <element name="Receiver" type="mpegmb:UserEntityType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="OfferIdentifier" type="anyURI"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of OfferEntityResponse -->
  <complexType name="OfferEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="OfferSuccess" type="mpegm:OfferSuccessType"/>
            <element name="OfferFailure" type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="OfferSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="IssueOffer" type="mpegm:IssueOfferType"
maxOccurs="unbounded"/>
```

```
          <element name="OfferIdentifier" type="anyURI"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.12.1.6.2 Semantics of protocol data format

Semantics of the `OfferEntityRequestType`:

Name	*Definition*
`OfferEntityRequestType`	`OfferEntityRequestType` extends `ProtocolRequestType`.
`NegotiationIdentifier`	The identifier of the negotiation.
`IssueOffer`	A set of one or more `IssueOfferType`.
`Sender`	The sender of the Offer.
`OfferIdentifier`	An identifier for the set of messages exchanged among the Owner and a Participant for negotiating particular issues.

Semantics of the `OfferEntityResponseType`:

Name	*Definition*
`OfferSuccess`	Response in case of success.
`OfferFailure`	Response in case of failure.

Semantics of the `OfferSuccessType`:

Name	*Definition*
`IssueOffer`	A set of one or more issues that are involved in the negotiation.
`OfferIdentifier`	An identifier for the set of messages exchanged among the Owner and a Participant for negotiating particular issues.

5.12.1.7 AgreementEntity

This is the final message exchanged that determines whether the negotiation reaches an agreement. At this exchange the message with the agreement is sent with a template item to be filled by both parties, to end up with a final entity (e.g., contract) that is the agreed one by both parties.

5.12.1.7.1 Syntax of protocol data format

```
  <!-- Definition of AgreementEntityRequest -->
  <complexType name="AgreementEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="FinalItemType">
    <choice>
      <element name="TemplateEntity" type="mpegmb:EntityBaseType">
        <annotation>
          <documentation xml:lang="en">
            Each regular Elementary Service puts here an Authentication Entity of
the appropriate type (e.g., mpegmb:ContentEntityType).
          </documentation>
        </annotation>
      </element>
      <element name="FinalEntity" type="mpegmb:EntityBaseType">
        <annotation>
          <documentation xml:lang="en">
            Must be either of type mpegmb:ContractEntityType or
mpegmb:LicenseEntityType.
          </documentation>
        </annotation>
      </element>
    </choice>
  </complexType>

  <!-- Definition of AgreementEntityResponse -->
  <complexType name="AgreementEntityResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AgreementEntitySuccess"
type="mpegm:AgreementEntitySuccessType"/>
          <element name="AgreementEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AgreementEntitySuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.12.1.7.2 Semantics of protocol data format

Semantics of the `AgreementEntityRequestType`:

Name	*Definition*
`AgreementEntityRequestType`	`AgreementEntityRequestType` extends `ProtocolRequestType`.
`FinalItem`	The template or final entity exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

Semantics of the `FinalItemType`:

Name	*Definition*
`TemplateEntity`	The template version of the entity negotiated and agreed (i.e., contract entity or license entity).
`FinalEntity`	The final and signed version of the entity negotiated and agreed (i.e., contract entity or license entity).

Semantics of the `AgreementEntityResponseType`:

Name	*Definition*
`AgreementEntityResponseType`	`AgreementEntityResponseType` extends `ProtocolResponseType`.
`AgreementEntitySuccess`	Response in case of success.
`AgreementEntityFailure`	Response in case of failure.

Semantics of the `AgreementEntitySuccessType`:

Name	*Definition*
`FinalItem`	The template or final entity exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

5.12.1.8 Extension to SID

For the representation of Service Instance Declarations of *Negotiate Entity*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="NegotiateEntitySIDType" abstract="true">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
```

```
    </complexContent>
  </complexType>
```

5.12.2 Negotiate Contract

5.12.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Negotiate Contract elementary service.

The Negotiate Contract elementary service is based on the abstract Negotiate Entity elementary service defined in 5.12.1.

The item of the negotiation is a CEL Contract represented as `mpegmb:ContractEntityType`.

5.12.2.2 Interfaces and protocol specification

The Negotiate Contract Protocol extends the abstract Negotiate Entity Protocol defined in 5.12.1.3, from which it inherits the protocol specification.

5.12.2.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Negotiation Contract Services                   -->
  <!-- ################################################ -->

  <!-- Definition of SetupNegotiationContractRequest -->
  <element name="SetupNegotiationContractRequest"
type="mpegm:SetupNegotiationContractRequestType"/>
  <complexType name="SetupNegotiationContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="NegotiationName" type="string" minOccurs="0"/>
          <element name="NegotiationModel" type="mpeg7:ControlledTermUseType">
            <annotation>
              <documentation xml:lang="en">
                Proposed Classification Scheme: urn:mpeg:mpegM:cs:03-es-
NS:2012:AuctionModelCS
              </documentation>
            </annotation>
          </element>
          <element name="Description" type="mpeg7:TextAnnotationType"
minOccurs="0"/>
          <element name="Owner" type="mpegmb:UserEntityType"/>
          <element name="Participant" type="mpegmb:UserEntityType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="NegotiationItemTemplate"
type="mpegmb:ContractEntityType">
            </element>
          <element name="IssueOffer" type="mpegm:IssueOfferType"
maxOccurs="unbounded"/>
        </sequence>
        <attribute name="publish" type="boolean" use="optional" default="true"/>
```

```
      </extension>
    </complexContent>
  </complexType>

  <element name="SetupNegotiationContractResponse"
type="mpegm:SetupNegotiationEntityResponseType"/>

  <element name="AdmissionContractRequest"
type="mpegm:AdmissionEntityRequestType"/>
  <element name="AdmissionContractResponse"
type="mpegm:AdmissionEntityResponseType"/>

  <element name="OfferContractRequest" type="mpegm:OfferEntityRequestType"/>
  <element name="OfferContractResponse" type="mpegm:OfferEntityResponseType"/>

  <element name="AgreementContractRequest"
type="mpegm:AgreementContractRequestType"/>
  <element name="AgreementContractResponse"
type="mpegm:AgreementContractResponseType"/>

<!-- Definition of AgreementContractRequest -->
  <complexType name="AgreementContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemContractType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="FinalItemContractType">
    <choice>
      <element name="TemplateContract" type="mpegmb:ContractEntityType"/>
      <element name="FinalContract" type="mpegmb:ContractEntityType"/>
    </choice>
  </complexType>

  <!-- Definition of AgreementContractResponse -->
  <complexType name="AgreementContractResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AgreementContractSuccess"
type="mpegm:AgreementContractSuccessType"/>
          <element name="AgreementContractFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AgreementContractSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemContractType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.12.2.4 Semantics of protocol data format

Semantics of the `SetupNegotiationContractRequest`:

Name	Definition
`SetupNegotiationContractRequest`	This message from the owner launches the negotiation using the supplied parameters.

Semantics of the `SetupNegotiationContractRequestType`:

Name	*Definition*
`NegotiationName`	The name given to the negotiation.
`NegotiationModel`	The negotiation model that is used for the negotiation. An example of a Classification Scheme is the AuctionModelCS (`urn:mpeg:mpegM:cs:03-es-NS:2012:AuctionModelCS`) defined in G.2.
`Description`	A description of the negotiation.
`Owner`	The negotiation owner. This is typically the party initiating the negotiation.
`Participant`	One or more participants that can be involved exclusively by the Owner in the negotiation.
`NegotiationItemTemplate`	The Contract under negotiation.
`IssueOffer`	The `NegotiationIssueType` that describes the issues related with the negotiation. All issues that are left blank in the `NegotiationItemTemplate` must be listed.
`publish`	Attribute indicating whether the negotiation shall be *public* (e.g., the SP will be able to advertise it) or not. If the negotiation is *public*, other Users may find and join this negotiation. NOTE The mechanisms for publication and discovery of negotiation is out of the scope of ISO/IEC 23006-4. The default value is *true*, indicating a public negotiation.

Semantics of the `SetupNegotiationContractResponse`:

Name	Definition
`SetupNegotiationContractResponse`	Protocol message sent from the SP to the client with the results of the Setup Negotiation.

Semantics of the `AdmissionContractRequest`:

Name	Definition
`AdmissionContractRequest`	A request from a participant to join the negotiation process.

Semantics of the `AdmissionContractResponse`:

Name	Definition
`AdmissionContractResponse`	Protocol message sent from the SP to the client with the results of the Admission.

Semantics of the `OfferContractRequest`:

Name	Definition
`OfferContractRequest`	Protocol message sent from the client to the SP to provide a Contract offer.

Semantics of the `OfferContractResponse`:

Name	Definition
`OfferContractResponse`	Protocol message sent from the SP to the client with the results of the Offer.

Semantics of the `AgreementContractRequest`:

Name	Definition
`AgreementContractRequest`	Protocol message that specify the Contract Agreement reached.

Semantics of the `AgreementContractResponse`:

Name	Definition
`AgreementContractResponse`	Protocol message sent from the SP to the client with the results of the Agreement.

Semantics of the `AgreementContractRequestType`:

Name	*Definition*	
`AgreementContractRequestType`	`AgreementContractRequestType` `ProtocolRequestType`.	extends

`FinalItem`	The template or final contract exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

Semantics of the `FinalItemContractType`:

Name	*Definition*
`TemplateContract`	The template version of the contract negotiated and agreed.
`FinalContract`	The final and signed version of the contract negotiated and agreed.

Semantics of the `AgreementContractResponseType`:

Name	*Definition*
`AgreementContractResponseType`	`AgreementContractResponseType` extends `ProtocolResponseType`.
`AgreementContractSuccess`	Response in case of success.
`AgreementContractFailure`	Response in case of failure.

Semantics of the `AgreementContractSuccessType`:

Name	*Definition*
`FinalItem`	The template or final contract exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

5.12.2.5 Additional validation rules

5.12.2.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.12.2.5.2 The `NegotiationItemTemplate` of the `SetupNegotiationContractRequest` must be of type `mpegmb:ContractEntityType`.

5.12.2.5.3 The `TemplateEntity` of the `AgreementContractRequest` must be of type `mpegmb:ContractEntityType`.

5.12.2.5.4 The `FinalEntity` of the `AgreementContractRequest` must be of type `mpegmb:ContractEntityType`.

5.12.2.5.5 The `TemplateEntity` of the `AgreementContractResponse` must be of type `mpegmb:ContractEntityType`.

5.12.2.5.6 The `FinalEntity` of the `AgreementContractResponse` must be of type `mpegmb:ContractEntityType`.

5.12.2.6 Extension to SID

For the representation of Service Instance Declarations of *Negotiate Contract*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="NegotiateContractSIDType">
    <complexContent>
      <extension base="sid:NegotiateEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.12.2.7 Example

In this example, we assume that the client wants to make a contract about buying a particular content with any seller, who is willing to agree on the price.

In the first step, the client sends a `SetupNegotiationContractRequest` message to the SP and the SP responds affirmative.

```
<SetupNegotiationContractRequest>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <NegotiationModel href="urn:mpeg:mpegM:cs:03-es-NS:2012:AuctionModelCS:1">
    <Name>English Auction</Name>
  </NegotiationModel>
  <Description>
    <FreeTextAnnotation>I want to buy the right to play "James Bond - Casino
Royal"</FreeTextAnnotation>
  </Description>
  <Owner>
    <UserIdentifier>angelo_seller</UserIdentifier>
  </Owner>
    <NegotiationItemTemplate xsi:type="ContractEntityType">
    <mpegmb:ContractRef>http://example.com/my-contract</mpegmb:ContractRef>
    </NegotiationItemTemplate>
  <IssueOffer>
    <Issue>
      <XPath>//price</XPath>
    </Issue>
    <ProposedValue>100</ProposedValue>
  </IssueOffer>
</SetupNegotiationContractRequest>


<SetupNegotiationContractResponse>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <SetupNegotiationSuccess>
    <NegotiationIdentifier>neg:321</NegotiationIdentifier>
  </SetupNegotiationSuccess>
</SetupNegotiationContractResponse>
```

In the second step, another client sends an `AdmissionContractRequest` message to the SP and the SP responds affirmative.

```
<AdmissionContractRequest>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <NegotiationIdentifier>neg:321</NegotiationIdentifier>
  <Receiver>
    <UserIdentifier>angelo_seller</UserIdentifier>
  </Receiver>
  <Sender>
    <UserIdentifier>sergio_buyer</UserIdentifier>
  </Sender>
</AdmissionContractRequest>

<AdmissionContractResponse>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <AdmissionSuccess/>
</AdmissionContractResponse>
```

In the third step, a client sends an `OfferContractRequest` message to the SP, it forwards to the owner of the negotiation and it responds affirmative via SP.

```
<OfferContractRequest>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <NegotiationIdentifier>neg:321</NegotiationIdentifier>
  <IssueOffer>
    <Issue>
      <XPath>//price</XPath>
    </Issue>
    <ProposedValue>50</ProposedValue>
  </IssueOffer>
  <OfferIdentifier/>
</OfferContractRequest>

<OfferContractResponse>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <OfferSuccess>
  <IssueOffer>
    <Issue>
      <XPath>//price</XPath>
    </Issue>
    <ProposedValue>75</ProposedValue>
  </IssueOffer>
    <OfferIdentifier/>
  </OfferSuccess>
</OfferContractResponse>
```

In the last step the owner of the negotiation communicate an agreement and the owner of the negotiation responds affirmative via SP.

```
<AgreementContractRequest>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <FinalItem>
    <TemplateEntity xsi:type="ContractEntityType">
      <mpegmb:ContractRef>http://example.com/my-final-
contract</mpegmb:ContractRef>
    </TemplateEntity>
```

```
  </FinalItem>
</AgreementContractRequest>

<AgreementContractResponse>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <AgreementContractSuccess>
    <FinalItem>
      <TemplateContract xsi:type="ContractEntityType">
        <mpegmb:ContractRef>http://example.com/my-final-
contract</mpegmb:ContractRef>
      </TemplateContract>
    </FinalItem>
  </AgreementContractSuccess>
</AgreementContractResponse>
```

5.12.3 Negotiate License

5.12.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Negotiate License elementary service.

The Negotiate License elementary service is based on the abstract Negotiate Entity elementary service defined in 5.12.1.

The item of the negotiation is License represented as `mpegmb:LicenseEntityType`.

5.12.3.2 Interfaces and protocol specification

The Negotiate License Protocol extends the abstract Negotiate Entity Protocol defined in 5.12.1.3, from which it inherits the protocol specification.

5.12.3.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Negotiation License Services                    -->
  <!-- ################################################ -->

  <!-- Definition of SetupNegotiationLicenseRequest -->
  <element name="SetupNegotiationLicenseRequest"
type="mpegm:SetupNegotiationLicenseRequestType"/>

  <complexType name="SetupNegotiationLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="NegotiationName" type="string" minOccurs="0"/>
          <element name="NegotiationModel" type="mpeg7:ControlledTermUseType">
            <annotation>
              <documentation xml:lang="en">
                Proposed Classification Scheme: urn:mpeg:mpegM:cs:03-es-
NS:2011:2012:AuctionModelCS
              </documentation>
            </annotation>
          </element>
```

```
          <element name="Description" type="mpeg7:TextAnnotationType"
minOccurs="0"/>
          <element name="Owner" type="mpegmb:UserEntityType"/>
          <element name="Participant" type="mpegmb:UserEntityType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="NegotiationItemTemplate"
type="mpegmb:LicenseEntityType">
            </element>
          <element name="IssueOffer" type="mpegm:IssueOfferType"
maxOccurs="unbounded"/>
        </sequence>
        <attribute name="publish" type="boolean" use="optional" default="true"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="SetupNegotiationLicenseResponse"
type="mpegm:SetupNegotiationEntityResponseType"/>

  <element name="AdmissionLicenseRequest"
type="mpegm:AdmissionEntityRequestType"/>
  <element name="AdmissionLicenseResponse"
type="mpegm:AdmissionEntityResponseType"/>

  <element name="OfferLicenseRequest" type="mpegm:OfferEntityRequestType"/>
  <element name="OfferLicenseResponse" type="mpegm:OfferEntityResponseType"/>

  <element name="AgreementLicenseRequest"
type="mpegm:AgreementLicenseRequestType"/>
  <element name="AgreementLicenseResponse"
type="mpegm:AgreementLicenseResponseType"/>

<!-- Definition of AgreementLicenseRequest -->
  <complexType name="AgreementLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemLicenseType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="FinalItemLicenseType">
    <choice>
      <element name="TemplateLicense" type="mpegmb:LicenseEntityType"/>
      <element name="FinalLicense" type="mpegmb:LicenseEntityType"/>
    </choice>
  </complexType>

  <!-- Definition of AgreementLicenseResponse -->
  <complexType name="AgreementLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="AgreementLicenseSuccess"
type="mpegm:AgreementLicenseSuccessType"/>
          <element name="AgreementLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
```

```
    </complexContent>
  </complexType>
  <complexType name="AgreementLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="FinalItem" type="mpegm:FinalItemLicenseType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.12.3.4 Semantics of protocol data format

Semantics of the `SetupNegotiationLicenseRequest`:

Name	Definition
`SetupNegotiationLicenseRequest`	This message from the owner launches the negotiation using the supplied parameters.

Semantics of the `SetupNegotiationLicenseRequestType`:

Name	*Definition*
`NegotiationName`	The name given to the negotiation.
`NegotiationModel`	The negotiation model that is used for the negotiation. An example of a Classification Scheme is the AuctionModelCS (`urn:mpeg:mpegM:cs:03-es-NS:2012:AuctionModelCS`) defined in G.2.
`Description`	A description of the negotiation.
`Owner`	The negotiation owner. This is typically the party initiating the negotiation.
`Participant`	One or more participants that can be involved exclusively by the Owner in the negotiation.
`NegotiationItemTemplate`	The License under negotiation.
`IssueOffer`	The `NegotiationIssueType` that describes the issues related with the negotiation. All issues that are left blank in the `NegotiationItemTemplate` must be listed.
`publish`	Attribute indicating whether the negotiation shall be *public* (e.g., the SP will be able to advertise it) or not. If the negotiation is *public*, other Users may find and join this negotiation. NOTE The mechanisms for publication and discovery of negotiation is out

of the scope of ISO/IEC 23006-4.

The default value is *true*, indicating a public negotiation.

Semantics of the `SetupNegotiationLicenseResponse`:

Name	Definition
`SetupNegotiationLicenseResponse`	Protocol message sent from the SP to the client with the results of the Setup Negotiation.

Semantics of the `AdmissionLicenseRequest`:

Name	Definition
`AdmissionLicenseRequest`	A request from a participant to join the negotiation process.

Semantics of the `AdmissionLicenseResponse`:

Name	Definition
`AdmissionLicenseResponse`	Protocol message sent from the SP to the client with the results of the Admission.

Semantics of the `OfferLicenseRequest`:

Name	Definition
`OfferLicenseRequest`	Protocol message sent from the client to the SP to provide a License offer.

Semantics of the `OfferLicenseResponse`:

Name	Definition
`OfferLicenseResponse`	Protocol message sent from the SP to the client with the results of the Offer.

Semantics of the `AgreementLicenseRequest`:

Name	Definition
`AgreementLicenseRequest`	Protocol message that specify the License Agreement reached.

Semantics of the `AgreementLicenseResponse`:

Name	Definition
`AgreementLicenseResponse`	Protocol message sent from the SP to the client with the results of the Agreement.

Semantics of the `AgreementLicenseRequestType`:

Name	*Definition*
`AgreementLicenseRequestType`	`AgreementLicenseRequestType` extends `ProtocolRequestType`.
`FinalItem`	The template or final license exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

Semantics of the `FinalItemLicenseType`:

Name	*Definition*
`TemplateLicense`	The template version of the license negotiated and agreed.
`FinalLicense`	The final and signed version of the license negotiated and agreed.

Semantics of the `AgreementLicenseResponseType`:

Name	*Definition*
`AgreementLicenseResponseType`	`AgreementLicenseResponseType` extends `ProtocolResponseType`.
`AgreementLicenseSuccess`	Response in case of success.
`AgreementLicenseFailure`	Response in case of failure.

Semantics of the `AgreementLicenseSuccessType`:

Name	*Definition*
`FinalItem`	The template or final license exchanged between parties. Usually it is the Owner that accepts all the negotiated issues provided by a Participant via SP.

5.12.3.4.1 Additional validation rules

5.12.3.4.1.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.12.3.4.1.2 The `NegotiationItemTemplate` of the `SetupNegotiationLicenseRequest` must be of type `mpegmb:LicenseEntityType`.

5.12.3.4.1.3 The `TemplateEntity` of the `AgreementLicenseRequest` must be of type `mpegmb:LicenseEntityType`.

5.12.3.4.1.4 The `FinalEntity` of the `AgreementLicenseRequest` must be of type `mpegmb:LicenseEntityType`.

5.12.3.4.1.5 The `TemplateEntity` of the `AgreementLicenseResponse` must be of type `mpegmb:LicenseEntityType`.

5.12.3.4.1.6 The `FinalEntity` of the `AgreementLicenseResponse` must be of type `mpegmb:LicenseEntityType`.

5.12.3.5 Extension to SID

For the representation of Service Instance Declarations of *Negotiate License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="NegotiateLicenseSIDType">
  <complexContent>
    <extension base="sid:NegotiateEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.13 The Package Services

5.13.1 Package Content

5.13.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Package Content Protocol.

This Elementary Service enables a User to prepare a Digital Item for Delivery. The Digital Item can, for example, be provided as the result of the Create Content or Process Content Services. This elementary service is in charge of fragmenting the Digital Item into several parts and binding them to a specific transport protocol.

The result is a set of data ready to be delivered in the form of file or stream according to given specifications. Moreover the protocol can also return a skeleton of a DI, without fragments.

5.13.1.2 Interfaces and protocol specification

The Package Content Protocol is as follows:

Steps	Client	Service Provider
1.	A User sends a `PackageContentRequest` message. This message should contain the information on which part of the DI should be packaged (even the DI itself) and how this packaging should be done.	
2.		The SP sends back a message containing the reference to the package items using a `PackageContentResponse`.

5.13.1.3 Syntax of protocol data format

```
<!-- ################################################ -->
  <!--  Package Content                                 -->
  <!-- ################################################ -->
  <!-- Definition of PackageContentRequest -->
  <element name="PackageContentRequest" type="mpegm:PackageContentRequestType"/>
  <complexType name="PackageContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
          <element name="Package" type="mpegm:PackageType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="PackageType">
    <sequence>
      <element name="PackageFormat" type="mpegm:PackageFormatType"
minOccurs="0"/>
      <choice>
        <sequence>
          <element name="Fragment" type="mpegmb:ContentEntityType"/>
          <element name="SupportedTransferProtocol"
type="mpegmb:SupportedTransferProtocolType"/>
        </sequence>
        <element name="BBLDoc" type="bbl:BBLDocType"/>
      </choice>
    </sequence>
  </complexType>
  <simpleType name="PackageFormatType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="STREAM"/>
          <enumeration value="FILE"/>
        </restriction>
      </simpleType>
```

```
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>

  <!-- Definition of PackageContentResponseType -->
  <element name="PackageContentResponse"
type="mpegm:PackageContentResponseType"/>
  <complexType name="PackageContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="PackageContentSuccess"
type="mpegm:PackageContentSuccessType"/>
          <element name="PackageContentFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="PackageContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.13.1.4 Semantics of protocol data format

Semantics of the `PackageContentRequest`:

Name	*Definition*
`PackageContentRequest`	The message for creating a request for PackageContent.
`PackageContentRequestType`	Top-level type for `PackageContentRequest`. `PackageContentRequestType` extends `ProtocolRequestType`.
`ContentEntity`	A Digital Item or a reference to it.
`Package`	The information about the packaging of one or more parts of the DI.

Semantics of the `PackageType`:

Name	*Definition*
`PackageFormat`	Indicates the data representation of the packaged content. The `PackageFormat` can be STREAM or FILE. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.
`FragmentID`	Fragment identifier within the Digital Item.
`SupportedTransferProtocol`	Transfer protocol for delivery.
`BBLDoc`	A BBL document describing the fragmentation and binding.

Semantics of the `PackageContentResponse`:

Name	*Definition*
`PackageContentResponse`	The response contains the result of the PackageContent Service, it could be successful or not. In the first case it is a set of references to one or more streams or files packaged. It may also contain a DI skeleton.
`PackageContentResponseType`	Top-level type for `PackageContentResponse`. `PackageContentResponseType` extends `ProtocolResponseType`.
`PackageContentSuccess`	Response in case of success.
`PackageContentFailure`	Response in case of failure.

Semantics of the `PackageContentSuccessType`:

Name	*Definition*
`PackageContentSuccessType`	Type of the response message part that is provided in case of success. `PackageContentSuccessType` extends `ProtocolSuccessType`.
`ContentEntity`	The packaged Digital Item, possibly fragmented over multiple `ContentEntity` elements.

5.13.1.5 Extension to SID

For the representation of Service Instance Declarations of *Package Content*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->
<!-- Definition of PackageContentSIDType -->
<complexType name="PackageContentSIDType">
```

```
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="SupportedTransferProtocol"
type="mpegmb:SupportedTransferProtocolType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:PackageContentSIDType`:

Name	*Definition*
`sid:PackageContentSIDType`	Top-level type for SIDs of the *Package Content* regular ES. `sid:PackageContentSIDType` extends `sid:ServiceInstanceDeclarationType`.
`sid:SupportedTransferProtocol`	Lists supported transfer protocols for which the SP can package the content.

5.13.1.6 Example

A User wants to have a content packaged in several parts in order to obtain several references. These can be reference resources to be delivered in different time and ways.

```
<PackageContentRequest>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
  <ContentEntity>
    <dii:Identifier>urn:mpegRA:mpeg21:dii:doi:10.1000/123456789</dii:Identifier>
  </ContentEntity>
  <Package>
    <Fragment>
      <didl:DIDL>
        <didl:Item>
          <!-- DI goes here -->
        </didl:Item>
      </didl:DIDL>
    </Fragment>
    <SupportedTransferProtocol>
      <mpegmb:TransferProtocol href="urn:mpeg:mpegM:cs:03-es-
NS:2012:TransferProtocolCS:FTP"/>
    </SupportedTransferProtocol>
  </Package>
</PackageContentRequest>
```

The SP will elaborate data to provide the packaged part of the given content. In response the SP will return a digital item (this digital item should be different from the one given in the `PackageContentRequest` message, some parts may be removed because of packaging) and a list of references to the packaged items.

```
<PackageContentResponse>
  <mpegmb:TransactionIdentifier>trans:123</mpegmb:TransactionIdentifier>
    <PackageContentSuccess>
```

```
      <ContentEntity>
        <didl:DIDL>
          <didl:Container><!-- DI goes here --></didl:Container>
        </didl:DIDL>
      </ContentEntity>
      <ContentEntity>
        <mpegmb:ContentRef>ftp://example.com/Segment1.mp4</mpegmb:ContentRef>
      </ContentEntity>
      <ContentEntity>
        <mpegmb:ContentRef>ftp://example.com/Segment2.mp4</mpegmb:ContentRef>
      </ContentEntity>
    </PackageContentSuccess>
</PackageContentResponse>
```

5.14 The Post Services

5.14.1 Post Content

5.14.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Post Content elementary service.

This Elementary Service enables Users to make other Users aware of their Content. The purpose of the Post Content Elementary Service is to make Content discoverable by relying it to a local or remote Post Content SP, which serves as a gateway to a Content distribution network taking responsibility of circulating information about the Content inside the network.

The Post Content ES supports content-centric distribution paradigms such as (but not limited to) publish/subscribe, P2P, cloud and anycast.

5.14.1.2 Interfaces and protocol specification

The Post Content Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `PostContentRequest` message. This message includes a reference to the Content.	
2.		The Post Content Service Provider replies with a Success/Failure message.

5.14.1.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!-- ################################################ -->
<!--  Post Services                                   -->
<!-- ################################################ -->
<!-- ################################################ -->
<!--  Post Content                                    -->
<!-- ################################################ -->
<!-- Definition of PostContentRequest -->
<element name="PostContentRequest" type="mpegm:PostContentRequestType"/>
    <complexType name="PostContentRequestType">
      <complexContent>
        <extension base="mpegmb:ProtocolRequestType">
          <sequence>
            <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <!-- Definition of PostContentResponse -->
    <element name="PostContentResponse" type="mpegm:PostContentResponseType"/>
    <complexType name="PostContentResponseType">
      <complexContent>
        <extension base="mpegmb:ProtocolResponseType">
          <choice>
            <element name="PostContentSuccess"
type="mpegmb:ProtocolSuccessType"/>
            <element name="PostContentFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </extension>
      </complexContent>
    </complexType>
```

5.14.1.4 Semantics of protocol data format

Semantics of the `PostContentRequest`:

Name	*Definition*
`PostContentRequest`	The request to Post Content.

Semantics of the `PostContentResponse`:

Name	*Definition*
`PostContentResponse`	The response indicating whether the Content has been successfully posted or not.
`PostContentSuccess`	Response in case of success.
`PostContentFailure`	Response in case of failure.

5.14.1.5 Extension to SID

For the representation of Service Instance Declarations of Post Content, the following complex type is defined in the SID XML Schema:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="PostContentSIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.14.1.6 Example

This example shows the run of the Post Content Protocol. The client sends a `PostContentRequest` message to (possibly a locally running) SP, requesting that a specified Content be circulated and made discoverable to other users using the network. Other users may be notified of the Content if they had expressed an interest in it and visibility of Content is regulated by Licenses embedded in it.

```
<PostContentRequest>
<mpegmb:TransactionIdentifier>123</mpegmb:TransactionIdentifier>
  <mpegmb:Timestamp>2011-12-31T12:00:00</mpegmb:Timestamp>
  <ContentEntity>
    <didl:DIDL DIDLDocumentId="http://example.org/idvalue1">
      <didl:Container id="idvalue11"/>
    </didl:DIDL>
  </ContentEntity>
</PostContentRequest>
```

The SP responds with a `PostContentResponse` message.

```
<PostContentResponse>
<mpegmb:TransactionIdentifier>123</mpegmb:TransactionIdentifier>
  <PostContentSuccess/>
</PostContentResponse>
```

5.15 The Present Services

5.15.1 Present Entity

5.15.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Present Entity Service. The protocol of this Elementary Service has to be used as a guide for any extra Presen Service that may be specified, apart from the ones included in the following Sections.

Given that MPEG-M handles XML data structures, this Elementary Service provides the means to present an Entity using a W3C XSLT transformation to be applied on the XML document. The Service Provider will apply the transformation and will return the HTML document, which shall be conformant to the XHTML specification.

5.15.1.2 Interfaces and protocol specification

The Present Entity Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `PresentEntityRequest` message. This message includes the Entity, plus the transformations that are to be done. If no transformations are specified, a default transformation will be done.	
2.		The Service Provider providing the Present Entity Service sends back a message `PresentEntityResponse` with the XHTML version of the Entity.

5.15.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Present Entity                                  -->
  <!-- ################################################ -->
  <!-- Definition of PresentEntityRequest -->
  <complexType name="PresentEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="PresentationEntity" type="mpegmb:EntityBaseType"/>
          <element ref="xsl:transform" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of PresentEntityResponse -->
  <complexType name="PresentEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="PresentEntitySuccess"
type="mpegm:PresentEntitySuccessType"/>
          <element name="PresentEntityFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="PresentEntitySuccessType">
    <sequence>
      <element ref="xhtml:html"/>
    </sequence>
  </complexType>
```

5.15.1.4 Semantics of protocol data format

Semantics of the `PresentEntityRequest`:

Name	*Definition*
`PresentEntityRequest`	The request to Present an Entity.
`PresentationEntity`	The entity to be presented.
`xsl:transform`	An XSLT transformation to be applied in the Entity.

Semantics of the `PresentEntityResponse`:

Name	*Definition*
`PresentEntityResponse`	The response to Present an Entity.
`PresentEntitySuccess`	Response in case of success.
`PresentEntityFailure`	Response in case of failure.

Semantics of the `PresentEntitySuccessType`:

Name	*Definition*
`PresentEntitySuccessType`	Response in case of success.
`xhtml:html`	The document presented, given as an HTML.

5.15.1.5 Extension to SID

For the representation of Service Instance Declarations of Present Entity, the following complex type is defined in the SID XML Schema:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="PresentEntitySIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.15.2 Present Contract

5.15.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Present Contract Service.

This Elementary Service enables a User to format a Contract for its presentation. The Contract shall be given along with a W3C XSLT transformation to be applied on the XML contract. The Service Provider will apply the transformation (e.g., transforming a `cel:title` element into a `<h1>` HTML element), will present the contract, and will return the HTML document, which shall be conformant to the XHTML specification.

5.15.2.2 Interfaces and protocol specification

The Present Contract Protocol extends the abstract Present Entity Protocol defined in 5.15.1.2, from which it inherits the protocol specification.

5.15.2.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Present Contract                                -->
  <!-- ################################################ -->
  <!-- Definition of PresentContractRequest -->
  <element name="PresentContractRequest"
type="mpegm:PresentContractRequestType"/>
  <complexType name="PresentContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="PresentationContract" type="mpegmb:ContractEntityType"/>
          <element ref="xsl:transform" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of PresentContractResponse -->
  <element name="PresentContractResponse"
type="mpegm:PresentEntityResponseType"/>
```

5.15.2.4 Semantics of protocol data format

Semantics of the `PresentContractRequest`:

Name	*Definition*
`PresentContractRequest`	The request to Present a Contract.
`PresentationContract`	A MPEG-21 CEL contract as described in 5.2.3.10.
`xsl:transform`	An XSLT transformation to be applied in the contract.

Semantics of the `PresentContractResponse`:

Name	*Definition*
`PresentContractResponse`	The response to Present a Contract.

5.15.2.5 Extension to SID

For the representation of Service Instance Declarations of *Present Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="PresentContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.15.3 Present License

5.15.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Present License Service.

This Elementary Service enables a User to format a License for its presentation. The License shall be given along with a W3C XSLT transformation to be applied on the XML contract. The Service Provider will apply the transformation (e.g., transforming a `r:license` element into a `<h1>` HTML element), will present the license, and will return the HTML document, which shall be conformant to the XHTML specification.

5.15.3.2 Interfaces and protocol specification

The Present License Protocol extends the abstract Present Entity Protocol defined in 5.15.1.2, from which it inherits the protocol specification.

5.15.3.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Present License                                 -->
<!-- ################################################ -->
<!-- Definition of PresentLicenseRequest -->
<element name="PresentLicenseRequest" type="mpegm:PresentLicenseRequestType"/>
<complexType name="PresentLicenseRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="PresentationLicense" type="mpegmb:LicenseEntityType"/>
        <element ref="xsl:transform" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Definition of PresentLicenseResponse -->
<element name="PresentLicenseResponse" type="mpegm:PresentEntityResponseType"/>
```

5.15.3.4 Semantics of protocol data format

Semantics of the `PresentLicenseRequest`:

Name	*Definition*
`PresentLicenseRequest`	The request to Present a License.
`PresentationLicense`	A MPEG-21 REL license as described in 5.2.3.12.
`xsl:transform`	An XSLT transformation to be applied in the contract.

Semantics of the `PresentLicenseResponse`:

Name	*Definition*
`PresentLicenseResponse`	The response to Present a License.

5.15.3.5 Extension to SID

For the representation of Service Instance Declarations of *Present License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="PresentLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```


5.16 The Process Services

5.16.1 Process Content

5.16.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Process Content elementary service.

This Elementary Service enables a User to alter a Digital Item or the resources within a Digital Item.

The new Digital Item may express the relationship it maintains with the original Digital Item via a `dii:RelatedIdentifier` element (see B.8.5 and Annex F), or a similar semantic linking statement, which carries the Identifier of the original Digital Item and expresses the nature of the relationship.

The Processing of Content may comprise, but is not limited to the following Service Types (see Annex B):

— Speech recognition

— Speech synthesis

— Language processing

— Language translation

— Extraction of Sensory Information

— Adaptation of Digital Items

— Transcoding of resources of Digital Items

— Stream repurposing

A Service Type is the binding of a specific purpose (specified through extensions of `RequestParametersBaseType` and optionally `CompletionParametersBaseType` as well as their associated semantics) to a generic Elementary Service (i.e., Process Content). Note that such a generic Service cannot be implemented by a Service Instance Declaration without a corresponding Service Type. Furthermore, such a generic Service cannot be part of an Aggregated Service without a corresponding Service Type.

The Service Type can be implemented by a Service Instance Declaration in the same way as other Services. The Service Type can be part of an Aggregated Service in the same way as other Services.

Each Service Type for Process Content specifies its own complex type for the `RequestParameters` element that conveys the additional parameters for the processing. Optionally, it may also specify a complex type for the `CompletionParameters` element that conveys additional parameters of the result of the processing. The `RequestParameters` must specify a complex type that extends the `RequestParametersBaseType` abstract base type defined in 5.2.3.15. Similarly, the `CompletionParameters` must specify a complex type that extends the `CompletionParametersBaseType` abstract base type defined in 5.2.3.16. The complex type for the `RequestParameters` element and optionally the complex type for the `CompletionParameters` element are specified in the Service Instance Declaration of each Process Content Service Instance.

Annex B provides a non-exhaustive list of Service Types for Process Content. Further examples of Service Types are provided in Annex C of ISO/IEC 23006-5.

5.16.1.2 Interfaces and protocol specification

The Process Content Protocol is as follows:

Steps	Client	Service Provider
1.	The client provides the specified parameters and sends a `ProcessContentRequest` message, to the SP.	
2.		The SP applies the supplied parameters as configuration to the Processing Engine and starts the processing. If the client has requested an immediate response message, the SP continues with step 3. Otherwise, if the client has not requested an immediate response message, the SP continues with step 4.

Steps	Client	Service Provider
3.		*(optional)* The SP sends a `ProcessContentResponse` message to the client, indicating whether the processing can be performed. If the processing cannot be performed, the protocol ends here.
4.		After the processing has finished, the SP sends a `ProcessContentCompletion` message to the client. Depending on the type of processing, the message may contain a Digital Item as the result.

The `TransactionIdentifier` remains the same over the entire protocol run.

In addition to the specified protocol, the client may also apply the session control messages defined in 5.4.1 to pause, resume, or stop the processing.

The client may also apply the session status polling message defined in 5.4.2 to repeatedly poll for the status of the processing.

5.16.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Process Content                                 -->
  <!-- ################################################ -->
  <!-- Definition of ProcessContentRequest -->
  <element name="ProcessContentRequest" type="mpegm:ProcessContentRequestType"/>
  <complexType name="ProcessContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ServiceTypeEntity" type="mpegm:ServiceTypeEntityType"/>
          <element name="ContentEntity" type="mpegm:ContentEntityType"
minOccurs="0"/>
          <element name="RequestParameters"
type="mpegmb:RequestParametersBaseType">
            <annotation>
              <documentation xml:lang="en">
                Type of parameters must correspond to the RequestParametersType
specified in the ServiceTypeEntity.
              </documentation>
            </annotation>
          </element>
        </sequence>
        <attribute name="immediateResponse" type="boolean" use="optional"
default="true"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of ProcessContentResponse -->
  <element name="ProcessContentResponse"
type="mpegm:ProcessContentResponseType"/>
  <complexType name="ProcessContentResponseType">
```

```
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="ProcessContentSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <!-- No additional elements required. So no Type declaration needed. --
>
          <element name="ProcessContentFailure"
type="mpegmb:ProtocolFailureType"/>
          <!-- No additional elements required. So no Type declaration needed. --
>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of ProcessContentCompletion -->
  <element name="ProcessContentCompletion"
type="mpegm:ProcessContentCompletionType"/>
  <complexType name="ProcessContentCompletionType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="ProcessContentCompletionSuccess"
type="mpegm:ProcessContentCompletionSuccessType"/>
          <element name="ProcessContentCompletionFailure"
type="mpegmb:ProtocolFailureType"/>
          <!-- No additional elements required. So no Type declaration needed. --
>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of ProcessContentCompletionSuccessType -->
  <complexType name="ProcessContentCompletionSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ProcessedContentEntity" type="mpegm:ContentEntityType"
minOccurs="0"/>
          <element name="CompletionParameters"
type="mpegmb:CompletionParametersBaseType" minOccurs="0">
            <annotation>
              <documentation xml:lang="en">
                Type of parameters must correspond to the
CompletionParametersType specified in the ServiceTypeEntity of the
ProcessContentRequest message.
              </documentation>
            </annotation>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.16.1.4 Semantics of protocol data format

Semantics of the `ProcessContentRequest`:

Name	*Definition*
`ProcessContentRequest`	Protocol message for instructing the SP to process the content.
`ProcessContentRequestType`	Top-level type for `ProcessContentRequest`. `ProcessContentRequestType` extends `ProtocolRequestType`.
`ServiceTypeEntity`	Specifies the Service Type of this Process Content protocol run.
`ContentEntity`	Either a Digital Item Declaration of the content to be processed or a URI referring to it. NOTE If a specific part of a Digital Item is needed, this can be specified in the `RequestParameters` e.g., through Fragment Identifiers as specified in ISO/IEC 21000-17.
`RequestParameters`	Service Type specific parameters for the processing.
`immidiateResponse`	Indicates whether the SP has to respond with a `ProcessContentResponse` message. If set to "false", the SP will only send the `ProcessContentCompletion` message after the processing has finished.

Semantics of the `ProcessContentResponse`:

Name	*Definition*
`ProcessContentResponse`	Protocol message sent from the SP to the client in response to the `ProcessContentRequest` message.
`ProcessContentResponseType`	Top-level type for `ProcessContentResponse`. `ProcessContentResponseType` extends `ProtocolResponseType`.
`ProcessContentSuccess`	Response in case of success.
`ProcessContentFailure`	Response in case of failure.

Semantics of the `ProcessContentCompletion`:

Name	*Definition*
`ProcessContentCompletion`	Protocol message sent from the SP to the client after the processing has been completed.
`ProcessContentCompletionType`	Top-level type for `ProcessContentCompletion`. `ProcessContentCompletionType` extends `ProtocolResponseType`.

`ProcessContentCompletionSuccess`	Message part in case of success. If a `SuccessCode` element is present, it should be set to "ACCEPTED".
`ProcessContentCompletionFailure`	Message part in case of failure.

Semantics of the `ProcessContentCompletionSuccessType`:

Name	*Definition*
`ProcessContentCompletionSuccessType`	Type of the response message part that is provided in case of success.
`ProcessedContentEntity`	The processed Content if available. It may carry a dii:RelatedIdentifier element (see B.8.5 and Annex F) or a similar semantic linking statement.
`CompletionParameters`	Service Type specific parameters for the result of the processing.

5.16.1.5 Additional validation rules

5.16.1.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.16.1.5.2 The Service Type specified by the `ServiceTypeEntity` must have the same Service Type Identifier as the one specified in the underlying Service Instance Declaration.

5.16.1.5.3 The type of `RequestParameters` must conform to the type specified by the `RequestParametersType` element of the Service Type for that particular Process Content Service Instance.

5.16.1.5.4 The type of `CompletionParameters` must conform to the type specified by the `CompletionParametersType` element of the Service Type for that particular Process Content Service Instance.

5.16.1.6 Extension to SID

For the representation of Service Instance Declarations of *Process Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="ProcessContentSIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="ServiceTypeEntity" type="mpegmb:ServiceTypeEntityType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
```

```
    </complexContent>
  </complexType>
```

Semantics of the `sid:ProcessContentSIDType`:

Name	*Definition*
`sid:ProcessContentSIDType`	Top-level type for SIDs of the *Process Content* specific ES. `sid:ProcessContentSIDType` extends `sid:ServiceInstanceDeclarationType`.
`sid:ServiceTypeEntity`	The Service Type of this Service Instance, typically included by identifier.

5.16.2 Process License

5.16.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Process License elementary service.

This Elementary Service enables a User to create a License starting from an existing License.

Possible operations can be:

— add or change a resource related to the license

— create a license template starting from another license

— re-issue a license with a new license principal starting from a template

5.16.2.2 Interfaces and protocol specification

The Process License Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `ProcessLicenseRequest` message. This message is used to create a license from a template (`TemplateGroupType`) or reissue a license (`ReissueGroupType`).	
2.		The server sends a `ProcessLicenseResponse`, if the operation ends successfully, it conveys the updated License.

5.16.2.3 Syntax of protocol data format

```
  <!-- ################################################################ -->
  <!--  Process License                                                 -->
  <!-- ################################################################ -->
  <!-- Definition of ProcessLicenseRequest -->
  <element name="ProcessLicenseRequest" type="mpegm:ProcessLicenseRequestType"/>
  <complexType name="ProcessLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
          <element ref="rel-r:resource" minOccurs="0"/>
          <element name="ProcessLicenseParameter"
type="mpegm:ProcessLicenseParameterType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of ProcessLicenseParameterType -->
  <complexType name="ProcessLicenseParameterType">
    <choice>
      <element name="TemplateGroup" type="mpegm:TemplateGroupType"/>
      <element name="ReissueGroup" type="mpegm:ReissueGroupType"/>
    </choice>
  </complexType>
  <!-- Definition of TemplateGroupType -->
  <complexType name="TemplateGroupType">
    <sequence>
      <element name="Condition" type="rel-r:Condition"/>
      <element name="Issuer" type="rel-r:Issuer"/>
    </sequence>
  </complexType>
  <!-- Definition of ReissueGroupType -->
  <complexType name="ReissueGroupType">
    <sequence>
      <element name="Principal" type="rel-r:Principal"/>
      <element name="LicenseRegistration" type="rel-r:License" minOccurs="0"/>
    </sequence>
  </complexType>

<!-- Definition of ProcessLicenseResponse -->
<element name="ProcessLicenseResponse" type="mpegm:ProcessLicenseResponseType"/>
  <complexType name="ProcessLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="ProcessLicenseSuccess"
type="mpegm:ProcessLicenseSuccessType"/>
          <element name="ProcessLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="ProcessLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
```

```
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.16.2.4 Semantics of protocol data format

Semantics of the `ProcessLicenseRequest`:

Name	*Definition*
`ProcessLicenseRequest`	The message for creating a request for ProcessLicense.
`ProcessLicenseRequestType`	Top-level type for `ProcessLicenseRequest`. `ProcessLicenseRequestType` extends `ProtocolRequestType`.
`License`	A REL License.
`rel-r:resource`	Resource reference.
`ProcessLicenseParameter`	Parameter to process the license.

Semantics of the `ProcessLicenseParameterType`:

Name	*Definition*
`ProcessLicenseParameterType`	`ProcessLicenseParameterType` extends `ProtocolBaseType`. It contains the parameter for the operation to be performed.
`TemplateGroup`	The TemplateGroup is useful for creating a license template from an existing license.
`ReissueGroup`	The `ReissueGroup` element is useful for making a new license starting from an existing license template.

Semantics of the `TemplateGroupType`:

Name	*Definition*
`TemplateGroupType`	The TemplateGroup allows creating a license template from an existing license.
`Condition`	A new condition to be added to the new license template.
`Issuer`	The new issuer for the new license template.

Semantics of the `ReissueGroupType`:

Name	*Definition*
`ReissueGroupType`	The `ReissueGroupType` is useful for making a new license starting from an existing license template.
`Principal`	The new principal for the new license; this is necessary if the license template contains a `forAll` element, which defines in the Template license a set of possible Principals having similar peculiarities.
`LicenseRegistration`	A REL license specifying the affiliation to a specific set of users. It will be used to match the possible Principals, which can receive the re-issued license. In order to successfully complete this operation, the identifier of this set of users must be the same of the one contained in the `forAll` element of the template license. If the `forAll` element is not present in the template license, this matching is not necessary.

Semantics of the `ProcessLicenseResponse`:

Name	*Definition*
`ProcessLicenseResponse`	Response message for processing a license. If the request message contains the `TemplateGroup`, the success response message shall contain the template license. Otherwise, if the request message contains the `ReissueGroup`, the success response message shall be a newly issued license.
`ProcessLicenseResponseType`	Top-level type for `ProcessLicenseResponse`. `ProcessLicenseResponseType` extends `ProtocolResponseType`.
`ProcessLicenseSuccess`	Response in case of success.
`ProcessLicenseFailure`	Response in case of failure.

Semantics of the `ProcessLicenseSuccessType`:

Name	*Definition*
`ProcessLicenseSuccessType`	`ProcessLicenseSuccessType` extends `ProtocolSuccessType`.
`LicenseEntity`	Processed license or a reference to it.

5.16.2.5 Extension to SID

For the representation of Service Instance Declarations of *Process License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="ProcessLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.17 The Request Services

5.17.1 Request Content

5.17.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Request Content Protocol.

This Elementary Service enables a User to gain access to a Digital Item or any of its components.

Before requesting the content, the User may optionally also request metadata about the content. As there are different metadata schemes, the request message should indicate the needed scheme. In multimedia systems (e.g., IPTV), metadata usually is organized in sections. Each section can be identified either by one or more identification level (e.g., provider's name, type of offering) or by a URI. Further, a section may have a version attribute for the purpose of updating metadata between different parties. Identification levels of a metadata scheme could be described using a Classification Scheme (CS).

5.17.1.2 Interfaces and protocol specification

The Request Content Protocol is as follows:

Steps	Client	Service Provider
1.	If the client needs to fetch metadata about the content before requesting the content, then it follows steps 2 to 5, otherwise it goes directly to step 6.	
2.	(optional) The client sends to the SP a `RequestMetadataRequest` message specifying the information (including metadata scheme, identification, and possibly the current version value) of the metadata section to be requested.	
3.		(optional) In case the request can be satisfied, the SP generates a `RequestMetadataResponse` message: - If no `version` value is specified or the version value of the requested section in the SP's database of SP is higher than that specified in the `RequestMetadataRequest` message, the content of requested metadata section will be included in the

Steps	Client	Service Provider
		RequestMetadataResponse message. If the requested metadata section at SP side is not newer than the current copy at the User side (indicated by the value of the version attribute), then no metadata section will be included in the RequestMetadataResponse message and the SuccessCode of the RequestMetadataSuccess element shall be set to "NO_CONTENT". - In case the request cannot be satisfied, the RequestMetadataFailure element conveys the information related to the reason of failure.
4.		(optional) The SP sends the RequestMetadataResponse message to the client.
5.	(optional) The client retrieves the metadata contained in the MetadataSection element or from the location specified by MetadataURL element of the RequestMetadataResponse.	
6.	(optional) The client sends a RequestContentRequest message. The message needs at least a ContentRef.	
7.		(optional) The SP sends a RequestContentResponse that may contain a digital item or a reference to it.

5.17.1.3 Syntax of protocol data format

```
<!-- ################################################################ -->
<!--  Request Content                                                 -->
<!-- ################################################################ -->

  <!-- ############################################### -->
  <!--          Optional Request Metadata steps        -->
  <!-- ############################################### -->
  <!-- Definition of RequestMetadataRequest -->
  <element name="RequestMetadataRequest"
type="mpegm:RequestMetadataRequestType"/>
  <complexType name="RequestMetadataRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="SupportedMetadataSchema"
type="mpegmb:SupportedMetadataSchemaType" minOccurs="0"/>
          <choice>
            <element name="SectionCriterion" type="mpegm:SectionCriterionType"
maxOccurs="unbounded"/>
```

```
            <element name="MetadataSectionURI" type="anyURI"/>
          </choice>
        </sequence>
        <attribute name="version" type="string" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of SectionCriterionType -->
  <complexType name="SectionCriterionType">
    <sequence>
      <element name="SectionKind" type="mpeg7:ControlledTermUseType"/>
      <element name="SectionIdentification"
type="mpegm:SectionIdentificationType" minOccurs="0"/>
    </sequence>
  </complexType>

  <!-- Definition of SectionIdentificationType -->
  <complexType name="SectionIdentificationType">
    <choice>
      <element name="NumericID" type="int"/>
      <element name="TextualID" type="string"/>
    </choice>
  </complexType>

  <!-- Definition of RequestMetadataResponse -->
  <element name="RequestMetadataResponse"
type="mpegm:RequestMetadataResponseType"/>
  <complexType name="RequestMetadataResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="RequestMetadataSuccess"
type="mpegm:RequestMetadataSuccessType"/>
          <element name="RequestMetadataFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of RequestMetadataSuccessType -->
  <complexType name="RequestMetadataSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="MetadataSection" type="mpegm:MetadataSectionType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of MetadataSectionType -->
  <complexType name="MetadataSectionType">
    <choice>
      <element name="MetadataSectionContent">
        <complexType>
          <sequence>
            <any namespace="##any" processContents="skip"/>
```

```
          </sequence>
        </complexType>
      </element>
      <element name="MetadataSectionURI" type="anyURI"/>
    </choice>
    <attribute name="version" type="string" use="optional"/>
  </complexType>


  <!-- ################################################ -->
  <!--          Actual Request Content steps            -->
  <!-- ################################################ -->
<!-- Definition of RequestContentRequest -->
<element name="RequestContentRequest" type="mpegm:RequestContentRequestType"/>
<complexType name="RequestContentRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
        <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
        <element name="UsageEnvironmentDescription"
type="dia:UsageEnvironmentType" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Definition of RequestContentResponse -->
<element name="RequestContentRespone" type="mpegm:RequestContentResponseType"/>
<complexType name="RequestContentResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
      <sequence>
        <choice>
          <element name="RequestContentSuccess"
type="mpegm:RequestContentSuccessType"/>
          <element name="RequestContentFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="RequestContentSuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolSuccessType">
      <sequence>
        <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```


5.17.1.4 Semantics of protocol data format

Semantics of the `RequestMetadataRequest`:

Name	*Definition*
`RequestMetadataRequest`	Protocol message sent from the User to the SP to request some metadata section.
`RequestMetadataRequestType`	Top-level type for `RequestMetadataRequest`. `RequestMetadataRequestType` extends `ProtocolRequestType`.
`SupportedMetadataSchema`	Indicates the metadata schema representation of requested metadata.
`SectionCriterion`	Indicates a criterion (i.e., identifier of the section level) of the metadata section being requested. The identifier of a higher level will appear before that of a lower level. The identifiers of all levels will be combined to determine the requested section.
`MetadataSectionURI`	Describes the URI of a requested metadata section. This URI may contain an XPath addressing a part of an XML document.
`version`	Indicates the current version of the metadata section that the client already has.

Semantics of the `SectionCriterionType`:

Name	*Definition*
`SectionCriterionType`	Top-level type for describing the conditions of a requested metadata section.
`SectionKind`	Indicates a kind of metadata section. Classification Schemes that could be used for this purpose are provided in A.2 and A.3.
`SectionIdentification`	Indicates the identification of a metadata section. The type and format of this element depends on the employed metadata schema.

Semantics of the `SectionIdentificationType`:

Name	*Definition*
`SectionIdentificationType`	Top-level type for describing the identification of a requested metadata section. NOTE The definition of a Classification Scheme term typically indicates whether to `NumericID` or `TextualID`.
`NumericID`	Indicates a numeric value as the identification of a metadata section.
`TextualID`	Indicates a textual string as the identification of a metadata section.

Semantics of the `RequestMetadataResponse`:

Name	*Definition*
`RequestMetadataResponse`	Protocol message sent from the SP to the User in response to a `RequestMetadataRequest` message to return a requested metadata section.
`RequestMetadataResponseType`	Top-level type for `RequestMetadataResponse`. `RequestMetadataResponseType` extends `ProtocolResponseType`.
`RequestMetadataSuccess`	Response in case of success. It may convey a requested metadata section.
`RequestMetadataFailure`	Response in case of failure.

Semantics of the `RequestMetadataSuccessType`:

Name	*Definition*
`RequestMetadataSuccessType`	`RequestMetadataSuccessType` extends `ProtocolSuccessType`.
`MetadataSection`	Conveys a requested metadata section. If the `version` attribute of the `RequestMetadataRequest` message matches the current metadata section version of the SP (i.e., the client already has the latest version of this metadata section), then this element will be omitted.

Semantics of the `MetadataSectionType`:

Name	*Definition*
`MetadataSectionType`	Top-level type for conveying a metadata section replied by a SP.
`MetadataSectionContent`	Contains a metadata section conveyed within the response message.
`MetadataSectionURI`	Specifying the URL from where the requested metadata section can be obtained. NOTE If the request already contains a `MetadataSectionURI` element, then the reply should not contain a `MetadataSectionURI` element.
`version`	Indicates the version of this metadata section. NOTE This International Standard does not specify a format for the version. However, if the metadata schema provides a versioning mechanism for its metadata, then the SP should use it for the `version` attribute.

Semantics of the `RequestContentRequest`:

Name	*Definition*
`RequestContentRequest`	Protocol message sent from the client to the SP in order to request the specified content.
`RequestContentRequestType`	Top-level type for `RequestContentRequest`. `RequestContentRequestType` extends `ProtocolRequestType`.
`ContentEntity`	A Digital Item or a reference to it.
`LicenseEntity`	A license associated with content that may be needed for verifying rights for the requested content.
`UsageEnvironmentDescription`	Description of device capabilities and user preferences. The capabilities may also contain Sensory Device Capabilities as specified by the `cidl:SensoryDeviceCapabilityBaseType` in ISO/IEC 23005-2. The user preferences may also contain User Sensory Preferences as specified by the `cidl:UserSensoryPreferenceBaseType` in ISO/IEC 23005-2.

Semantics of the `RequestContentResponse`:

Name	*Definition*
`RequestContentResponse`	This element is used as response to a Request Content operation.
`RequestContentResponseType`	Top-level type for `RequestContentResponse`. `RequestContentResponseType` extends `ProtocolResponseType`.
`RequestContentSuccess`	Response in case of success.
`RequestContentFailure`	Response in case of failure.

Semantics of the `RequestContentSuccessType`:

Name	*Definition*
`RequestContentSuccessType`	This element is used as a response to a Request Content operation.
`ContentEntity`	A Digital Item or a reference to it.

5.17.1.5 Extension to SID

For the representation of Service Instance Declarations of *Request Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="RequestContentSIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="SupportedMetadataSchema"
type="mpegmb:SupportedMetadataSchemaType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:RequestContentSIDType`:

Name	*Definition*
`sid:RequestContentSIDType`	Top-level type for SID of the *Request Content* ES. `sid:RequestContentSIDType` extends `sid:ServiceInstanceDeclarationType`.
`sid:SupportedMetadataSchema`	Lists metadata schemas in which metadata sections can be provided.

5.17.1.6 Examples

NOTE The provided examples cover only the Request Metadata steps.

EXAMPLE 1 A Service Provider (SP) has an enormous database of metadata about IPTV providers and their offerings. An illustration of the database in ETSI IPTV metadata schema is provided in Figure 2, where each physical metadata segment is a basic metadata section. The structure (with section levels) of the database can be represented by the Classification Scheme for ETSI IPTV (Annex A.2).

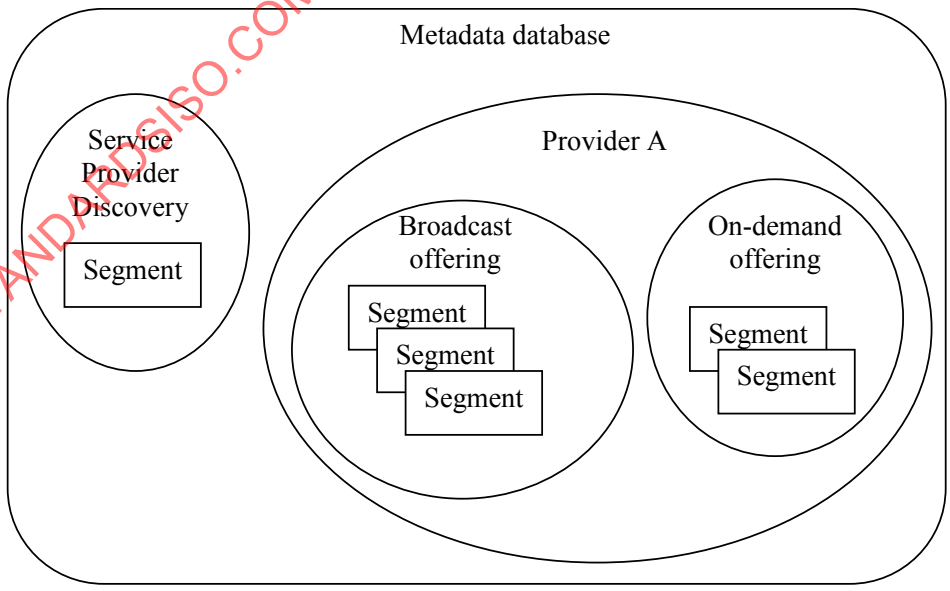

Figure 2 — Structure of the ETSI IPTV metadata schema

To learn about available IPTV providers, a Client first requests the metadata section of "ServiceProviderDiscovery" type of ETSI IPTV metadata schema using one of the two following request messages. Note that the database contains exactly one metadata section of type "ServiceProviderDiscovery" which contains information about all providers.

If the Client somehow knows the URI of that metadata section (e.g., "http://example.com/123"), then the request will be as follows.

```
<RequestMetadataRequest>
  <mpegmb:TransactionIdentifier>41</mpegmb:TransactionIdentifier>
  <SupportedMetadataSchema>
    urn:mpeg:mpegM:cs:02-iptvods-NS:2012:ETSIOfferingDiscoverySectionsCS
  </SupportedMetadataSchema>
  <MetadataSectionURI>http://example.com/123</MetadataSectionURI>
</RequestMetadataRequest>
```

Otherwise, the Client will send the following request, in which the metadata section's identification is based on the Classification Scheme of Annex A.2.

Note that there is only one the metadata section of type "ServiceProviderDiscovery" in the database.

```
<RequestMetadataRequest>
  <mpegmb:TransactionIdentifier>41</mpegmb:TransactionIdentifier>
  <SupportedMetadataSchema>
    urn:mpeg:mpegM:cs:02-iptvods-NS:2012:ETSIOfferingDiscoverySectionsCS
  </SupportedMetadataSchema>
  <SectionCriterion>
    <SectionKind href="urn:mpeg:mpegM:cs:02-iptvods-
NS:2012:ETSIOfferingDiscoverySectionsCS:1">
      <mpeg7:Name>ServiceProviderDiscovery</mpeg7:Name>
    </SectionKind>
  </SectionCriterion>
</RequestMetadataRequest>
```

The SP responds with an affirmative `RequestMetadataResponse` message to the Client, conveying the requested metadata section. In this example, the namespace "`etsi:`" is used to indicate the metadata of ETSI IPTV schema [9]. The replied metadata section (with a `version` value of "3") would provide a list of IPTV providers and, for each provider, the list of available offerings and IDs of associated metadata sections.

```
<RequestMetadataResponse>
  <mpegmb:TransactionIdentifier>41</mpegmb:TransactionIdentifier>
  <RequestMetadataSuccess version="3">
    <MetadataSection>
      <MetadataSectionContent>
        <etsi:ServiceDiscovery>
          <etsi:ServiceProviderDiscovery>
            <etsi:ServiceProvider DomainName="www.ProviderA.com">
              <etsi:Offering>
                <etsi:Push>
                  <etsi:PayloadId Id="2">
                    <etsi:Segment ID="1"/>
                    <etsi:Segment ID="2"/>
                  </etsi:PayloadId>
                </etsi:Push>
              </etsi:Offering>
            </etsi:ServiceProvider>
            <etsi:ServiceProvider DomainName="www.ProviderB.com">
```

```
                <etsi:Offering>
                   <!-- metadata of Provider B goes here... -->
                </etsi:Offering>
             </etsi:ServiceProvider>
          </etsi:ServiceProviderDiscovery>
       </etsi:ServiceDiscovery>
     </MetadataSectionContent>
   </MetadataSection>
  </RequestMetadataSuccess>
</RequestMetadataResponse>
```

EXAMPLE 2 Suppose that the Client repeats the same request but sets the `version` attribute to "3".

```
<RequestMetadataRequest version="3">
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <SupportedMetadataSchema>
    urn:mpeg:mpegM:cs:02-iptvods-NS:2012:ETSIOfferingDiscoverySectionsCS
  </SupportedMetadataSchema>
  <SectionCriterion>
    <SectionKind href="urn:mpeg:mpegM:cs:02-iptvods-
NS:2012:ETSIOfferingDiscoverySectionsCS:1">
      <mpeg7:Name>ServiceProviderDiscovery</mpeg7:Name>
    </SectionKind>
  </SectionCriterion>
</RequestMetadataRequest>
```

If the metadata section at the Provider is still the same, the response message below will not contain any metadata section, implying that the current metadata section at the Client does not need to be updated.

```
<RequestMetadataResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <RequestMetadataSuccess>
    <SuccessCode>NO_CONTENT</SuccessCode>
  </RequestMetadataSuccess>
</RequestMetadataResponse>
```

EXAMPLE 3 Now, suppose that the Client wants to run an Electronic Programme Guide (EPG) application showing all broadcast offerings of (IPTV) Provider A, whose identification is the domain name "www.ProviderA.com". For this purpose, the Client requests the metadata of ETSI IPTV schema that describes broadcast offerings of Provider A. Note that the term "offering" is also called "service" in IPTV terminology.

The following shows a `RequestMetadataRequest` message sent by the Client to the SP, requesting for a metadata section, which is segment number 1 of BroadcastService type (with a payload ID of "2" as explained in Annex A.2) of Provider A. The message includes two criteria: the first criterion specifies that the section belongs to Provider A, the second criterion specifies that the first broadcast service from the list is requested.

```
<RequestMetadataRequest>
  <mpegmb:TransactionIdentifier>43</mpegmb:TransactionIdentifier>
  <SectionCriterion>
    <SectionKind href="urn:mpeg:2010:DiscoveryCS-NS:2">
       <mpeg7:Name>ServiceProvider</mpeg7:Name>
    </SectionKind>
     <SectionIdentification>
       <TextualID>www.ProviderA.com</TextualValue>
    </SectionIdentification>
  </SectionCriterion>
```

```
  <SectionCriterion>
    <SectionKind href="urn:mpeg:2010:DiscoveryCS-NS:2.1">
      <mpeg7:Name>BroadcastService</mpeg7:Name>
    </SectionKind>
     <SectionIdentification>
       <NumericID>1</NumericID>
     </SectionIdentification>
   </SectionCriterion>
</RequestMetadataRequest>
```

Again, the SP responds with the following `RequestMetadataResponse` message to the Client, conveying the requested metadata section. The section has a `version` value of "5".

```
<RequestMetadataResponse>
  <mpegmb:TransactionIdentifier>43</mpegmb:TransactionIdentifier>
  <RequestMetadataSuccess version="5">
    <MetadataSection>
      <MetadataSectionContent>
        <!-- content of metadata section goes here -->
        <etsi:ServiceDiscovery>
          <etsi:BroadcastService>
          <!-- ... -->
          </etsi:BroadcastService>
        </etsi:ServiceDiscovery>
      </MetadataSectionContent>
    </MetadataSection>
  </RequestMetadataSuccess>
</RequestMetadataResponse>
```

EXAMPLE 4 Suppose that the Client has obtained a content identifier (e.g., through the Request Metadata steps described above). The Client can request the content item by sending a `RequestContentRequest` message. As the Client supports some Sensory Effects, the corresponding terminal capabilities are indicated in the `UsageEnvironmentDescription`.

```
<RequestContentRequest>
  <mpegmb:TransactionIdentifier>44</mpegmb:TransactionIdentifier>
   <ContentEntity>
     <mpegmb:ContentRef>urn:mpegRA:dii:crid:123</mpegmb:ContentRef>
   </ContentEntity>
  <UsageEnvironmentDescription>
    <dia:UsageEnvironmentProperty xsi:type="dia:TerminalsType">
      <dia:Terminal>
        <dia:TerminalCapability xsi:type="dcdv:LightCapabilityType"/>
        <dia:TerminalCapability xsi:type="dcdv:FogCapabilityType"/>
        <dia:TerminalCapability xsi:type="dcdv:WindCapabilityType"/>
        <dia:TerminalCapability xsi:type="dcdv:HeatingCapabilityType"/>
      </dia:Terminal>
    </dia:UsageEnvironmentProperty>
    <!-- More MPEG-21 DIA Usage Environment Description goes here ... -->
  </UsageEnvironmentDescription>
</RequestContentRequest>
```

The Service Provider resolves the DII and returns a `RequestContentResponse` message containing the location of the content. The Service Provider also takes care of including Sensory Information for the supported Sensory Devices in the content.

```
<RequestContentResponse>
  <mpegmb:TransactionIdentifier>44</mpegmb:TransactionIdentifier>
   <RequestContentSuccess>
     <ContentEntity>
       <mpegmb:ContentRef>rtsp://example.com/xyz</mpegmb:ContentRef>
     </ContentEntity>
   <RequestContentSuccess>
</RequestContentResponse>
```

5.17.2 Request Contract

5.17.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Request Contract elementary service.

This Elementary Service enables a User to gain access to a Contract.

5.17.2.2 Interfaces and protocol specification

The Request Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The Client sends a `RequestContractRequest` message. The message needs a `ContractEntity`. Usually the contract is represented by a `ContractIdentifier`.	
2.		The Service Provider sends a `RequestContractResponse` message. If the operation ends successfully, then the message contains a Contract or a reference to it.

5.17.2.3 Syntax of protocol data format

```
<!-- ################################################################ -->
<!--  Request Contract                                                -->
<!-- ################################################################ -->

<!-- Definition of RequestContractRequest -->
<element name="RequestContractRequest" type="mpegm:RequestContractRequestType"/>
  <complexType name="RequestContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

```
<!-- Definition of RequestContractResponse -->
<element name="RequestContractResponse"
type="mpegm:RequestContractResponseType"/>
  <complexType name="RequestContractResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="RequestContractSuccess"
type="mpegm:RequestContractSuccessType"/>
          <element name="RequestContractFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="RequestContractSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.17.2.4 Semantics of protocol data format

Semantics of the `RequestContractRequest`:

Name	*Definition*
`RequestContractRequest`	Request message for requesting a contract.
`RequestContractRequestType`	Top-level type for `RequestContractRequest`. `RequestContractRequestType` extends `ProtocolRequestType`.
`ContractEntity`	a `ContractEntity` associated with the requested entity. Typically it will contain a `ContractIdentifier`.

Semantics of the `RequestContractResponse`:

Name	*Definition*
`RequestContractResponse`	Response message for requesting a contract.
`RequestContractResponseType`	Top-level type for `RequestContractResponse`. `RequestContractResponseType` extends `ProtocolResponseType`.
`RequestContractSuccess`	Response in case of success.
`RequestContractFailure`	Response in case of failure.

Semantics of the `RequestContractSuccessType`:

Name	*Definition*
`ContractEntity`	The requested contract.

5.17.2.5 Extension to SID

For the representation of Service Instance Declarations of *Request Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RequestContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.17.3 Request Device

5.17.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Request Device Protocol.

This Elementary Service enables a User to request a Device (such as some kind of software, i.e. an IPMPTool) basing on the usage environment characteristic of the User.

5.17.3.2 Interfaces and protocol specification

The Request Device Protocol is as follows:

Steps	Client	Service Provider
1.	A client sends a `RequestDeviceRequest` message. The message contains the device identifier and some device information.	
2.		The SP sends a `RequestDeviceResponse` message indicating whether the operation was successful or not. In case of success, the message contains the requested device.

5.17.3.3 Syntax of protocol data format

```
<!-- ################################################################ -->
<!--  Request Device                                                  -->
<!-- ################################################################ -->

<!-- Definition of RequestDeviceRequest -->
<element name="RequestDeviceRequest" type="mpegm:RequestDeviceRequestType"/>
  <complexType name="RequestDeviceRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="DeviceEntity" type="mpegmb:DeviceEntityType"/>
          <element name="DeviceInformation" type="dia:UsageEnvironmentType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<!-- Definition of RequestDeviceResponse -->
<element name="RequestDeviceResponse" type="mpegm:RequestDeviceResponseType"/>
  <complexType name="RequestDeviceResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="RequestDeviceSuccess"
type="mpegm:RequestDeviceSuccessType"/>
            <element name="RequestDeviceFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="RequestDeviceSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="DeviceEntity" type="mpegmb:DeviceEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.17.3.4 Semantics of protocol data format

Semantics of the `RequestDeviceRequest`:

Name	*Definition*
`RequestDeviceRequest`	The Request for Request Device.
`RequestDeviceRequestType`	Top-level type for `RequestDeviceRequest`. `RequestDeviceRequestType` extends `ProtocolRequestType`.

Name	*Definition*
`DeviceEntity`	Typically the device identifier.
`DeviceInformation`	Optional Information about the device.

Semantics of the `RequestDeviceResponse`:

Name	*Definition*
`RequestDeviceResponse`	The response message for requesting a device.
`RequestDeviceResponseType`	Top-level type for `RequestDeviceResponse`. `RequestDeviceResponseType` extends `ProtocolResponseType`.
`RequestDeviceSuccess`	Response in case of success.
`RequestDeviceFailure`	Response in case of failure.

Semantics of the `RequestDeviceSuccess`:

Name	*Definition*
`DeviceEntity`	The requested Device or a reference to it.

5.17.3.5 Extension to SID

For the representation of Service Instance Declarations of *Request Device*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RequestDeviceSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```


5.17.4 Request Event

5.17.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Request Event Protocol.

This Elementary Service enables a User both to request the MPEG-21 Event Report Request that may be associated with a Content or a set of MPEG-21 Event Reports associated with an MPEG-21 Event Report Request.

5.17.4.2 Interfaces and protocol specification

The Request Event Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `RequestEventRequest` message. The message contains either: — a `ContentEntity` if the user is looking for the MPEG-21 ERR associated to that Content; or — an Event Report Request (`erl:ERR`) if the user is looking for the Event Reports related to it.	
2.		The SP sends a `RequestEventResponse`. If the request contains a Content, then the response is an MPEG-21 Event Report Request (`erl:ERR`), otherwise it is a set of MPEG-21 Event Reports (`erl:ER`).

5.17.4.3 Syntax of protocol data format

```
<!-- ********************************************************** -->
<!--                  RequestEvent                              -->
<!-- ********************************************************** -->
<element name="RequestEventRequest" type="mpegm:RequestEventRequestType"/>
<complexType name="RequestEventRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <choice>
        <element ref="erl:ERR" />
        <element name="ContentEntity" type="mpegm:ContentEntityType" />
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="RequestEventResponse" type="mpegm:RequestEventResponseType"/>
<complexType name="RequestEventResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
    <sequence>
      <choice>
      <element name="RequestEventSuccess" type="mpegm:RequestEventSuccessType"/>
      <element name="RequestEventFailure" type="mpegmb:ProtocolFailureType"/>
      </choice>
    </sequence>
    </extension>
  </complexContent>
```

```
</complexType>

<complexType name="RequestEventSuccessType">
  <complexContent>
    <extension base="mpegmb:ProtocolSuccessType">
      <choice>
        <element ref="erl:ERR"/>
        <element ref="erl:ER" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

5.17.4.4 Semantics of protocol data format

Semantics of the `RequestEventRequest`:

Name	*Definition*
`RequestEventRequest`	The message for creating a request for Request Event.
`RequestEventRequestType`	Top-level type for `RequestEventRequest`. `RequestEventRequestType` extends `ProtocolRequestType`.
`ContentEntity`	The reference to a content associated with an Event Report Request.
`erl:ERR`	An MPEG-21 Event Report Request.

Semantics of the `RequestEventResponse`:

Name	*Definition*
`RequestEventResponse`	The response contains the result of the RequestEvent Service.
`RequestEventResponseType`	Top-level type for `RequestEventResponse`. `RequestEventResponseType` extends `ProtocolResponseType`.
`RequestEventSuccess`	Response in case of success.
`RequestEventFailure`	Response in case of failure.

Semantics of the `RequestEventSuccessType`:

Name	*Definition*
`erl:ERR`	An MPEG-21 Event Report Request.
`erl:ER`	A set of one or more Event Reports.

5.17.4.5 Additional validation rules

5.17.4.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.17.4.5.2 If the `RequestEventRequest` message contains a `ContentEntity` and the `RequestEventResponse` message contains a `RequestEventSuccess`, then the `RequestEventSuccess` shall contain a `erl:ERR`.

5.17.4.5.3 If the `RequestEventRequest` message contains a `erl:ERR` and the `RequestEventResponse` message contains a `RequestEventSuccess`, then the `RequestEventSuccess` shall contain one or more `erl:ER` elements.

5.17.4.6 Extension to SID

For the representation of Service Instance Declarations of *Request Event*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RequestEventSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.17.5 Request License

5.17.5.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Request License Protocol.

This Elementary Service enables a User to gain access to a License.

5.17.5.2 Interfaces and protocol specification

The Request License Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `RequestLicenseRequest` message. The message needs at least a `ContentEntity`. A `LicenseEntity` can be specified since a Content may be associated with multiple Licenses, so a specific license can be defined.	

Steps	Client	Service Provider
2.		The SP sends a `RequestLicenseResponse`, message. In case of success, the message contains the requested License or a reference to it.

5.17.5.3 Syntax of protocol data format

```
<!-- ################################################################ -->
<!--  Request License                                                 -->
<!-- ################################################################ -->

<!-- Definition of RequestLicenseRequest -->
<element name="RequestLicenseRequest" type="mpegm:RequestLicenseRequestType"/>
  <complexType name="RequestLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<!-- Definition of RequestLicenseResponse -->
<element name="RequestLicenseResponse" type="mpegm:RequestLicenseResponseType"/>
  <complexType name="RequestLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="RequestLicenseSuccess"
type="mpegm:RequestLicenseSuccessType"/>
          <element name="RequestLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="RequestLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.17.5.4 Semantics of protocol data format

Semantics of the `RequestLicenseRequest`:

Name	*Definition*
`RequestLicenseRequest`	Request message for requesting a license.
`RequestLicenseRequestType`	Top-level type for `RequestLicenseRequest`. `RequestLicenseRequestType` extends `ProtocolRequestType`.
`ContentEntity`	The content with which the license is associated.
`LicenseEntity`	A LicenseEntity associated with the requested entity. Typically it contains a `LicenseIdentifier`. Because a Content may be associated with multiple Licenses, a specific license can be defined.

Semantics of the `RequestLicenseResponse`:

Name	*Definition*
`RequestLicenseResponse`	Response message for requesting a license.
`RequestLicenseResponseType`	Top-level type for `RequestLicenseResponse`. `RequestLicenseResponseType` extends `ProtocolResponseType`.
`RequestLicenseSuccess`	Response in case of success.
`RequestLicenseFailure`	Response in case of failure.

Semantics of the `RequestLicenseSuccessType`:

Name	*Definition*
`LicenseEntity`	The requested license.

5.17.5.5 Extension to SID

For the representation of Service Instance Declarations of *Request License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RequestLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.18 The Revoke Services

5.18.1 Revoke Entity

5.18.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the abstract Revoke Entity Protocol.

The Revoke Entity abstract Elementary Service allows Users to discontinue the validity of an Entity (i.e., a contract, a license or a content). The Service Provider delivering the Revoke Entity service keeps track of the Entities, under its authority, and their status, which may be one of "in formation", "in force", or "terminated" (e.g., cancelled, expired, etc.). The SP needs also to be able to verify if Users invoking this service have the permission to revoke the earlier mentioned revocable Entities. The way SPs maintain this information is out of the scope of this specification, being implementation-dependent. The protocol of this Elementary Service has to be used as a guide for any extra Revoke Service that may be specified, apart from the ones included in the following Sections.

5.18.1.2 Interfaces and protocol specification

The Revoke Entity Protocol is as follows:

Steps	Client	Service Provider
1.	The Client sends a message of type `RevokeEntityRequestType` containing the item to be revoked. The request messages will also make use of the `saml:Assertion` child element of `mpegmb:ProtocolRequestType`, in order to assert the identity of the client.	
2.		The SP analyzes the request parameters and the rights metadata, governing the Entity in scope, to determine if the client has the necessary permissions to call for the revocation of said Entity. If so, the Entity is revoked/eliminated from the SP's scope. The maintenance of the rights metadata, by the SP, is outside the scope of this specification.
3.		The server sends a message of type `RevokeEntityResponseType`, with information on the result of the operation.

5.18.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Revoke Entity                                   -->
  <!-- ################################################ -->
  <!-- Definition of RevokeEntityRequestType -->
  <complexType name="RevokeEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="RevokedEntity" type="mpegmb:EntityBaseType">
            <annotation>
              <documentation xml:lang="en">
                Each regular Elementary Service puts here an Entity of the
appropriate type (e.g., mpegmb:ContractEntityType).
              </documentation>
            </annotation>
          </element>
          <element name="RevocationDate" type="dateTime" minOccurs="0"/>
          <element name="RevocationReason" type="mpegmb:RevocationReasonType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>


  <!-- Definition of RevokeEntityResponseType -->
  <complexType name="RevokeEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="RevokeEntitySuccess"
type="mpegmb:ProtocolSuccessType"/>
            <element name="RevokeEntityFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.18.1.4 Semantics of protocol data format

Semantics of the `RevokeEntityRequestType`:

Name	*Definition*
`RevokeEntityRequestType`	Top-level type for requests for revoking an Entity.
`RevokedEntity`	The Entity to be revoked (i.e., content, contract, or license).
`RevocationDate`	Date when the Entity shall be revoked.
`RevocationReason`	Reason for the revocation as defined in 5.2.3.19.

Semantics of the `RevokeEntityResponseType`:

Name	*Definition*
`RevokeEntityResponseType`	Top-level type for responses for the revocation of an Entity.
`RevokeEntitySuccess`	Response in case of success.
`RevokeEntityFailure`	Response in case of failure.

5.18.1.5 Extension to SID

For the representation of Service Instance Declarations of *Revoke Entity*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RevokeEntitySIDType" abstract="true">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.18.2 Revoke Content

5.18.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Revoke Content Protocol.

This Elementary Service enables a User to discontinue the validity of a digital content, represented/declared by a specific Digital Item Declaration.

It differs from other Revoke Elementary Services in the sense that it performs the elimination of a content (and, implicitly, of all licenses that pertain to it), and not merely of a set of rights over said content.

For instance, revoking the license to a specific content means revoking the rights of the specific set of users that possessed that license. Many other licenses may exist pertaining to said content, and, as such, the content may remain accessible to other users.

When revoking the content itself, all references to said content, which serve as entry points to it and allow retrieving it, are being removed from the system's scope. The Content becomes unreachable and unavailable to Users. Copies of the Content which are stored in the private space of the user's device or in protected locations may still be available.

The invocation of this ES has, thus, much broader consequences than the invocation of Revoke License or Revoke Contract ESs.

The Revoke Content elementary service is based on the abstract Revoke Entity elementary service defined in 5.18.1.

The item of the revocation is a DID represented as `mpegmb:ContentEntityType`.

The saml:Assertion of the mpegmb:ProtocolRequestType provides a signed assertion of the user's identity. This enables the verification that the user calling the Revoke Content ES is the one specified in a License previously emitted by the owner of the Content that enables the revocation of the content in question.

5.18.2.2 Interfaces and protocol specification

The Revoke Content Protocol extends the abstract Revoke Entity Protocol defined in 5.18.1.2, from which it inherits the protocol specification.

5.18.2.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!-- Revoke Content                                   -->
  <!-- ################################################ -->
  <!-- Definition of RevokeContentRequest -->

  <complexType name="RevokeContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="RevokedContent" type="mpegmb:ContentEntityType"/>
          <element name="RevocationDate" type="dateTime" minOccurs="0"/>
          <element name="RevocationReason" type="mpegmb:RevocationReasonType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of RevokeContentResponse -->
  <element name="RevokeContentResponse" type="mpegm:RevokeEntityResponseType"/>
```

5.18.2.4 Semantics of protocol data format

Semantics of the RevokeContentRequest:

Name	*Definition*
RevokeContentRequest	Request for revoking a specific Content. RevokeContentRequest is specified according to RevokeEntityRequestType.
RevokedContent	The ContentEntity to be revoked
RevocationDate	See semantics of the RevokeEntityRequestType in 5.18.1.4.
RevocationReason	See semantics of the RevokeEntityRequestType in 5.18.1.4.

Semantics of the RevokeContentResponse:

Name	*Definition*
RevokeContentResponse	Response message for the revoking a content. RevokeContentResponse is of type RevokeEntityResponseType.

5.18.2.5 Additional validation rules

5.18.2.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.18.2.5.2 The `RevokedEntity` element of the `RevokeContentRequest` message must be of type `mpegmb:ContentEntityType`.

5.18.2.6 Extension to SID

For the representation of Service Instance Declarations of *Revoke Content*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->
  <complexType name="RevokeContentSIDType">
    <complexContent>
      <extension base="sid:RevokeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.18.3 Revoke Contract

5.18.3.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Revoke Contract elementary service.

The Revoke Contract Service allows Users to discontinue the validity of a Contract. The Service Provider providing Revoke Contract keeps a store of Contracts together with its status.

The Revoke Contract elementary service is based on the abstract Revoke Entity elementary service defined in 5.18.1.

The item of the revocation is a CEL Contract represented as `mpegmb:ContractEntityType`.

5.18.3.2 Interfaces and protocol specification

The Revoke Contract Protocol extends the abstract Revoke Entity Protocol defined in 5.18.1.2, from which it inherits the protocol specification.

5.18.3.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Revoke Contract                                 -->
  <!-- ################################################ -->
  <!-- Definition of RevokeContractRequest -->

  <complexType name="RevokeContractRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="RevokedContract" type="mpegmb:ContractEntityType"/>
          <element name="RevocationDate" type="dateTime" minOccurs="0"/>
```

```
          <element name="RevocationReason" type="mpegmb:RevocationReasonType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of RevokeContractResponse -->
  <element name="RevokeContractResponse" type="mpegm:RevokeEntityResponseType"/>
```

5.18.3.4 Semantics of protocol data format

Semantics of the `RevokeContractRequest`:

Name	*Definition*
`RevokeContractRequest`	Request message for revoking a contract. `RevokeContractRequest` is specified according to `RevokeEntityRequestType`.
`RevokedContract`	The ContractEntity to be revoked
`RevocationDate`	See semantics of the `RevokeEntityRequestType` in 5.18.1.4.
`RevocationReason`	See semantics of the `RevokeEntityRequestType` in 5.18.1.4.

Semantics of the `RevokeContractResponse`:

Name	*Definition*
`RevokeContractResponse`	Response message for revoking a contract. `RevokeContractResponse` is of type `RevokeEntityResponseType`.

5.18.3.5 Additional validation rules

5.18.3.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.18.3.5.2 The `RevokedEntity` element of the `RevokeContractRequest` message must be of type `mpegmb:ContractEntityType`.

5.18.3.6 Extension to SID

For the representation of Service Instance Declarations of *Revoke Contract*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="RevokeContractSIDType">
    <complexContent>
      <extension base="sid:RevokeEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.18.4 Revoke License

5.18.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Revoke License Protocol.

This Elementary Service enables a User to discontinue the validity of a License.

The Revoke License elementary service is based on the abstract Revoke Entity elementary service defined in 5.18.1.

The item of the revocation is a license represented as `mpegmb:LicenseEntityType`.

5.18.4.2 Interfaces and protocol specification

The Revoke License Protocol extends the abstract Revoke Entity Protocol defined in 5.18.1.2, from which it inherits the protocol specification.

5.18.4.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!-- Revoke License                                   -->
  <!-- ################################################ -->
  <!-- Definition of RevokeLicenseRequest -->

  <complexType name="RevokeLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="RevokedLicense" type="mpegmb:LicenseEntityType"/>
          <element name="RevocationDate" type="dateTime" minOccurs="0"/>
          <element name="RevocationReason" type="mpegmb:RevocationReasonType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of RevokeLicenseResponse -->
  <element name="RevokeLicenseResponse" type="mpegm:RevokeEntityResponseType"/>
```

5.18.4.4 Semantics of protocol data format

Semantics of the `RevokeLicenseRequest`:

Name	*Definition*
`RevokeLicenseRequest`	Request for revoking a license. `RevokeLicenseRequest` is specified according to `RevokeEntityRequestType`.
`RevokedLicense`	The LicenseEntity to be revoked
`RevocationDate`	See semantics of the `RevokeEntityRequestType` in 5.18.1.4.
`RevocationReason`	See semantics of the `RevokeEntityRequestType` in 5.18.1.4.

Semantics of the `RevokeLicenseResponse`:

Name	*Definition*
`RevokeLicenseResponse`	Response message for the revoking a license. `RevokeLicenseResponse` is of type `RevokeEntityResponseType`.

5.18.4.5 Additional validation rules

5.18.4.5.1 Introduction

For the purpose of referencing the additional validation rules are numbered.

5.18.4.5.2 The `RevokedEntity` element of the `RevokeLicenseRequest` message must be of type `mpegmb:LicenseEntityType`.

5.18.4.6 Extension to SID

For the representation of Service Instance Declarations of *Revoke License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="RevokeLicenseSIDType">
  <complexContent>
    <extension base="sid:RevokeEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.19 The Search Services

5.19.1 Search Entity

5.19.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the abstract Search Entity Protocol.

The Search Entity Protocol is employed to request to locate Entities (Content, Contract, Device, License, Service, and User) in a multimedia content value chain satisfying some criteria. With the Search Entity Protocol a User can request a Search Entity Service Provider (SP) to search for Entities of Describe Entity Service Providers (DBs/Ontologies) according to specified terms and conditions represented in the MPEG Query Format. The Search Entity Protocol is applied between a User and Search Entity Service Provider. The MPEG Query Format is typically also employed between the Search Entity Service Provider and Describe Entity Service Providers. However, the communication between the Search Entity SP and the Describe Entity SPs is out of the scope of MPEG-M.

The Search Entity Service generally involves two users:

a) **Client**, which is a User requesting the Search Entity service.

b) **Search Entity Service Provider** providing the Search Entity service for the requesting User.

5.19.1.2 Interfaces and protocol specification

The Search Entity Protocol specifies how to request a search for an Entity Item from DB/Ontologies of Describe Entity Service Providers. The Protocol is applied between a User who is an Service Provider/or an End User, and Search Entity SP. Between Search Entity SP and Describe Entity Service Providers (DB/Ontology), MPEG Query Format is applied as defined in ISO/IEC 15938-12:2008 and ISO/IEC 15938-12/:2008/Amd.2. The Semantic enhancement to the MPEG Query Format enables the search for an entity in an ontology, e.g., Media Value Chain Ontology, using semantic relation which includes various properties in the ontology. Therefore, different query conditions for the Search Entity are possible to be composed using the properties defined in the ontology without relying on the predefined MPEG-7 Content description types. The example of semantic query is provided in the informative subclause. The protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a message of type `SearchEntityRequestType` to the Search Entity SP, containing the description of the search.	
2.		The Search Entity SP, upon receiving the message, performs the following steps:
3.		The SP sends to Describe Entity Service Provider(s), the query management messages concerning service discovery, querying service capability, and service capability description.
4.		The SP sends to Describe Entity Service Provider(s), the MPEG Query message.

Steps	Client	Service Provider
5.		After having received the MPEG Query results from multiple Describe Entity Service Provider(s), the SP generates a SearchEntityResponse message.
6.		The SP sends a message of type `SearchEntityResponseType` to the client.

5.19.1.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Search Entity                                   -->
  <!-- ################################################ -->

  <!-- Definition of SearchEntityRequestType -->
  <complexType name="SearchEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="InputQuery" type="mpqf:InputQueryType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>


  <!-- Definition of SearchEntityResponseType -->
  <complexType name="SearchEntityResponseType">
    <complexContent>
      <extension base=" mpegmb:ProtocolResponseType">
        <choice>
          <element name="SearchEntitySuccess"
type="mpegm:SearchEntitySuccessType"/>
          <element name="SearchEntityFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="SearchEntitySuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="OutputQuery" type="mpqf:OutputQueryType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.19.1.4 Semantics of protocol data format

Semantics of the `SearchEntityRequestType`:

Name	*Definition*
`SearchEntityRequestType`	Top-level type for Search Entity Request messages. `SearchEntityRequestType` extends `ProtocolRequestType`.
`InputQuery`	A set of conditions and/or the specification of the structure and content of the output query format and a declaration part as specified in the MPEG QF.

Semantics of the `SearchEntityResponseType`:

Name	*Definition*
`SearchEntityResponseType`	Top-level type for Search Entity Response messages. `SearchEntityResponseType` extends `ProtocolResponseType`.
`SearchEntitySuccess`	Response in case of success.
`SearchEntityFailure`	Response in case of failure.

Semantics of the `SearchEntitySuccessType`:

Name	*Definition*
`OutputQuery`	An element identifying the query results as defined in MPEG QF.

5.19.1.5 Extension to SID

For the representation of Service Instance Declarations of *Search Entity*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="SearchEntitySIDType" abstract="true">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="SupportedMetadataSchema"
type="mpegmb:SupportedMetadataSchemaType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:SearchEntitySIDType`:

Name	*Definition*
`sid:SearchEntitySIDType`	Top-level type for SIDs of the *Search Entity* abstract ES. `sid:SearchEntitySIDType` extends `sid:ServiceInstanceDeclarationType`.
`sid:SupportedMetadataSchema`	Lists metadata schemas in which the Entity description can be provided. NOTE If the `strictMetadataSupport` attribute of the SID (see C.3) is set to "true", the `SupportedMetadataSchema` element is redundant and shall not be used.

5.19.1.6 Examples

The following examples show the use of the MPEG Query format.

EXAMPLE 1 This example shows the case that the query requests images whose MPEG-7 metadata has the `CreationInformation/Creation/Title` element with the value "Miracle Query Format" exactly.

```
<mpqf:MpegQuery mpqfID="someID">
  <mpqf:Query>
    <mpqf:Input>
      <mpqf:QueryCondition>
        <mpqf:TargetMediaType>image/jpeg</mpqf:TargetMediaType>
        <mpqf:Condition xsi:type="mpqf:QueryByDescription" matchType="exact">
          <mpqf:DescriptionResource resourceID="desc001">
            <mpqf:AnyDescription>
              <mpeg7:Mpeg7>
                <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationInformationType">
                  <mpeg7:Creation>
                    <mpeg7:Title>Miracle Query Format</mpeg7:Title>
                  </mpeg7:Creation>
                </mpeg7:DescriptionUnit>
              </mpeg7:Mpeg7>
            </mpqf:AnyDescription>
          </mpqf:DescriptionResource>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>
```

EXAMPLE 2 This example illustrates the use of semantic expressions on an imaginary semantic annotation in a surveillance system scenario, as defined in ISO/IEC 15938-12:2008/Amd 2. The example searches for persons and its location and time information (see `mpqf:QutputDescription`) that matches the drawn RDF subgraph (see `mpqf:Condition` element). The example demonstrates the use of the integrated `anchor` and `anchorDistance` attributes. Here the first SemanticRelation-Condition defines the anchor meaning that all other specified conditions build the search graph pattern related to their given `anchorDistance`.

```
<mpqf:MpegQuery mpqfID="someID">
  <mpqf:Query>
    <mpqf:Input>
      <mpqf:QFDeclaration>
        <mpqf:Prefix name="xsd" uri="http://www.w3.org/2001/XMLSchema#"/>
```

```
        <mpqf:Prefix name="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"/>
        <mpqf:Prefix name="ontology" uri="http://example.com/ontology#"/>
        <mpqf:Prefix name="config" uri="http://example.com/configuration#"/>
      </mpqf:QFDeclaration>
      <mpqf:OutputDescription>
        <mpqf:ReqSemanticField>?person</mpqf:ReqSemanticField>
        <mpqf:ReqSemanticField>?location</mpqf:ReqSemanticField>
        <mpqf:ReqSemanticField>?temp</mpqf:ReqSemanticField>
      </mpqf:OutputDescription>
      <mpqf:QueryCondition>
        <mpqf:Condition xsi:type="mpqf:AND">
          <mpqf:Condition xsi:type="mpqf:SemanticRelation" anchor="true"
anchorDistance="0">
            <mpqf:Subject>?person</mpqf:Subject>
            <mpqf:Property>ontology:hasName</mpqf:Property>
            <mpqf:Object>?name</mpqf:Object>
          </mpqf:Condition>
          <mpqf:Condition xsi:type="mpqf:SemanticRelation" anchorDistance="3">
            <mpqf:Subject>?cam</mpqf:Subject>
            <mpqf:Property>ontology:isLocated</mpqf:Property>
            <mpqf:Object>?location</mpqf:Object>
          </mpqf:Condition>
          <mpqf:Condition xsi:type="mpqf:QueryByFreeText">
            <mpqf:FreeText>Room240</mpqf:FreeText>
            <mpqf:SearchField>?location</mpqf:SearchField>
          </mpqf:Condition>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>
```

EXAMPLE 3 The following example illustrates the use of the SPARQL [12] query type on an imaginary semantic annotation in a surveillance system scenario. The example searches for persons and its location and time information (see `mpqf:OutputDescription`) that matches the drawn RDF subgraph (see `mpqf:ASK` part within the `mpqf:Query` element).

```
<mpqf:MpegQuery mpqfID="someID">
  <mpqf:Query>
    <mpqf:Input>
      <mpqf:OutputDescription>
        <mpqf:ReqSemanticField>?person</mpqf:ReqSemanticField>
        <mpqf:ReqSemanticField>?location</mpqf:ReqSemanticField>
        <mpqf:ReqSemanticField>?temp</mpqf:ReqSemanticField>
      </mpqf:OutputDescription>
      <mpqf:QueryCondition>
        <mpqf:Condition xsi:type="mpqf:QueryBySPARQL">
          <mpqf:SPARQL>
            <![CDATA[
        PREFIX xsd:      <http://www.w3.org/2001/XMLSchema#>
        PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX ontology: <http://example.com/ontology#>
        PREFIX config:   <http://example.com/configuration#>
        ASK
        { ?person   ontology:hasName      ?name ;
                    ontology:hasPosition  ?node .
          ?node     ontology:hasPosition_Location  ?roi ;
                    ontology:hasPosition_Duration  ?temp .
```

```
          ?roi      ontology:hasCamera    ?cam .
          ?cam      ontology:isLocated    ?location .
          ?location ontology:hasName     "Room240"^^xsd:string .
        } ]]>
          </mpqf:SPARQL>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>
```

5.19.2 Search Content

5.19.2.1 Introduction

The Search Content elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

5.19.2.2 Interfaces and protocol specification

The Search Content Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.2.3 Syntax of protocol data format

```
<!-- ########################################### -->
<!--  Search Content                             -->
<!-- ########################################### -->

<!-- Definition of SearchContentRequest -->
<element name="SearchContentRequest" type="mpegm:SearchEntityRequestType"/>

<!-- Definition of SearchContentResponse -->
<element name="SearchContentResponse" type=" mpegm:SearchEntityResponseType"/>
```

5.19.2.4 Semantics of protocol data format

Semantics of the `SearchContentRequest`:

Name	*Definition*
`SearchContentRequest`	Protocol message sent from the client to the SP to request that certain Content is sought under provided terms and conditions.

Semantics of the `SearchContentResponse`:

Name	*Definition*
`SearchContentResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.2.5 Extension to SID

For the representation of Service Instance Declarations of *Search Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="SearchContentSIDType">
    <complexContent>
      <extension base="sid:SearchEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.19.2.6 Examples

EXAMPLE 1 This example shows the run of the Search Content Protocol. The User sends a `SearchContentRequest` message to the SP, requesting content with a text annotation "Volcano".

```
<SearchContentRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <InputQuery>
    <mpqf:QueryCondition>
      <mpqf:Condition xsi:type="mpqf:QueryByFreeText">
        <mpqf:FreeText>Volcano</mpqf:FreeText>

<mpqf:SearchField>Mpeg7/Description/AudioVisual/TextAnnotation</mpqf:SearchField>
      </mpqf:Condition>
    </mpqf:QueryCondition>
  </InputQuery>
</SearchContentRequest>
```

The SP responds with a `SearchContentResponse` message containing a list of matching resources.

```
<SearchContentResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <SearchEntitySuccess>
    <OutputQuery>
      <mpqf:GlobalComment>This is the message from the responder
</mpqf:GlobalComment>
      <mpqf:ResultItem recordNumber="001" rank="1" confidence="1.0">

<mpqf:MediaResource>http://example.com/db/video/19701221.mpg</mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="002" rank="2" confidence="0.99">

<mpqf:MediaResource>http://example.com/db/video/19690117.mpg</mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="003" rank="2" confidence="0.98">

<mpqf:MediaResource>http://example.com/db/video/19980212.mpg</mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="004" rank="3" confidence="0.87">

<mpqf:MediaResource>http://example.com/db/video/19990414.mpg</mpqf:MediaResource>
```

```
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="005" rank="3" confidence="0.85">

<mpqf:MediaResource>http://example.com/db/video/20071119.mpg</mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:SystemMessage>
        <mpqf:Status>
          <mpqf:Code>001</mpqf:Code>
          <mpqf:Description>Query was successful</mpqf:Description>
        </mpqf:Status>
      </mpqf:SystemMessage>
    </OutputQuery>
  </SearchEntitySuccess>
</SearchContentResponse>
```

EXAMPLE 2 The User sends a `SearchContentRequest` message to the SP, requesting content containing Sensory Information with wind effects (represented as `sev:WindEffect`). The SP responds with the collection of matching files.

NOTE Sensory Information is considered as content (not as metadata) and is therefore represented as a `didl:Resource` inside a `didl:Component` of a Digital Item.

```
<SearchContentRequest>
  <mpegmb:TransactionIdentifier>45</mpegmb:TransactionIdentifier>
  <InputQuery>
    <mpqf:QueryCondition>
      <mpqf:Condition xsi:type="mpqf:Equal">
        <mpqf:StringField>//sedl:SEM//sedl:Effect@xsi:type</mpqf:StringField>
        <mpqf:StringValue>sev:WindType</mpqf:StringValue>
      </mpqf:Condition>
    </mpqf:QueryCondition>
  </InputQuery>
</SearchContentRequest>
```

The SP responds with a `SearchContentResponse` message containing a list of matching resources.

```
<SearchContentResponse>
  <mpegmb:TransactionIdentifier>45</mpegmb:TransactionIdentifier>
  <SearchEntitySuccess>
    <OutputQuery>
      <mpqf:GlobalComment>
      This is the message from the responder
    </mpqf:GlobalComment>
      <mpqf:ResultItem recordNumber="001" rank="1" confidence="1.0">
        <mpqf:MediaResource>
        http://example.com/content/some-content.didl
      </mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="002" rank="2" confidence="0.99">
        <mpqf:MediaResource>
        http://example.com/content/other-content.m21
      </mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:SystemMessage>
        <mpqf:Status>
          <mpqf:Code>001</mpqf:Code>
          <mpqf:Description>Query was successful</mpqf:Description>
        </mpqf:Status>
```

```
      </mpqf:SystemMessage>
    </OutputQuery>
  </SearchEntitySuccess>
</SearchContentResponse>
```

5.19.3 Search Contract

5.19.3.1 Introduction

The Search Contract elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

5.19.3.2 Interfaces and protocol specification

The Search Contract Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.3.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Search Contract                                 -->
<!-- ################################################ -->

<!-- Definition of SearchContractRequest -->
<element name="SearchContractRequest" type="mpegm:SearchEntityRequestType"/>

<!-- Definition of SearchContractResponse -->
<element name="SearchContractResponse" type="mpegm:SearchEntityResponseType"/>
```

5.19.3.4 Semantics of protocol data format

Semantics of the `SearchContractRequest`:

Name	*Definition*
`SearchContractRequest`	Protocol message sent from the client to the SP to request that certain Contract is sought under provided terms and conditions.

Semantics of the `SearchContractResponse`:

Name	*Definition*
`SearchContractResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.3.5 Extension to SID

For the representation of Service Instance Declarations of *Search Contract*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="SearchContractSIDType">
    <complexContent>
      <extension base="sid:SearchEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.19.3.6 Example

This example shows the run of the Search Contract Protocol. The client sends a `SearchContractRequest` message to the SP, requesting for a contract of a specific date.

```
<SearchContractRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <InputQuery>
    <mpqf:OutputDescription maxItemCount="3" maxPageEntries="10"
outputNameSpace="urn:mpeg:mpeg7:schema:2004">
      <mpqf:ReqField typeName="title">metadata/dc:title</mpqf:ReqField>
      <mpqf:ReqField
typeName="description">metadata/dc:description</mpqf:ReqField>
      <mpqf:ReqField typeName="creator">metadata/dc:creator</mpqf:ReqField>
      <mpqf:ReqField typeName="date">metadata/dc:date</mpqf:ReqField>
    </mpqf:OutputDescription>
    <mpqf:QueryCondition>
      <mpqf:Condition xsi:type="mpqf:Equal">
        <mpqf:DateTimeField>metadata/dc:date</mpqf:DateTimeField>
        <mpqf:DateTimeValue>2009-10-26T10:40:39</mpqf:DateTimeValue>
      </mpqf:Condition>
    </mpqf:QueryCondition>
  </InputQuery>
</SearchContractRequest>
```

The SP responds with a `SearchContractResponse` message containing the results of the search.

```
<SearchContractResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <SearchEntitySuccess>
    <OutputQuery>
      <mpqf:GlobalComment>This is the message from the responder
</mpqf:GlobalComment>
      <mpqf:ResultItem recordNumber="001" rank="1" confidence="1.0">

<mpqf:MediaResource>http://example.com/db/videoContract/19701221</mpqf:MediaResou
rce>
        <mpqf:Description>
          <dc:description>This is the contract for an audio file
</dc:description>
          <dc:creator>John Lawo</dc:creator>
          <dc:date>2009-10-26T10:40:39</dc:date>
        </mpqf:Description>
      </mpqf:ResultItem>
      <mpqf:SystemMessage>
        <mpqf:Status>
          <mpqf:Code>001</mpqf:Code>
```

```
          <mpqf:Description>Query was successful</mpqf:Description>
        </mpqf:Status>
      </mpqf:SystemMessage>
    </OutputQuery>
  </SearchEntitySuccess>
</SearchContractResponse>
```

5.19.4 Search Device

5.19.4.1 Introduction

The Search Device elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

5.19.4.2 Interfaces and protocol specification

The Search Device Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.4.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Search Device                                   -->
<!-- ################################################ -->

<!-- Definition of SearchDeviceRequest -->
<element name="SearchDeviceRequest" type="mpegm:SearchEntityRequestType"/>

<!-- Definition of SearchDeviceResponse -->
<element name="SearchDeviceResponse" type="mpegm:SearchEntityResponseType"/>
```

5.19.4.4 Semantics of protocol data format

Semantics of the `SearchDeviceRequest`:

Name	*Definition*
`SearchDeviceRequest`	Protocol message sent from the client to the SP to request that certain Devices are sought under provided terms and conditions.

Semantics of the `SearchDeviceResponse`:

Name	*Definition*
`SearchDeviceResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.4.5 Extension to SID

For the representation of Service Instance Declarations of *Search Device*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="SearchDeviceSIDType">
  <complexContent>
    <extension base="sid:SearchEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.19.5 Search License

5.19.5.1 Introduction

The Search License elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

5.19.5.2 Interfaces and protocol specification

The Search License Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.5.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Search License                                  -->
<!-- ################################################ -->

<!-- Definition of SearchLicenseRequest -->
<element name="SearchLicenseRequest" type="mpegm:SearchEntityRequestType"/>

<!-- Definition of SearchLicenseResponse -->
<element name="SearchLicenseResponse" type="mpegm:SearchEntityResponseType"/>
```

5.19.5.4 Semantics of protocol data format

Semantics of the `SearchLicenseRequest`:

Name	*Definition*
`SearchLicenseRequest`	Protocol message sent from the client to the SP to request that certain License is sought under provided terms and conditions.

Semantics of the `SearchLicenseResponse`:

Name	*Definition*
`SearchLicenseResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.5.5 Extension to SID

For the representation of Service Instance Declarations of *Search License*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="SearchLicenseSIDType">
    <complexContent>
      <extension base="sid:SearchEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.19.5.6 Example

This example shows the run of the Search License Protocol. The User sends a `SearchLicenseRequest` message to the SP, requesting for a License with a text annotation "Second Revision Endowed"

```
<SearchLicenseRequest>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <InputQuery>
    <mpqf:QueryCondition>
      <mpqf:Condition xsi:type="mpqf:QueryByFreeText">
        <mpqf:FreeText>Second Revision Endowed</mpqf:FreeText>

<mpqf:SearchField>Mpeg21/LicenseDescription/AudioVisual/TextAnnotation</mpqf:Sear
chField>
      </mpqf:Condition>
    </mpqf:QueryCondition>
  </InputQuery>
</SearchLicenseRequest>
```

The SP responds with a `SearchLicenseResponse` message containing the results of the search.

```
<SearchLicenseResponse>
  <mpegmb:TransactionIdentifier>42</mpegmb:TransactionIdentifier>
  <SearchEntitySuccess>
    <OutputQuery>
      <mpqf:GlobalComment>This is the message from the responder
</mpqf:GlobalComment>
      <mpqf:ResultItem recordNumber="001" rank="1" confidence="1.0">

<mpqf:MediaResource>http://example.com/db/videoLicense/19701221</mpqf:MediaResour
ce>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="002" rank="2" confidence="0.99">
```

```
<mpqf:MediaResource>http://example.com/db/videoLicense/19690117</mpqf:MediaResour
ce>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="003" rank="2" confidence="0.98">

<mpqf:MediaResource>http://example.com/db/videoLicense/19980212</mpqf:MediaResour
ce>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="004" rank="3" confidence="0.87">

<mpqf:MediaResource>http://example.com/db/videoLicense/19990414</mpqf:MediaResour
ce>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="005" rank="3" confidence="0.85">

<mpqf:MediaResource>http://example.com/db/videoLicense/20071119</mpqf:MediaResour
ce>
      </mpqf:ResultItem>
      <mpqf:SystemMessage>
        <mpqf:Status>
          <mpqf:Code>001</mpqf:Code>
          <mpqf:Description>Query was successful</mpqf:Description>
        </mpqf:Status>
      </mpqf:SystemMessage>
    </OutputQuery>
  </SearchEntitySuccess>
</SearchLicenseResponse>
```

5.19.6 Search Service

5.19.6.1 Introduction

The Search Service elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

The Search Service Protocol allows the search for Service Instances. The representation of Service Instances is specified in Annex C.

5.19.6.2 Interfaces and protocol specification

The Search Service Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.6.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--  Search Service                                  -->
  <!-- ################################################ -->

  <!-- Definition of SearchServiceRequest -->
  <element name="SearchServiceRequest" type="mpegm:SearchEntityRequestType"/>

  <!-- Definition of SearchServiceResponse -->
  <element name="SearchServiceResponse" type="mpegm:SearchEntityResponseType"/>
```

5.19.6.4 Semantics of protocol data format

Semantics of the SearchServiceRequest:

Name	*Definition*
`SearchServiceRequest`	Protocol message sent from the client to the SP to request that certain Services are sought under provided terms and conditions.

Semantics of the SearchServiceResponse:

Name	*Definition*
`SearchServiceResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.6.5 Extension to SID

For the representation of Service Instance Declarations of *Search Service*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="SearchServiceSIDType">
  <complexContent>
    <extension base="sid:SearchEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.19.7 Search User

5.19.7.1 Introduction

The Search User elementary service is based on the abstract Search Entity elementary service defined in 5.19.1.

5.19.7.2 Interfaces and protocol specification

The Search User Protocol extends the abstract Search Entity Protocol defined in 5.19.1.2, from which it inherits the protocol specification.

5.19.7.3 Syntax of protocol data format

```
<!-- ################################################ -->
<!--  Search User                                     -->
<!-- ################################################ -->

<!-- Definition of SearchUserRequest -->
<element name="SearchUserRequest" type="mpegm:SearchEntityRequestType"/>
```

```
  <!-- Definition of SearchUserResponse -->
  <element name="SearchUserResponse" type="mpegm:SearchEntityResponseType"/>
```

5.19.7.4 Semantics of protocol data format

Semantics of the `SearchUserRequest`:

Name	*Definition*
`SearchUserRequest`	Protocol message sent from the client to the SP to request that certain Users are sought under provided terms and conditions.

Semantics of the `SearchUserResponse`:

Name	*Definition*
`SearchUserResponse`	Protocol message sent from the SP to the client with the results of the search.

5.19.7.5 Extension to SID

For the representation of Service Instance Declarations of *Search User*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="SearchUserSIDType">
    <complexContent>
      <extension base="sid:SearchEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.19.7.6 Example

This example shows the run of the Search User Protocol. The client sends a `SearchUserRequest` message to the SP, requesting for a User with a last name "Brown" in the media creator list. The SearchUserResponse message is sent to the User with the result "John Brown".

```
<SearchUserRequest>
  <mpegmb:TransactionIdentifier>52</mpegmb:TransactionIdentifier>
  <InputQuery>
    <mpqf:OutputDescription outputNameSpace="urn:mpeg:mpeg7:schema:2004">
      <mpqf:ReqField typeName="MediaLocatorType">//MediaUri</mpqf:ReqField>
      <mpqf:ReqField
typeName="CreationInformationType">//Creation/Title</mpqf:ReqField>
      <mpqf:ReqField
typeName="CreationInformationType">//Creation/Creator</mpqf:ReqField>
```

```
    </mpqf:OutputDescription>
    <mpqf:QueryCondition>
      <mpqf:TargetMediaType
xsi:type="mpqf:mimeType">image/jpeg</mpqf:TargetMediaType>
      <mpqf:Condition xsi:type="mpqf:Equal">
        <mpqf:StringField>//Creation/Creator/Agent/FamilyName</mpqf:StringField>
        <mpqf:StringValue>Brown</mpqf:StringValue>
      </mpqf:Condition>
    </mpqf:QueryCondition>
  </InputQuery>
</SearchUserRequest>
```

The SP responds with a `SearchUserResponse` message containing the results.

```
<SearchUserResponse>
  <mpegmb:TransactionIdentifier>52</mpegmb:TransactionIdentifier>
  <SearchEntitySuccess>
    <OutputQuery currPage="1" totalPages="1" expirationDate="2008-05-
30T09:00:00">
      <mpqf:ResultItem xsi:type="mpqf:ResultItemType" recordNumber="1">
        <mpqf:TextResult>Title 01</mpqf:TextResult>
        <mpqf:Description xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
          <mpeg7:Mpeg7>
            <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationType">
              <mpeg7:Title>Title 01</mpeg7:Title>
              <mpeg7:Creator xsi:type="mpeg7:CreatorType">
                <mpeg7:Role href="urn:mpeg:mpeg7:cs:RoleCS:2001:AUTHOR"/>
                <mpeg7:Agent xsi:type="mpeg7:PersonType">
                  <mpeg7:Name>
                    <mpeg7:GivenName>John</mpeg7:GivenName>
                    <mpeg7:FamilyName>Brown</mpeg7:FamilyName>
                  </mpeg7:Name>
                </mpeg7:Agent>
              </mpeg7:Creator>
            </mpeg7:DescriptionUnit>
          </mpeg7:Mpeg7>
        </mpqf:Description>
      </mpqf:ResultItem>
    </OutputQuery>
  </SearchEntitySuccess>
</SearchUserResponse>
```


5.20 The Store Services

5.20.1 Store Content

5.20.1.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Store Content elementary service.

This Elementary Service enables a User to store a Digital Item or any of its components on another device.

5.20.1.2 Interfaces and protocol specification

The Store Content Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `StoreContentRequest` message. The message contains one Digital Item or a reference to it and the required transfer protocol.	
2.		The SP retrieves all resources referenced by the DID and stores them locally. The SP starts updating the references in the DID according to the new locations of the resources. If the client has requested an immediate response message, the SP continues with step . Otherwise, if the client has not requested an immediate response message, the SP continues with step 4. NOTE The SP is not required to parse any of these resources. It shall not store any data that is referenced by these resources (e.g., references to external media data via the `moov/mvhd/mdia/minf/dinf/dref/url` box of an MP4 file). The SP may indicate the existence of such referenced data via the `DisplayString` element of the `StoreContentResponse` message.
3.		(optional) The SP sends a `StoreContentResponse` message to the client, indicating whether the upload will be performed. If the upload cannot be performed, the protocol ends here.
4.		After the upload has finished, the SP sends a `StoreContentCompletion` message to the client, indicating whether the delivery has been performed successfully. In case of success, the `StoreContentCompletion` message contains the updated DID.

5.20.1.3 Syntax of protocol data format

```
  <!-- ############################################################ -->
  <!--  Store Content                                               -->
  <!-- ############################################################ -->
  <!-- Definition of StoreContentRequest -->
  <element name="StoreContentRequest" type="mpegm:StoreContentRequestType"/>
  <complexType name="StoreContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentEntity" type="mpegmb:ContentEntityType"/>
          <element name="SupportedTransferProtocol"
type="mpegmb:SupportedTransferProtocolType" maxOccurs="unbounded"/>
```

```
        </sequence>
        <attribute name="immediateResponse" type="boolean" use="optional"
default="true"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of StoreContentResponse -->
  <element name="StoreContentResponse" type="mpegm:StoreContentResponseType"/>
  <complexType name="StoreContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="StoreContentSuccess" type="mpegmb:ProtocolSuccessType"/>
          <element name="StoreContentFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <!-- Definition of StoreContentCompletion -->
  <element name="StoreContentCompletion"
type="mpegm:StoreContentCompletionType"/>
  <complexType name="StoreContentCompletionType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="StoreContentCompletionSuccess"
type="mpegm:StoreContentCompletionSuccessType"/>
          <element name="StoreContentCompletionFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="StoreContentCompletionSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="StoredContentEntity" type="mpegmb:ContentEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.20.1.4 Semantics of protocol data format

Semantics of the `StoreContentRequest`:

Name	*Definition*
`StoreContentRequest`	Protocol message sent from the client to the SP to request that the specified Content is stored.
`StoreContentRequestType`	Top-level type for `StoreContentRequest`. `StoreContentRequestType` extends `ProtocolRequestType`.

ContentEntity	A DI or a reference to it.
SupportedTransferProtocol	Specifies the transfer protocols that the client wants to use to store the content (the list of the transfer protocols accepted by the SP is listed in the SID).
immediateResponse	Indicates whether the SP has to respond with a StoreContentResponse message. If set to "false", the SP will only send the StoreContentCompletion message after the upload has finished.

Semantics of the StoreContentResponse:

Name	*Definition*
StoreContentResponse	This message is used as response to a Store Content operation. It indicates whether the upload will be performed.
StoreContentResponseType	Top-level type for StoreContentResponse. StoreContentResponseType extends ProtocolResponseType.
StoreContentSuccess	Response in case of success.
StoreContentFailure	Response in case of failure.

Semantics of the StoreContentCompletion:

Name	*Definition*
StoreContentCompletion	This message is sent after the Store Content operation. It indicates whether the content was uploaded and stored successfully.
StoreContentCompletionType	Top-level type for StoreContentCompletion. StoreContentCompletionType extends ProtocolResponseType.
StoreContentCompletionSuccess	Response in case of success.
StoreContentCompletionFailure	Response in case of failure.

Semantics of the StoreContentCompletionSuccessType:

Name	*Definition*
StoredContentEntity	An updated DID for the stored content. Typically this contains the location of the stored content.

5.20.1.5 Extension to SID

For the representation of Service Instance Declarations of *Store Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="StoreContentSIDType">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType">
        <sequence>
          <element name="SupportedTransferProtocol"
type="mpegmb:SupportedTransferProtocolType" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:StoreContentSIDType`:

Name	*Definition*
`sid:StoreContentSIDType`	Top-level type for SIDs of the *Store Content* regular ES. `sid:StoreContentSIDType` extends `sid:ServiceInstanceDeclarationType`.
`sid:SupportedTransferProtocol`	Lists supported transfer protocols over which the Content can be sent.

5.20.2 Store Contract

5.20.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Store Contract Protocol. This Service is allows Users to save Contracts in a Device for later use.

NOTE Given the simplicity of a Contract Entity compared to the Content Entity, no transference protocols can be specified nor the location or destination of the stored Contract.

5.20.2.2 Interfaces and protocol specification

The Store Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `StoreContractRequest` message, containing the contract to be stored.	
2.		The SP sends a `StoreContractResponse` indicating whether the operation was successful or not.

5.20.2.3 Syntax of protocol data format

```
  <!-- ################################################################ -->
  <!--  Store Contract                                                  -->
  <!-- ################################################################ -->
  <!-- Definition of StoreContractRequest -->
  <element name="StoreContractRequest" type="mpegm:StoreEntityRequestType"/>
  <complexType name="StoreEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContractEntity" type="mpegmb:ContractEntityType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of StoreContractResponse -->
  <element name="StoreContractResponse" type="mpegm:StoreContractResponseType"/>
<complexType name="StoreContractResponseType">    <complexContent>
<extension base="mpegmb:ProtocolResponseType">      <choice>          <element
name="StoreContractSuccess" type="mpegmb:ProtocolSuccessType"/>          <element
name="StoreContractFailure" type="mpegmb:ProtocolFailureType"/>        </choice>
</extension>    </complexContent>  </complexType>
```

5.20.2.4 Semantics of protocol data format

Semantics of the `StoreContractRequest`:

Name	*Definition*
`StoreContractRequest`	The request for storing a contract contains simply the contract.
`ContractEntity`	One or more contracts on MPEG-21 CEL format.

Semantics of the `StoreContractResponse`:

Name	*Definition*
`StoreContractResponse`	The response contains the result of the Store Contract Service.
`StoreContractResponseType`	Top-level type for `StoreContractResponse`. `StoreContractResponseType` extends `ProtocolResponseType`.
`StoreContractSuccess`	Response in case of success.
`StoreContractFailure`	Response in case of failure.

5.20.2.5 Extension to SID

For the representation of Service Instance Declarations of *Store Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="StoreContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.20.3 Store Event

5.20.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Store Event Protocol.

This Elementary Service enables a User to store one of more MPEG-21 Event Reports occurred on a Device or an MPEG-21 Event Report Request related to a particular Digital Item (as made on 5.8.1). In the first case, the related Event Report Request can be specified in the `ER/ERDescriptor/ERSource` field and in the second case the identifier of the Digital Item can be specified in the `ERR/ERSpecification` field.

5.20.3.2 Interfaces and protocol specification

The Store Event Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `StoreEventRequest` message, to report that an event occurred. The event is an MPEG-21 ER that occurred on a Device or is an MPEG-21 ERR related to a particular Digital Item.	
2.		The SP sends a `ReportEventResponse` indicating whether the operation was successful or not.

5.20.3.3 Syntax of protocol data format

```
<!-- ************************************************************ -->
<!--                  StoreEvent                                  -->
<!-- ************************************************************ -->
<element name="StoreEventRequest" type="mpegm:StoreEventRequest"/>
<complexType name="StoreEventRequest">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
```

```
        <choice>
          <element ref="erl:ERR"/>
          <element ref="erl:ER" maxOccurs="unbounded"/>
         </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="StoreEventResponse" type="mpegm:StoreEventResponseType"/>
<complexType name="StoreEventResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
    <choice>
      <element name="StoreEventSuccess" type="mpegmb:ProtocolSuccessType"/>
      <element name="StoreEventFailure" type="mpegmb:ProtocolFailureType"/>
    </choice>
    </extension>
  </complexContent>
</complexType>
```

5.20.3.4 Semantics of protocol data format

Semantics of the `StoreEventRequest`:

Name	*Definition*
`StoreEventRequest`	The Request for Store Event as one of more MPEG-21 Event Reports occurred on a Device or an MPEG-21 Event Report Request related to a particular Digital Item
`StoreEventRequestType`	Top-level type for `StoreEventRequest`. `StoreEventRequestType` extends `ProtocolResponseType`.
`erl:ER`	An MPEG-21 Event Report to be stored.
`erl:ERR`	An MPEG-21 Event Report Request.
`ContentEntity`	A DI or a reference to it.

Semantics of the `StoreEventResponse`:

Name	*Definition*
`StoreEventResponse`	The response contains the result of the Store Event Service.
`StoreEventResponseType`	Top-level type for `StoreEventResponse`. `StoreEventResponseType` extends `ProtocolResponseType`.
`StoreEventSuccess`	Response in case of success.
`StoreEventFailure`	Response in case of failure.

5.20.3.5 Extension to SID

For the representation of Service Instance Declarations of *Store Event*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="StoreEventSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.20.4 Store License

5.20.4.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Store License Protocol.

This Elementary Service enables a User to store a License to a Device.

5.20.4.2 Interfaces and protocol specification

The Store License Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `StoreLicenseRequest` message. The message contains the licenses (inside the `License` or the `ContentEntity` element, depending on the number of licenses sent) that the client wants to store and a reference to the Digital Item involved.	
2.		The server stores the license for later retrieval and sends a `StoreLicenseResponse`.

5.20.4.3 Syntax of protocol data format

```
<!-- ############################################################ -->
<!--  Store License                                               -->
<!-- ############################################################ -->

<!-- Definition of StoreLicenseRequest -->
<element name="StoreLicenseRequest" type="mpegm:StoreLicenseRequestType"/>
  <complexType name="StoreLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <choice>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
```

```
          <element name="ContentEntity" type="ContentEntityType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

<!-- Definition of StoreLicenseResponse -->
<element name="StoreLicenseResponse" type="mpegm:StoreLicenseResponseType"/>
  <complexType name=" StoreLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="StoreLicenseSuccess"
type="mpegm:StoreLicenseSuccessType"/>
          <element name="StoreLicenseFailure" type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="StoreLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.20.4.4 Semantics of protocol data format

Semantics of the `StoreLicenseRequest`:

Name	*Definition*
`StoreLicenseRequest`	This element is used to send one (or more) rights information related to a content item
`StoreLicenseRequestType`	Top-level type for `StoreLicenseRequest`. `StoreLicenseRequestType` extends `ProtocolRequestType`.
`License`	A REL license.
`ContentEntity`	A reference to a content containing one or more licenses.

Semantics of the `StoreLicenseResponse`:

Name	*Definition*
`StoreLicenseResponse`	This element is used as response to a Store License operation.
`StoreLicenseResponseType`	Top-level type for `StoreLicenseResponse`. `StoreLicenseResponseType` extends `ProtocolResponseType`.

Name	*Definition*
`StoreLicenseSuccess`	Response in case of success.
`StoreLicenseFailure`	Response in case of failure.

Semantics of the `StoreLicenseSuccessType`:

Name	*Definition*
`LicenseEntity`	The location of the stored license.

5.20.4.5 Extension to SID

For the representation of Service Instance Declarations of *Store License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="StoreLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```


5.21 The Transact Services

5.21.1 Transact Entity

5.21.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the abstract Transact Entity Protocol. In particular, this subclause provides the protocol performed by a Seller/Buyer and their Transact Entity SPs (TESP) and the XML representation of the data structures of messages exchanged between a Subscriber and a TESP.

5.21.1.2 Reference model

The Transact Entity Protocol assumes that there are in general 4 actors:

a) **Seller** owning rights to a digital media object

b) **Buyer** wishing to acquire rights to that digital media object

c) **Subscriber**, indicating either a Seller or a Buyer in case the distinction between Seller and Buyer is not relevant

d) **Transact Entity Service Provider** (TESP) handling transactions between Seller and Buyer

The actors involved in this reference model are represented in Figure 3.

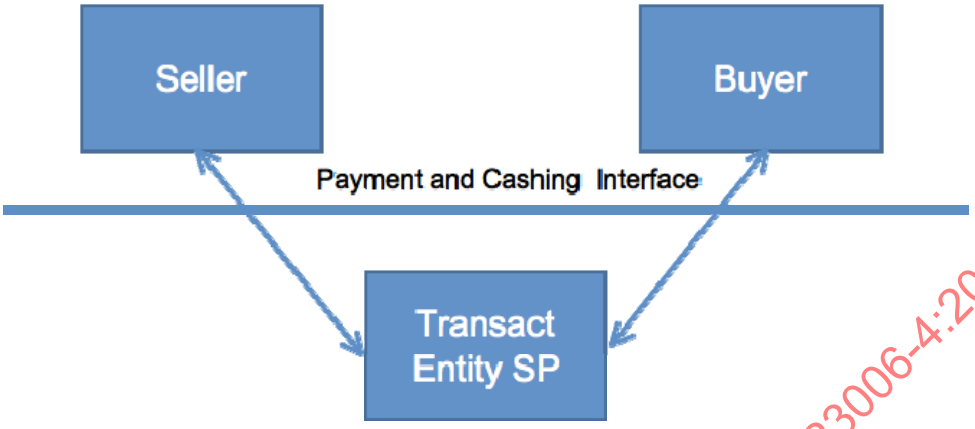

Figure 3 — Actors in a payment and cashing system

A general actor is enabled to:

— perform transactions (payment or cashing operations) (SellerPaymentOrder and BuyerPaymentOrderRequest)

— give a mandate (allowing his TESP to pay a specific seller without been requested for confirmation) (MandateOffer)

— revoke a mandate (MandateRevoke)

5.21.1.3 Interfaces and protocol specification

The Transact Entity Protocol specifies how to perform the transaction of an Entity. The protocol is as follows:

Steps	Client (Seller)	Service Provider	Buyer
1.			(optional) The buyer decides to give a debit mandate to the TESP without being requested for a confirmation for each transaction with the seller, sending a message of type `MandateOfferEntityRequestType`.
2.		(optional) The TESP confirm that a mandate with given seller has been set, by sending a message of type `MandateOfferEntityResponseType` to the buyer.	
3.			(optional) The buyer decides to remove revoke a debit mandate to the TESP without being requested for a confirmation for each transaction with the seller,

Steps	Client (Seller)	Service Provider	Buyer
			sending a message of type `MandateRevokeEntityRequestType`.
4.		(optional) The TESP confirm that a mandate with given seller has been revoked, by sending a message of type `MandateRevokeEntityResponseType` to the buyer.	
5.	A Seller starts a transaction by sending a message of type `SellerPaymentOrderEntityRequestType`to his TESP.		
6.		(optional) The TESP sends a message of type `BuyerPaymentOrderEntityRequestType` to the Buyer requesting confirmation of payment for the given transaction. These steps are only performed if the TESP does not have a corresponding debit mandate from the buyer.	
7.			(optional) Upon receipt, the Buyer confirms payment by sending a message of type `BuyerPaymentOrderEntityResponseType` to the TESP.
8.		The TESP sends a message of type `SellerPaymentOrderEntityResponseType` to the Seller to inform on payment result for the given transaction.	

5.21.1.4 Seller Payment Order Entity

5.21.1.4.1 Introduction

A Seller wishing to bill a Buyer, e.g., after receiving a Purchase Request, exchanges the following messages with the relevant users.

A message of type `SellerPaymentOrderEntityRequestType` is sent by Seller to his TESP.

A message of type `SellerPaymentOrderEntityResponseType` is sent by Seller's TESP to Seller.

5.21.1.4.2 Syntax of protocol data format

```
  <!-- Definition of SellerPaymentOrderEntityRequest -->
  <complexType name="SellerPaymentOrderEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of SellerPaymentOrderEntityResponse -->
  <complexType name="SellerPaymentOrderEntityResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="SellerPaymentOrderEntitySuccess"
type="mpegm:SellerPaymentOrderEntitySuccessType"/>
            <element name="SellerPaymentOrderEntityFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="SellerPaymentOrderEntitySuccessType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.21.1.4.3 Semantics of protocol data format

Semantics of the `SellerPaymentOrderEntityRequestType`:

Name	*Definition*
`SellerPaymentOrderEntityRequestType`	Abstract top-level type for initiating a payment process. `SellerPaymentOrderEntityRequestType` extends `ProtocolRequestType`.

Name	*Definition*
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall not be set.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

Semantics of the `SellerPaymentOrderEntityResponseType`:

Name	*Definition*
`SellerPaymentOrderEntityResponseType`	Abstract top-level type for confirming that the payment was performed. `SellerPaymentOrderEntityResponseType` extends `ProtocolResponseType`.
`SellerPaymentOrderEntitySuccess`	Message in case of success. Contains a payment confirmation.
`SellerPaymentOrderEntityFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `SellerPaymentOrderEntitySuccessType`:

Name	*Definition*
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price.
`BuyerAccount`	Account from which payment will be effected.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

5.21.1.5 Buyer Payment Order Entity

5.21.1.5.1 Introduction

A TESP may request a Buyer to pay for a content item or a service.

A message of type `BuyerPaymentOrderEntityRequestType` is sent by the Buyer's TESP to the Buyer.

A message of type `BuyerPaymentOrderEntityResponseType` is sent by Buyer to Buyer's TESP.

5.21.1.5.2 Syntax of protocol data format

```
  <!-- Definition of BuyerPaymentOrderEntityRequest -->
  <complexType name="BuyerPaymentOrderEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType" />
          <element name="SellerAccount" type="mpeg7:UniqueIDType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of BuyerPaymentOrderEntityResponse -->
  <complexType name="BuyerPaymentOrderEntityResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="BuyerPaymentOrderEntitySuccess"
type="mpegm:BuyerPaymentOrderEntitySuccessType"/>
            <element name="BuyerPaymentOrderEntityFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="BuyerPaymentOrderEntitySuccessType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType" />
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType" />
          <element name="SellerTESP" type="mpeg7:UniqueIDType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```


5.21.1.5.3 Semantics of protocol data format

Semantics of the `BuyerPaymentOrderEntityRequest`:

Name	*Definition*
`BuyerPaymentOrderEntityRequestType`	Abstract top-level type for request the buyer to perform a payment. `BuyerPaymentOrderEntityRequestType` extends `ProtocolRequestType`.

Name	*Definition*
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall be set.
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`SellerPCSP`	Information on seller's TESP.

Semantics of the `BuyerPaymentOrderEntityResponse`:

Name	*Definition*
`BuyerPaymentOrderEntityResponseType`	Abstract top-level type for confirming that the payment was performed. `BuyerPaymentOrderEntityResponseType` extends `ProtocolResponseType`.
`BuyerPaymentOrderEntitySuccess`	Message part in case of success. Contains a payment confirmation.
`BuyerPaymentOrderEntityFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `BuyerPaymentOrderEntitySuccessType`:

Name	*Definition*
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller to buyer.

5.21.1.6 Mandate Offer Entity

5.21.1.6.1 Introduction

A Buyer wishing for payments to be performed by a Buyer's TESP without seeking specific approval by the Buyer for each transaction, exchanges the following messages:

A message of type `MandateOfferEntityRequestType` is sent by a Buyer to his TESP.

A message of type `MandateOfferEntityResponseType` is sent by Buyer's TESP to Seller.

5.21.1.6.2 Syntax of protocol data format

```
 <!-- Definition of MandateOfferEntityRequest -->
 <complexType name="MandateOfferEntityRequestType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType" />
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType" />
          <element name="MandateOptions" type="mpegm:MandateOptionsType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="MandateOptionsType">
    <sequence>
      <element name="MandateID" type="string"/>
      <element name="TransactionAmount" type="rel-sx:Rate"/>
      <element name="GlobalAmount" type="rel-sx:Rate"/>
      <element name="StartDate" type="dateTime"/>
      <element name="ExpirationDate" type="dateTime"/>
    </sequence>
  </complexType>

  <!-- Definition of MandateOfferEntityResponseType -->
  <complexType name="MandateOfferEntityResponseType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="MandateOfferEntitySuccess"
type="mpegm:MandateOfferEntitySuccessType"/>
            <element name="MandateOfferEntityFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="MandateOfferEntitySuccessType" abstract="true">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="EntityOrderInformation"
type="mpegmb:EntityOrderInformationType"/>
          <element name="MandateID" type="string"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType" />
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType" />
          <element name="SellerTESP" type="mpeg7:UniqueIDType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.21.1.6.3 Semantics of protocol data format

Semantics of the `MandateOfferEntityRequest`:

Name	*Definition*
`MandateOfferEntityRequestType`	Abstract top-level type for offering a debit mandate. `MandateOfferEntityRequestType` extends `ProtocolRequestType`.
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payments will be credited.
`MandateOptions`	Information on mandate.

Semantics of the `MandateOptionsType`:

Name	*Definition*
`MandateID`	Identifier of the mandate.
`TransactionAmount`	Maximum amount for a single transaction (including any VAT).
`GlobalAmount`	Maximum amount for the whole mandate (including any VAT).
`StartDate`	Starting date for the mandate.
`ExpirationDate`	Deadline for the deal.

Semantics of the `MandateOfferEntityResponse`:

Name	*Definition*
`MandateOfferEntityResponseType`	Abstract top-level type for confirming the mandate. `MandateOfferEntityResponseType` extends `ProtocolResponseType`.
`MandateOfferEntitySuccess`	A mandate confirmation.
`MandateOfferEntityFailure`	A mandate refusal.

Semantics of the `MandateOfferEntitySuccessType`:

Name	*Definition*
`EntityOrderInformation`	Information about the order as specified in 5.2.3.20, including order ID, transacted item, buyer, and price.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.
`BuyerAccount`	Information on buyer's account.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`SellerTESP`	Information on seller's TESP.

5.21.1.7 Mandate Revoke Entity

5.21.1.7.1 Introduction

When Buyer wants to revoke a debit mandate, the following messages are exchanged between Buyer and relevant users.

A message of type `MandateRevokeEntityRequestType` is sent by Buyer to Buyer's TESP.

A message of type `MandateRevokeEntityResponseType` is sent by Buyer's TESP to Buyer.

5.21.1.7.2 Syntax of protocol data format

```
<!-- Definition of MandateRevokeEntityRequest -->
<complexType name="MandateRevokeEntityRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="MandateID" type="string"/>
          <element name="RevocationDate" type="dateTime" minOccurs="0"/>
          <element name="RevocationReason" type="mpegmb:RevocationReasonType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of MandateRevokeEntityResponse -->
  <complexType name="MandateRevokeEntityResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <element name="MandateID" type="string"/>
          <choice>
            <element name="MandateRevokeEntitySuccess"
type="mpegmb:ProtocolSuccessType"/>
            <element name="MandateRevokeEntityFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.21.1.7.3 Semantics of protocol data format

Semantics of the `MandateRevokeEntityRequest`:

Name	*Definition*
`MandateRevokeEntityRequestType`	Top-level type for revoking a debit mandate. `MandateRevokeEntityRequestType` extends `ProtocolRequestType`.
`MandateID`	A reference to the Mandate agreement.
`RevocationDate`	Date when the mandate shall be revoked.
`RevocationReason`	Reason for the revocation as defined in 5.2.3.19.

Semantics of the `MandateRevokeEntityResponse`:

Name	*Definition*
`MandateRevokeEntityResponseType`	Top-level type for confirming the revocation of the mandate. `MandateRevokeEntityResponseType` extends `ProtocolResponseType`.
`MandateID`	A reference to the Mandate agreement.
`MandateRevokeEntitySuccess`	A mandate revocation confirmation.
`MandateRevokeEntityFailure`	A mandate revocation failure.

5.21.1.8 Extension to SID

For the representation of Service Instance Declarations of *Transact Entity*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="TransactEntitySIDType" abstract="true">
    <complexContent>
      <extension base="sid:ServiceInstanceDeclarationType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.21.2 Transact Content

5.21.2.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Transact Content elementary service.

This Elementary Service enables Users to interface with Payment and Cashing systems with regard to Contents.

The Transact Content elementary service is based on the abstract Transact Entity elementary service defined in 5.21.1.

The item of the transaction is a Digital Item represented as `mpegmb:ContentEntityType`.

5.21.2.2 Interfaces and protocol specification

The Transact Content Protocol extends the abstract Transact Entity Protocol defined in 5.21.1.3 from which it inherits the protocol specification.

5.21.2.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--                Transact Content                  -->
  <!-- ################################################ -->
  <!-- Definition of SellerPaymentOrderContentRequest -->
  <element name="SellerPaymentOrderContentRequest"
type="mpegm:SellerPaymentOrderContentRequestType"/>
  <complexType name="SellerPaymentOrderContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of SellerPaymentOrderContentResponse -->
  <element name="SellerPaymentOrderContentResponse"
type="mpegm:SellerPaymentOrderContentResponseType"/>
  <complexType name="SellerPaymentOrderContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="SellerPaymentOrderContentSuccess"
type="mpegm:SellerPaymentOrderContentSuccessType"/>
            <element name="SellerPaymentOrderContentFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
```

```
  </complexType>
  <complexType name="SellerPaymentOrderContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of BuyerPaymentOrderContentRequest -->
  <element name="BuyerPaymentOrderContentRequest"
type="mpegm:BuyerPaymentOrderContentRequestType"/>
  <complexType name="BuyerPaymentOrderContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of BuyerPaymentOrderContentResponse -->
  <element name="BuyerPaymentOrderContentResponse"
type="mpegm:BuyerPaymentOrderContentResponseType"/>
  <complexType name="BuyerPaymentOrderContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="BuyerPaymentOrderContentSuccess"
type="mpegm:BuyerPaymentOrderContentSuccessType"/>
            <element name="BuyerPaymentOrderContentFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="BuyerPaymentOrderContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
```

```
    </complexContent>
  </complexType>

  <!-- Definition of MandateOfferContentRequest -->
  <element name="MandateOfferContentRequest"
type="mpegm:MandateOfferContentRequestType"/>
  <complexType name="MandateOfferContentRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateOptions" type="mpegm:MandateOptionsType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of MandateOfferContentResponseType -->
  <element name="MandateOfferContentResponse"
type="mpegm:MandateOfferContentResponseType"/>
  <complexType name="MandateOfferContentResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="MandateOfferContentSuccess"
type="mpegm:MandateOfferContentSuccessType"/>
            <element name="MandateOfferContentFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="MandateOfferContentSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="ContentOrderInformation"
type="mpegmb:ContentOrderInformationType"/>
          <element name="MandateID" type="string"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<element name="MandateRevokeContentRequest"
type="mpegm:MandateRevokeEntityRequestType"/>
<element name="MandateRevokeContentResponse"
type="mpegm:MandateRevokeEntityResponseType"/>
```

5.21.2.4 Semantics of protocol data format

Semantics of the `SellerPaymentOrderContentRequest`:

Name	*Definition*
`SellerPaymentOrderContentRequest`	The message is sent to inform Seller's TESP of the beginning of a transaction.
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall not be set.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

Semantics of the `SellerPaymentOrderContentResponse`:

Name	*Definition*
`SellerPaymentOrderContentResponse`	The message is sent as response from Seller's TESP to Seller.
`SellerPaymentOrderContentSuccess`	Message in case of success. Contains a payment confirmation.
`SellerPaymentOrderContentFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `SellerPaymentOrderContentSuccessType`:

Name	*Definition*
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price.
`BuyerAccount`	Account from which payment will be effected.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

Semantics of the `BuyerPaymentOrderContentRequest`:

Name	*Definition*
`BuyerPaymentOrderContentRequest`	The message is sent from Buyer's TESP to the Buyer.

Name	*Definition*
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall be set.
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`SellerPCSP`	Information on seller's TESP.

Semantics of the `BuyerPaymentOrderContentResponse`:

Name	*Definition*
`BuyerPaymentOrderContentResponse`	The message is sent as response to a Buyer Payment Order.
`BuyerPaymentOrderContentSuccess`	Message part in case of success. Contains a payment confirmation.
`BuyerPaymentOrderContentFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `BuyerPaymentOrderContentSuccessType`:

Name	*Definition*
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller to buyer.

Semantics of the `MandateOfferContentRequest`:

Name	*Definition*
`MandateOfferContentRequest`	The message is sent from Seller to Seller's TESP.
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price.

Name	*Definition*
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payments will be credited.
`MandateOptions`	Information on mandate.

Semantics of the `MandateOfferContentResponse`:

Name	*Definition*
`MandateOfferContentResponse`	The message is sent from Seller's TESP to Seller in response of a MandateOffer.
`MandateOfferContentSuccess`	A mandate confirmation.
`MandateOfferContentFailure`	A mandate refusal.

Semantics of the `MandateOfferContentSuccessType`:

Name	*Definition*
`ContentOrderInformation`	Information about the content order as specified in 5.2.3.21, including order ID, transacted content, buyer, and price.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.
`BuyerAccount`	Information on buyer's account.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`SellerTESP`	Information on seller's TESP.

Semantics of the `MandateRevokeContentRequest`:

Name	Definition
`MandateRevokeContentRequest`	The message is sent to revoke a mandate.

Semantics of the `MandateRevokeContentResponse`:

Name	*Definition*
`MandateRevokeContentResponse`	The message is sent in response of a MandateRevoke.

5.21.2.5 Extension to SID

For the representation of Service Instance Declarations of *Transact Content*, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="TransactContentSIDType">
    <complexContent>
      <extension base="sid:TransactEntitySIDType"/>
      <!-- No additional elements needed in this extension. -->
    </complexContent>
  </complexType>
```

5.21.3 Transact License

5.21.3.1 Introduction

This subclause specifies interfaces, protocol specifications as well as syntax and semantics of the protocol data formats of the Transact License elementary service.

This Elementary Service enables Users to interface with payment and cashing systems with regard to Licenses. An example of use of this Elementary Service is to provide a User the ability to transact Content to another User.

The Transact License elementary service is based on the abstract Transact Entity elementary service defined in 5.21.1.

The item of the transaction is a Digital Item represented as `mpegmb:LicenseEntityType`.

5.21.3.2 Interfaces and protocol specification

The Transact License Protocol extends the abstract Transact Entity Protocol defined in 5.21.1.3 from which it inherits the protocol specification.

5.21.3.3 Syntax of protocol data format

```
  <!-- ################################################ -->
  <!--                Transact License                  -->
  <!-- ################################################ -->
  <!-- Definition of SellerPaymentOrderLicenseRequest -->
  <element name="SellerPaymentOrderLicenseRequest"
type="mpegm:SellerPaymentOrderLicenseRequestType"/>
  <complexType name="SellerPaymentOrderLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
```

```
  </complexType>

  <!-- Definition of SellerPaymentOrderLicenseResponse -->
  <element name="SellerPaymentOrderLicenseResponse"
type="mpegm:SellerPaymentOrderLicenseResponseType"/>
  <complexType name="SellerPaymentOrderLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="SellerPaymentOrderLicenseSuccess"
type="mpegm:SellerPaymentOrderLicenseSuccessType"/>
            <element name="SellerPaymentOrderLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="SellerPaymentOrderLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateID" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of BuyerPaymentOrderLicenseRequest -->
  <element name="BuyerPaymentOrderLicenseRequest"
type="mpegm:BuyerPaymentOrderLicenseRequestType"/>
  <complexType name="BuyerPaymentOrderLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of BuyerPaymentOrderLicenseResponse -->
  <element name="BuyerPaymentOrderLicenseResponse"
type="mpegm:BuyerPaymentOrderLicenseResponseType"/>
  <complexType name="BuyerPaymentOrderLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="BuyerPaymentOrderLicenseSuccess"
type="mpegm:BuyerPaymentOrderLicenseSuccessType"/>
```

```
            <element name="BuyerPaymentOrderLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="BuyerPaymentOrderLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of MandateOfferLicenseRequest -->
  <element name="MandateOfferLicenseRequest"
type="mpegm:MandateOfferLicenseRequestType"/>
  <complexType name="MandateOfferLicenseRequestType">
    <complexContent>
      <extension base="mpegmb:ProtocolRequestType">
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="BuyerTESP" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="MandateOptions" type="mpegm:MandateOptionsType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Definition of MandateOfferLicenseResponseType -->
  <element name="MandateOfferLicenseResponse"
type="mpegm:MandateOfferLicenseResponseType"/>
  <complexType name="MandateOfferLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
            <element name="MandateOfferLicenseSuccess"
type="mpegm:MandateOfferLicenseSuccessType"/>
            <element name="MandateOfferLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="MandateOfferLicenseSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
```

```
        <sequence>
          <element name="LicenseOrderInformation"
type="mpegmb:LicenseOrderInformationType"/>
          <element name="MandateID" type="string"/>
          <element name="BuyerAccount" type="mpeg7:UniqueIDType"/>
          <element name="Seller" type="mpegmb:UserEntityType"/>
          <element name="SellerAccount" type="mpeg7:UniqueIDType"/>
          <element name="SellerTESP" type="mpeg7:UniqueIDType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

<element name="MandateRevokeLicenseRequest"
type="mpegm:MandateRevokeEntityRequestType"/>
<element name="MandateRevokeLicenseResponse"
type="mpegm:MandateRevokeEntityResponseType"/>
```

5.21.3.4 Semantics of protocol data format

Semantics of the `SellerPaymentOrderLicenseRequest`:

Name	*Definition*
`SellerPaymentOrderLicenseRequest`	The message is sent to inform Seller's TESP of the beginning of a transaction.
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall not be set.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

Semantics of the `SellerPaymentOrderLicenseResponse`:

Name	*Definition*
`SellerPaymentOrderLicenseResponse`	The message is sent as response from Seller's TESP to Seller.
`SellerPaymentOrderLicenseSuccess`	Message in case of success. Contains a payment confirmation.
`SellerPaymentOrderLicenseFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `SellerPaymentOrderLicenseSuccessType`:

Name	*Definition*
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price.
`BuyerAccount`	Account from which payment will be effected.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.

Semantics of the `BuyerPaymentOrderLicenseRequest`:

Name	*Definition*
`BuyerPaymentOrderLicenseRequest`	The message is sent from Buyer's TESP to the Buyer.
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price. NOTE The `TransactionDate` shall not be set since the transaction has not been completed. NOTE The `Amount` shall be set.
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`SellerPCSP`	Information on seller's TESP.

Semantics of the `BuyerPaymentOrderLicenseResponse`:

Name	*Definition*
`BuyerPaymentOrderLicenseResponse`	The message is sent as response to a Buyer Payment Order.
`BuyerPaymentOrderLicenseSuccess`	Message part in case of success. Contains a payment confirmation.
`BuyerPaymentOrderLicenseFailure`	Message part in case of failure. Contains the reason why the transaction was not completed.

Semantics of the `BuyerPaymentOrderLicenseSuccessType`:

Name	*Definition*
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price.
`BuyerTESP`	Information on buyer's TESP.

Name	*Definition*
`Seller`	Identifier of the seller.
`SellerAccount`	Account to which payment will be credited.
`MandateID`	Optional identifier of a mandate given to seller to buyer.

Semantics of the `MandateOfferLicenseRequest`:

Name	*Definition*
`MandateOfferLicenseRequest`	The message is sent from Seller to Seller's TESP.
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price.
`BuyerTESP`	Information on buyer's TESP.
`Seller`	Information on seller.
`SellerAccount`	Account to which payments will be credited.
`MandateOptions`	Information on mandate.

Semantics of the `MandateOfferLicenseResponse`:

Name	*Definition*
`MandateOfferLicenseResponse`	The message is sent from Seller's TESP to Seller in response of a MandateOffer.
`MandateOfferLicenseSuccess`	A mandate confirmation.
`MandateOfferLicenseFailure`	A mandate refusal.

Semantics of the `MandateOfferLicenseSuccessType`:

Name	*Definition*
`LicenseOrderInformation`	Information about the license order as specified in 5.2.3.22, including order ID, transacted license, buyer, and price.
`MandateID`	Optional identifier of a mandate given to seller by the buyer.
`BuyerAccount`	Information on buyer's account.
`Seller`	Information on seller.
`SellerAccount`	Account to which payment will be credited.
`SellerTESP`	Information on seller's TESP.

Semantics of the `MandateRevokeLicenseRequest`:

Name	*Definition*
`MandateRevokeLicenseRequest`	The message is sent to revoke a mandate.

Semantics of the `MandateRevokeLicenseResponse`:

Name	*Definition*
`MandateRevokeLicenseResponse`	The message is sent in response of a MandateRevoke.

5.21.3.5 Extension to SID

For the representation of Service Instance Declarations of *Transact License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="TransactLicenseSIDType">
  <complexContent>
    <extension base="sid:TransactEntitySIDType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```


5.22 The Verify Services

5.22.1 Verify Contract

5.22.1.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Verify Contract elementary service.

The Verify Contract Service allows Users to check the validity of a Contract in a multimedia content value chain. The Service Provider of this Service verifies the validity of the Contract that is passed.

5.22.1.2 Interfaces and protocol specification

The Verify Contract Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `VerifyContractRequest` message. The message contains the contract.	
2.		The SP performs verification of the contract. The SP then sends a `VerifyContractResponse`.

5.22.1.3 Syntax of protocol data format

```
<element name="VerifyContractRequest" type="mpegm:VerifyContractRequestType"/>
<complexType name="VerifyContractRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="ContractEntity" type="mpegmb:ContractEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

 <element name="VerifyContractResponse" type="mpegm:VerifyContractResponseType"/>
  <complexType name="VerifyContractResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <choice>
          <element name="VerifyContractSuccess"
type="mpegm:VerifyContractSuccessType"/>
          <element name="VerifyContractFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="VerifyContractSuccessType">
    <complexContent>
      <extension base="mpegmb:ProtocolSuccessType">
        <sequence>
          <element name="VerifyContractExplanation"
type="mpegm:VerifyContractExplanationType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <simpleType name="VerifyContractExplanationType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="OK"/>
          <enumeration value="ISSUER_IS_NOT_RIGHTS_OWNER"/>
          <enumeration value="CONTRACT_INCONSISTENT"/>
          <enumeration value="CONTRACT_WITH_BAD_REFERENCES"/>
          <enumeration value="UNKNOWN_ERROR"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>
```

5.22.1.4 Semantics of protocol data format

Semantics of the `VerifyContractRequest`:

Name	*Definition*
`VerifyContractRequest`	Message to request the verification of a contract.
`ContractEntity`	A MPEG-21 CEL contract as described in 5.2.3.10.

Semantics of the `VerifyContractResponse`:

Name	*Definition*
`VerifyContractResponse`	Message to respond the verification of one contract.
`VerifyContractSuccess`	Response in case of success, an element of type `mpegm:VerifyContractSuccessType`. This states that the contract was verified.
`VerifyContractFailure`	Response in case of failure, an element of type `mpegmb:ProtocolFailureType`.

Semantics of the `VerifyContractSuccessType`:

Name	*Definition*
`VerifyContractSuccessType`	Type of the response message part that is provided in case of success. NOTE The `VerifyContractSuccess` element only means that the protocol was carried out successfully. The result of the contract verification is indicated in the `VerifyContractExplanation` element.
`VerifyContractExplanation`	Information about the verification result. Possible values are shown in Table 12. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.
`VerifyContractExplanationType`	Type to convey the specific information about the result of the verification.

Table 12 — Possible values of the `VerifyContractExplanationType`

Protocol Code	Definition
OK	The contract verification was positive.
ISSUER_IS_NOT_RIGHTS_OWNER	The Issuer of the contract has no rights over the licenses content or service.
CONTRACT_INCONSISTENT	Contradictions found in the contract, or any other inconsistency.
CONTRACT_WITH_BAD_REFERENCES	Either the content, the issuer, or the second party could not be found.
UNKNOWN_ERROR	The contract verification was negative for an unspecified reason.

5.22.1.5 Extension to SID

For the representation of Service Instance Declarations of *Verify Contract*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="VerifyContractSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.22.2 Verify Device

5.22.2.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Verify Device elementary service.

This Protocol specifies how a MPEG-M Device can be verified from a Verify DeviceSP. This protocol is based on the Remote Attestation technique. This can be achieved employing any hardware provided with advanced security features, such as TPM (Trusted Platform Module).

5.22.2.2 Interfaces and protocol specification

The Verify Device Protocol is as follows:

Steps	Client	Service Provider
1.	The client performs a set of measures, generates a `VerifyDeviceRequest` message and sends it to the Service Provider.	
2.		The SP verifies the digital signature of the message if present.

Steps	Client	Service Provider
3.		The SP extracts the information from the message and decides whether to accept the request. In both cases the SP generates an `VerifyDeviceResponse` message containing either a `VerifyDeviceResponseSuccess` element indicating the successful completion of the Protocol or a `VerifyDeviceResponseFailure` element indicating the reason of failure.

5.22.2.3 Syntax of protocol data format

```
<!-- Definition of VerifyDeviceRequest -->
<element name="VerifyDeviceRequest" type="mpegm:VerifyDeviceRequestType"/>
<complexType name="VerifyDeviceRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="PCRs" type="base64Binary"/>
        <element name="Sign" type="base64Binary"/>
        <element name="AttestationIdentityKey" type="dsig:KeyInfoType"/>
        <element name="Nonce" type="base64Binary" minOccurs="0"/>
        <element name="MeasurementLog" type="mpegm:MeasurementLogType"
minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Definition of MeasurementLogType -->
<complexType name="MeasurementLogType">
  <choice>
    <element name="MeasurementLogBlob" type="base64Binary"/>
    <element name="MeasurementLogTSS" type="mpegm:MeasurementLogTSSType"/>
  </choice>
</complexType>

<complexType name="MeasurementEventType">
  <complexContent>
    <element name="EventPCRIndex" type="integer"/>
    <element name="EventClass" type="mpegm:EventClassType"/>
    <element name="EventValue" type="base64Binary"/>
    <element name="EventPCRValue" type="base64Binary"/>
  </complexContent>
</complexType>

<complexType name="MeasurementLogTSSType">
  <sequence maxOccurs="unbounded">
    <element name="MeasurementEvent" type="mpegm:MeasurementEventType"/>
  </sequence>
</complexType>

<!-- Definition of EventClassType -->
  <simpleType name="EventClassType">
    <union>
```

```
      <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="LIBRARY"/>
          <enumeration value="CONFIG_FILE"/>
          <enumeration value="EXECUTABLE"/>
          <enumeration value="SCRIPT"/>
          <enumeration value="DISC_IMAGE"/>
          <enumeration value="USER_INPUT"/>
          <enumeration value="CRYPTOGRAPHIC_MATERIAL"/>
          <enumeration value="GENERIC_FILE"/>
          <enumeration value="GENERIC_DATA"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="mpeg7:termReferenceType"/>
      </simpleType>
    </union>
  </simpleType>

<!-- Definition of VerifyDeviceResponse -->
<element name="VerifyDeviceResponse" type="mpegm:VerifyDeviceResponseType"/>
<complexType name="VerifyDeviceResponseType">
  <complexContent>
    <extension base="mpegmb:ProtocolResponseType">
      <sequence>
        <choice>
          <element name="VerifyDeviceResponseSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="VerifyDeviceResponseFailure"
type="mpegmb:ProtocolFailureType"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```


5.22.2.4 Semantics of protocol data format

Semantics of the `VerifyDeviceRequest`:

Name	*Definition*
`VerifyDeviceRequest`	The message for creating a request for Attestation.
`VerifyDeviceRequestType`	Top-level type for `VerifyDeviceRequest`. `VerifyDeviceRequestType` extends `ProtocolRequestType`.
`PCRs`	The value of the signed Platform Configuration Registers (PCR) (the serialization of the TCPA_PCR_INFO Trusted Software Stack (TSS) data structure [4]) in the format specified, according to Table 13.

Name	*Definition*
`Sign`	The signature made over the PCRs by the Trusted Platform Module (TPM). This is the result of the Tspi_TPM_Quote or Tspi_TPM_Quote2 functions.
`AttestationIdentityKey`	The TPM key (AIK) used for computing the signature over the PCRs included its certificate chain.
`Nonce`	A nonce used for ensuring signature's freshness. This is used by the Tspi_TPM_Quote or Tspi_TPM_Quote2 functions.
`MeasurementLog`	The list of logs related to the PCR values of type `MeasurementLogType`.

Semantics of the `MeasurementLogType`:

Name	*Definition*
`MeasurementLogBlob`	A Base64 binary representation of the measurement.
`MeasurementLogTSS`	A `MeasurementLogTSSType` element representing the measurements log according TSS specification. This is a list of measurements where each element corresponds to a TSS_PCR_EVENT TSS data structure.

Semantics of the `MeasurementLogTSSType`:

Name	*Definition*
`MeasurementEvent`	The set of the measurement step.

Semantics of the `MeasurementEventType`:

Name	*Definition*
`EventPCRIndex`	The identification number of the PCR.
`EventClass`	The class of the event being measured.
`EventValue`	A blob which defines the event (it is left to the application to define this blob, e.g., the name of a measured file).
`EventPCRValue`	The value computed by the TSS after the PCR extension.

Semantics of the `EventClassType`:

Name	*Definition*
`EventClassType`	The class of event being measured. Possible values are defined in Table 13 . Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

Semantics of the `VerifyDeviceResponse`:

Name	*Definition*
`VerifyDeviceResponse`	The message conveys the Result attribute indicating whether the Protocol was successful or not.
`VerifyDeviceResponseType`	Top-level type for `VerifyDeviceResponse`. `VerifyDeviceResponseType` extends `ProtocolResponseType`.
`VerifyDeviceResponseSuccess`	Response in case of success.
`VerifyDeviceResponseFailure`	Response in case of failure.

Table 13 provides the TCPA_PCR_INFO TSS data structure format:

Table 13 — TCPA_PCR_INFO TSS data structure format

Field name	Field length
mask_size	u16
mask	mask_size
pcrs_size	u32
pcrs	pcrs_size

5.22.2.5 Extension to SID

For the representation of Service Instance Declarations of *Verify Device*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="VerifyDeviceSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

5.22.3 Verify License

5.22.3.1 Introduction

This subclause specifies interfaces, protocol specification as well as syntax and semantics of the protocol data formats of the Verify License elementary service.

The Verify License Service allows Users to check the validity of a License in a multimedia content value chain.

5.22.3.2 Interfaces and protocol specification

The Verify License Protocol is as follows:

Steps	Client	Service Provider
1.	The client sends a `VerifyLicenseRequest` message. The message contains the license.	
2.		The SP performs verification of the license. The SP then sends a `VerifyLicenseResponse`.

5.22.3.3 Syntax of protocol data format

```
<element name="VerifyLicenseRequest" type="mpegm:VerifyLicenseRequestType"/>
<complexType name="VerifyLicenseRequestType">
  <complexContent>
    <extension base="mpegmb:ProtocolRequestType">
      <sequence>
        <element name="LicenseEntity" type="mpegmb:LicenseEntityType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="VerifyLicenseResponse" type="mpegm:VerifyLicenseResponseType"/>
  <complexType name="VerifyLicenseResponseType">
    <complexContent>
      <extension base="mpegmb:ProtocolResponseType">
        <sequence>
          <choice>
          <element name="VerifyLicenseSuccess"
type="mpegmb:ProtocolSuccessType"/>
          <element name="VerifyLicenseFailure"
type="mpegmb:ProtocolFailureType"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

5.22.3.4 Semantics of protocol data format

Semantics of the `VerifyLicenseRequest`:

Name	*Definition*
`VerifyLicenseRequest`	Message to request the verification of a license.
`LicenseEntity`	A MPEG-21 REL License to be verified.

Semantics of the `VerifyLicenseResponse`:

Name	*Definition*
`VerifyLicenseResponseType`	Message sent as a result of the Verify service invocation.
`VerifyLicenseSuccess`	Message part in case of success.
`VerifyLicenseFailure`	Message part in case of failure.

5.22.3.5 Extension to SID

For the representation of Service Instance Declarations of *Verify License*, the following complex type is defined in the *SID XML Schema*:

```
<!-- NOTE: This extension is defined in the SID XML Schema. -->

<complexType name="VerifyLicenseSIDType">
  <complexContent>
    <extension base="sid:ServiceInstanceDeclarationType"/>
    <!-- No additional elements needed in this extension. -->
  </complexContent>
</complexType>
```

Annex A
(normative)

Classification Schemes for IPTV offering discovery sections

A.1 General

This Annex defines MPEG-7 Classification Schemes that are used by Request Metadata steps of Request Content (specified in 5.17.1).

A.2 Classification Scheme for ETSI IPTV (DVB-IP) offering discovery sections

IPTV standards of the European Telecommunications Standards Institute (ETSI) are developed by the Digital Video Broadcasting Project (DVB) and usually referred to as DVB-IP. This subclause provides a Classification Scheme that describes the metadata sections of DVB-IP offering discovery metadata.

NOTE The terms defined in this Classification Scheme and the payload IDs mentioned therein correspond to definitions in ETSI TS 102 034[9], Table 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<mpeg7:Mpeg7 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004
http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
7_schema_files/mpeg7-v2.xsd">
  <mpeg7:Description xsi:type="mpeg7:ClassificationSchemeDescriptionType">
    <!-- ********************************************************** -->
    <!--                  ETSIOfferingDiscoverySectionsCS           -->
    <!-- ********************************************************** -->
    <mpeg7:ClassificationScheme uri="urn:mpeg:mpegM:cs:02-iptvods-
NS:2012:ETSIOfferingDiscoverySectionsCS">
      <mpeg7:Term termID="1">
        <mpeg7:Name xml:lang="en">ServiceProviderDiscovery</mpeg7:Name>
        <mpeg7:Definition xml:lang="en"> Indicates the metadata for service
provider discovery information (payload ID 0x01). The information includes a list
of IDs of metadata sections (or segments in ETSI IPTV) for each offering type
(described below) of each IPTV provider. If there is only one metadata section of
this type, it will contain metadata for all providers. If there are multiple
metadata sections of this type, each section will contain the metadata for one
provider and be identified by a TextualID representing the provider's domain
name.
        </mpeg7:Definition>
      </mpeg7:Term>
      <mpeg7:Term termID="2">
        <mpeg7:Name xml:lang="en">ServiceProvider</mpeg7:Name>
        <mpeg7:Definition xml:lang="en"> Indicates all offering discovery
metadata of an IPTV service provider. Each provider is identified by a TextualID
representing its domain name.
        </mpeg7:Definition>
          <mpeg7:Term termID="2.1">
            <mpeg7:Name xml:lang="en">BroadcastService</mpeg7:Name>
```

```
            <mpeg7:Definition xml:lang="en"> Indicates metadata for broadcast
offering discovery of an IPTV service provider (payload ID 0x02). Metadata of
this type could be divided into sections, identified by NumericIDs.
          </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.2">
          <mpeg7:Name xml:lang="en">CoDService</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata for Content-on-
Demand offering discovery of an IPTV service provider (payload ID 0x03). Metadata
of this type could be divided into sections, identified by NumericIDs.
          </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.3">
          <mpeg7:Name xml:lang="en">ServicesFromOtherSP</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata for referenced
offering discovery at an IPTV service provider (payload ID 0x04). Metadata of
this type could be divided into sections, identified by NumericIDs.
          </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.4">
          <mpeg7:Name xml:lang="en">PackageService</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata for package
offering discovery of an IPTV service provider (payload ID 0x05). Metadata of
this type could be divided into sections, identified by NumericIDs.
          </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.5">
          <mpeg7:Name xml:lang="en">BCGService</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata for BCG (Broadband
Content Guide) offering discovery of an IPTV service provider (payload ID 0x06).
Metadata of this type could be divided into sections, identified by NumericIDs.
          </mpeg7:Definition>
        </mpeg7:Term>
      </mpeg7:Term>
    </mpeg7:ClassificationScheme>
  </mpeg7:Description>
</mpeg7:Mpeg7>
```

A.3 Classification Scheme for ATIS IIF offering discovery sections

The IPTV Interoperability Forum (IIF) of the Alliance for Telecommunication Industry Solutions (ATIS) is in charge of developing standards and specifications to enable an end-to-end solution for Internet Protocol Television (IPTV). This subclause provides a Classification Scheme that describes the metadata sections of ATIS IIF offering discovery metadata.

NOTE The terms defined in this Classification Scheme correspond to the definitions in ATIS-0800022[10].

```
<?xml version="1.0" encoding="UTF-8"?>
<mpeg7:Mpeg7 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004
http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
7_schema_files/mpeg7-v2.xsd">
  <mpeg7:Description xsi:type="mpeg7:ClassificationSchemeDescriptionType">
```

```
    <!-- ************************************************************** -->
    <!--                    ATISOfferingDiscoverySectionsCS             -->
    <!-- ************************************************************** -->
    <mpeg7:ClassificationScheme uri="urn:mpeg:mpegM:cs:02-iptvods-
NS:2012:ATISOfferingDiscoverySectionsCS">
      <mpeg7:Term termID="1">
        <mpeg7:Name xml:lang="en">ServiceProviderInfo</mpeg7:Name>
        <mpeg7:Definition xml:lang="en">Indicates the metadata of ATIS IIF
ServiceProviderInfoTableType which provides information about different IPTV
service providers. There is only one metadata section (or record in ATIS IIF) of
this type.</mpeg7:Definition>
      </mpeg7:Term>
      <mpeg7:Term termID="2">
        <mpeg7:Name xml:lang="en">ServiceProvider</mpeg7:Name>
        <mpeg7:Definition xml:lang="en">Indicates all offering discovery metadata
of an IPTV Service Provider. Each service provider is identified by a TextualID
representing its identifier.</mpeg7:Definition>
        <mpeg7:Term termID="2.1">
          <mpeg7:Name xml:lang="en">ProvisioningInfo</mpeg7:Name>
          <mpeg7:Definition xml:lang="en">Indicates metadata of ATIS IIF
ProvisioningInfoType which provides provisioning information from an IPTV service
provider. There is only one metadata section of this type. </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.2">
          <mpeg7:Name xml:lang="en">Master SI Table</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata of ATIS IIF
MasterSiTableType which is a list of virtual channel maps of a given IPTV service
provider. There is only one metadata section of this type. </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.3">
          <mpeg7:Name xml:lang="en">Virtual Channel Map</mpeg7:Name>
          <mpeg7:Definition xml:lang="en">Indicates metadata of ATIS IIF
VirtualChannelMapType which is a list of virtual channels. Metadata of this type
could be divided into sections/records, each is identified by a TextualID
representing its URI. </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.4">
          <mpeg7:Name xml:lang="en">Virtual Channel Description</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata of ATIS IIF
VirtualChannelDescriptionTableType which is a description of virtual channels.
Metadata of this type could be divided into sections/records, each is identified
by a textualID representing its URI. </mpeg7:Definition>
        </mpeg7:Term>
        <mpeg7:Term termID="2.5">
          <mpeg7:Name xml:lang="en">Source</mpeg7:Name>
          <mpeg7:Definition xml:lang="en"> Indicates metadata of ATIS IIF
SourceTableType which shows acquisition information for virtual channels.
Metadata of this type could be divided into sections/records, each is identified
by a TextualID representing its URI. </mpeg7:Definition>
        </mpeg7:Term>
      </mpeg7:Term>
    </mpeg7:ClassificationScheme>
  </mpeg7:Description>
</mpeg7:Mpeg7>
```

Annex B
(normative)

Service Types for Process Content

B.1 General

This Annex defines Service Types for the Process Content service specified in 5.16.1.

B.2 Recognize Speech

B.2.1 Introduction

This subclause specifies the Recognize Speech Service Type for the Process Content service.

The Recognize Speech Service Type is employed to request to recognize user's speech input and produce output text. This type specifies how to request speech recognition of the user's speech. This type is applied between a User who speaks to a microphone or a remote control, and Recognize Speech Service Provider. This Service Type provides accessibility for people with disability or old people with difficulty to handle devices, as well as convenient user interface to other users.

The following diagrams explain Speech Recognition method in two ways. Both cases are reflected in the schema of Recognize Speech Service type.

Case 1: The speech signal is sent to the SP without any preprocessing as shown in Figure B.1.

Figure B.1 — Speech signal sent from User to SP

Case2: The speech features extracted and compressed from the speech signal is sent to the SP as shown in Figure B.2.

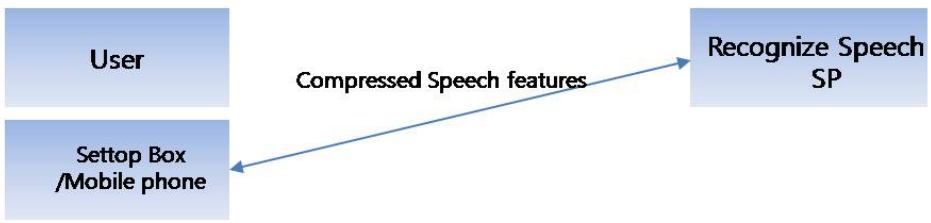

Figure B.2 — Compressed speech features sent to the SP

B.2.2 Syntax of Service Type data format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004" xmlns:rs="urn:mpeg:mpegM:processing-
type:01-recognize-speech-NS:2012" targetNamespace="urn:mpeg:mpegM:service-
type:01-recognize-speech-NS:2012" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="ISO/IEC 23006-4 2nd edition"
id="recognize-speech.xsd">
  <import namespace="urn:mpeg:mpegM:schema:01-base-NS:2012"
schemaLocation="../mpeg-m-base.xsd"/>
  <import namespace="urn:mpeg:mpeg7:schema:2004"
schemaLocation="http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
7_schema_files/mpeg7-v2.xsd"/>

  <!-- Definition of Recognize Speech Service Type -->
  <complexType name="RecognizeSpeechType">
    <complexContent>
      <extension base="mpegmb:RequestParametersBaseType">
        <sequence>
          <choice maxOccurs="unbounded">
            <element name="SpeechInput" type="mpegmb:ContentEntityType"/>
            <element name="SpeechFeature" type="mpeg7:AudioDType"/>
          </choice>
          <element name="Language" type="language" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

B.2.3 Semantics of Service Type data format

Semantics of the `RecognizeSpeechType`:

Name	*Definition*
`RecognizeSpeechType`	Request parameters type for the "Recognize Speech" Service Type. `RecognizeSpeechType` extends `mpegmb:ProtocolBaseType`. NOTE The `ContentEntity` element is not used in this Service Type.
`SpeechInput`	Speech signal spoken by the user.
`SpeechFeature`	Compressed speech features extracted from the speech signal.
`Language`	Indicates the language of the input speech. NOTE If present, the `Language` element shall take precedence over other language indications present within the speech input.

NOTE The recognized speech is provided in the `ProcessedContentEntity` element of the `ProcessContentCompletion` message.

B.2.4 ServiceType XML declaration

The XML declaration of the "Recognize Speech" Service Type is defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpegmb:ServiceType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:rs="urn:mpeg:mpegM:service-type:01-recognize-speech-NS:2012">

  <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:01-recognize-speech-
NS:2012</mpegmb:ServiceTypeIdentifier>
  <mpegmb:Name>Recognize Speech</mpegmb:Name>

<mpegmb:RequestParametersType>rs:RecognizeSpeechType</mpegmb:RequestParametersTyp
e>
  <!-- NOTE: This Service Type defines no extension for the
mpegmb:CompletionParametersType. -->
</mpegmb:ServiceType>
```

B.2.5 Example

This example shows the function of the Process Content service with the Service Type of Recognize Speech.

The client sends a `ProcessContentRequest` message to the SP, containing a Digital Item with the audio resources to be processed and the parameters specified by the Service Type.

```
<ProcessContentRequest immidiateResponse="true">
  <mpegmb:TransactionIdentifier>52</mpegmb:TransactionIdentifier>
  <ServiceTypeEntity>
    <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:01-recognize-
speech-NS:2012</mpegmb:ServiceTypeIdentifier>
  </ServiceTypeEntity>
  <ContentEntity>
    <didl:DIDL>
      <didl:Item>
        <didl:Component>
          <didl:Resource ref="http://example.com/my-speech.mp3"
mimeType="audio/mp3"/>
        </didl:Component>
      </didl:Item>
    </didl:DIDL>
  </ContentEntity>
  <RequestParameters type="rs:RecognizeSpeechType">
    <rs:Language>ko</rs:Language>
  </RequestParameters>
</ProcessContentRequest>
```

The SP concludes that the processing can be performed and sends an affirmative `ProcessContentResponse` message to the client.

```
<ProcessContentResponse>
  <mpegmb:TransactionIdentifier>52</mpegmb:TransactionIdentifier>
  <ProcessContentSuccess>
    <SuccessCode>ACCEPTED</SuccessCode>
  </ProcessContentSuccess>
</ProcessContentResponse>
```

After successfully creating a textual transcript of the audio content, the SP sends a `ProcessContentCompletion` message to the client.

```
<ProcessContentCompletion>
  <mpegmb:TransactionIdentifier>52</mpegmb:TransactionIdentifier>
  <ProcessContentCompletionSuccess>
    <ContentEntity>
      <didl:DIDL>
        <didl:Item>
          <didl:Component>
            <didl:Resource ref="http://example.com/your-transcript.txt"
mimeType="text/plain"/>
          </didl:Component>
        </didl:Item>
      </didl:DIDL>
    </ContentEntity>
  </ProcessContentCompletionSuccess>
</ProcessContentCompletion>
```

B.3 Synthesize Speech

B.3.1 Introduction

This subclause specifies the Synthesize Speech service type for the Process Content elementary service.

The Synthesize Speech service type is employed to request to produce output speech from the user input which is text. This service type specifies how to request speech synthesis of the text the user provides. This type is applied between a User inputs a text, and Synthesize Speech Service Provider. This Service Type provides accessibility for people with disability to speak, as well as convenient user interface to other users in the services such as machine announcement, answering services and automatic language translation.

B.3.2 Syntax of Service Type data format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
  xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
  xmlns:ss="urn:mpeg:mpegM:service-type:02-synthesize-speech-NS:2012"
  targetNamespace="urn:mpeg:mpegM:service-type:02-synthesize-speech-NS:2012"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="ISO/IEC 23006-4 2nd edition" id="Synthesize-speech.xsd">
  <import namespace="urn:mpeg:mpegM:schema:01-base-NS:2012"
schemaLocation="../mpeg-m-base.xsd"/>

  <!-- Definition of Synthesize Speech Service Type -->
  <complexType name="SynthesizeSpeechType">
    <complexContent>
      <extension base="mpegmb:RequestParametersBaseType">
        <sequence>
          <element name="Language" type="language" minOccurs="0"/>
          <element name="TaggedInput" type="mpeg7:TermUseType" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
        <attribute name="voiceGender" use="optional" default="unspecified">
          <simpleType>
```

```
            <restriction base="NMTOKEN">
              <enumeration value="female"/>
              <enumeration value="male"/>
              <enumeration value="neuter"/>
              <enumeration value="unspecified"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name="voicePitch" type="mpeg7:nonNegativeReal"/>
        <attribute name="voiceSpeed" type="mpeg7:nonNegativeReal"/>
        <attribute name="voiceName" type="string"/>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

B.3.3 Semantics of Service Type data format

Semantics of the `SynthesizeSpeechType`:

Name	*Definition*
`SynthesizeSpeechType`	Top-level type for `SynthesizeSpeechRequest`. It contains the necessary parameters for the Synthesize Speech Service Type. `SynthesizeSpeechType` extends `ProtocolBaseType`.
`Language`	Indicates the language of the input text. NOTE If present, the `Language` element shall take precedence over other language indications present within the input.
`TaggedInput`	Input tagged for specifying the voice features such as speed, pitch and emotion for parts of the input.
`voiceGender`	Gender of the output voice.
`voicePitch`	Pitch of the output voice in Hz.
`voiceSpeed`	Speed of the output voice in words per minute.
`voiceName`	Name of the output voice from a list provided by the Service Provider.

NOTE The synthesized speech is provided in the `ProcessedContentEntity` element of the `ProcessContentCompletion` message.

B.3.4 ServiceType XML declaration

The XML declaration of the "Synthesize Speech" Service Type is defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpegmb:ServiceType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:ss="urn:mpeg:mpegM:service-type:02-synthesize-speech-NS:2012">
```

```
  <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:02-synthesize-speech-
NS:2012</mpegmb:ServiceTypeIdentifier>
  <mpegmb:Name>Synthesize Speech</mpegmb:Name>

<mpegmb:RequestParametersType>ss:SynthesizeSpeechType</mpegmb:RequestParametersTy
pe>
  <!-- NOTE: This Service Type defines no extension for the
mpegmb:CompletionParametersType. -->
</mpegmb:ServiceType>
```

B.3.5 Example

This example shows the function of the Process Content service with the service type of `Synthesize Speech`.

For the actual processing, the client sends a `ProcessContentRequest` message to the SP, containing a text input to be processed and the parameters specified by the SP.

```
<ProcessContentRequest immediateResponse="true">
  <mpegmb:TransactionIdentifier>55</mpegmb:TransactionIdentifier>
  <ServiceTypeEntity>
    <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:02-synthesize-
speech-NS:2012</mpegmb:ServiceTypeIdentifier>
  </ServiceTypeEntity>
  <ContentEntity>
    <didl:DIDL>
      <didl:Item>
        <didl:Component>
          <didl:Resource ref="http://example.com/my-inputtext.txt"
mimeType="text/plain"/>
        </didl:Component>
      </didl:Item>
    </didl:DIDL>
  </ContentEntity>
  <RequestParameters type="ss:SynthesizeSpeechType" voiceGender="female"
voicePitch="190" voiceSpeed="200">
    <ss:Language>en-US</ss:Language>
  </RequestParameters>
</ProcessContentRequest>
```

The SP concludes that the processing can be performed and sends an affirmative `ProcessContentResponse` message to the client.

```
<ProcessContentResponse>
  <mpegmb:TransactionIdentifier>55</mpegmb:TransactionIdentifier>
  <ProcessContentSuccess>
    <SuccessCode>ACCEPTED</SuccessCode>
  </ProcessContentSuccess>
</ProcessContentResponse>
```

After successfully creating a speech output from the input content, the SP sends a `ProcessContentCompletion` message to the client.

```
<ProcessContentCompletion>
  <mpegmb:TransactionIdentifier>55</mpegmb:TransactionIdentifier>
  <ProcessContentCompletionSuccess>
```

```
    <ContentEntity>
      <didl:DIDL>
        <didl:Item>
          <didl:Component>
            <didl:Resource ref="http://example.com/your-transcript.mp3"
mimeType="audio/mp3"/>
          </didl:Component>
        </didl:Item>
      </didl:DIDL>
    </ContentEntity>
  </ProcessContentCompletionSuccess>
</ProcessContentCompletion>
```

B.4 Process Language

B.4.1 Introduction

This subclause specifies the Process Language Service Type for the Process Content service.

The Process Language Service Type is employed to request to produce tagged output from text input which is a natural language. This Service Type specifies how to request tagging of the text input according to the tagging type defined. The Process Language Service Type is employed between a User who provides the input text and a Process Language Service Provider which marks up words or phrases in the text corpus according to the specified tagging set. This Service Type is utilized for analyzing natural language text semantically and for extracting keywords from sentences. These keywords are used, in turn, in various situations when semantic text analysis is required, e.g., extracting social issues or emotion words from the social media text. Tagging for *Named Entity Recognition (NER)* such as person names, place names or organization names (cf. [14]) and *Part-of-speech (POS)* tagging for marking words in a text (such as nouns, verbs, adjective, etc.) are two important examples of the Process Language Service Type.

Available NER taggers comprise the Illinois NER system [15], Stanford Named Entity Recognizer [16], LingPipe Named Entity system [17], ETRI NE Tagger [18] and others.

Available POS taggers comprise Brill's tagger [19], Penn Treebank tagger [20], CLAWS [21], Stanford Log-linear Part-Of-Speech Tagger [22] and others (cf. [23]).

EXAMPLE 1 The input to a NER is a text, such as this one:

```
UN official Jim Croons heads for Bagdad.
```

And the output of the NER is an annotated block of text, such as this one:

```
UN/org official/job Jim Croons/person heads for Bagdad/location.
```

This sentence contains four named entities: `Jim Croons` is a person, `UN` is an organization, `official` is a job, and `Bagdad` is a location. Each NER system uses its own tagging set depending on the application which NER is required for.

EXAMPLE 2 The following example is for the case in the VOD service with Speech Interface. In the VOD service scenario in ISO/IEC 23006-6, the User is allowed to search and order a video by using keywords such as the movie title and/or the name of a person involved in the movie. By adopting the Process Language Service Type, the VOD service can process a User's order with full natural language text.

For example, the User input is:

```
I would like to see the movie Black Rain by the director Ridley Scott.
```

The output of Process Language would be:

```
Black Rain/movie_name + Ridley Scott/director_name
```

Based on this output, the VOD service will find a movie with the key "Black Rain/movie_name + Ridley Scott/director_name".

B.4.2 Syntax of Service Type data format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:pl="urn:mpeg:mpegM:service-type:05-process-language-NS:2012"
targetNamespace="urn:mpeg:mpegM:service-type:05-process-language-NS:2012"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="ISO/IEC 23006-4 2nd edition" id="Process-language.xsd">
  <import namespace="urn:mpeg:mpegM:schema:01-base-NS:2012"
schemaLocation="../mpeg-m-base.xsd"/>
  <import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="../mpeg/mpeg7-
v3.xsd"/>
  <!-- Definition of Process Language Service Type -->
  <complexType name=" ProcessLanguageType">
    <complexContent>
      <extension base="mpegmb:RequestParametersBaseType">
        <sequence>
          <element name="Language" type="language"/>
          <element name="TaggingType" type="mpeg7:ControlledTermUseType">
            <annotation>
              <documentation xml:lang="en">Suggested ClassificationScheme:
"urn:mpeg:mpegM:cs:03-es-NS:2012:TaggingTypeCS"</documentation>
            </annotation>
          </element>
          <element name="TaggingSet" type="mpeg7:InlineTermDefinitionType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```


B.4.3 Semantics of Service Type data format

Semantics of the `ProcessLanguageType`:

Name	*Definition*
`ProcessLanguageType`	Parameters for the `ProcessContentRequest` message of Service Type "Process Language". It contains the necessary parameters for natural language processing. `ProcessLanguageType` extends `mpegmb:RequestParametersBaseType`.
`Language`	Indicates the language of the input text.
`TaggingType`	Indicates the type of the tagging such as tagging Named Entity or tagging Part-of-Speech. An example of a Classification Scheme is the TaggingTypeCS (`urn:mpeg:mpegM:cs:03-es-NS:2012:TaggingTypeCS`) defined in G.5.
`TaggingSet`	Indicates the name of the tagging set for the tagging. The tagging set shall be specified via a URI (e.g., "http://www.cis.upenn.edu/~treebank/" for the Penn Treebank POS tagger).

NOTE The input to Process Language is provided in the `ContentEntity` element of the `ProcessContentRequest` message.

NOTE If a tagging set requires additional parameters (e.g., a *classifier* for the Stanford Named Entity Tagger), then those parameters shall be provided in the `mpegmb:Entry` or `mpegmb:ApplicationSpecificData` element of the `ProcessContentRequest` message.

NOTE The output of Process Language is provided in the `ProcessedContentEntity` element of the `ProcessContentCompletion` message.

B.4.4 Extension to SID

For the representation of Service Instance Declarations of the *Process Language* Service Type, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="STProcessLanguageSIDType">
    <complexContent>
      <extension base="sid:ProcessContentSIDType">
        <sequence>
          <element name="SupportedTaggingType"
type="mpeg7:InlineTermDefinitionType" minOccurs="0" maxOccurs="unbounded"/>
          <element name="SupportedTaggingSet"
type="mpeg7:InlineTermDefinitionType" minOccurs="0" maxOccurs="unbounded"/>
          <element name="SupportedLanguage" type="language" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:STProcessLanguageSIDType`:

Name	*Definition*
`sid:STProcessLanguageSIDType`	Top-level type for SIDs of the *Process Language* Service Type of the *Process Content* specific ES. `sid:STProcessLanguageSIDType` extends `sid:ProcessContentSIDType`.
`sid:SupportedTaggingType`	Optional list of tagging types that the SP supports.
`sid:SupportedTaggingSet`	Optional list of tagging sets that the SP supports.
`sid:SupportedTaggingType`	Optional list of language that the SP can process.

B.4.5 ServiceType XML declaration

The XML declaration of the "Process Language" Service Type is defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpegmb:ServiceType          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:pl="urn:mpeg:mpegM:service-type:05-process-language-NS:2012">
```

```
  <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:05-process-language-
NS:2012</mpegmb:ServiceTypeIdentifier>
  <mpegmb:Name>Process Language</mpegmb:Name>

<mpegmb:RequestParametersType>pl:ProcessLanguageType</mpegmb:RequestParametersTyp
e>
  <!-- NOTE: This Service Type defines no extension for the
mpegmb:CompletionParametersType. -->
</mpegmb:ServiceType>
```

B.4.6 Example

This example shows a protocol run of Process Content with the Service Type "Process Language".

The client sends a `ProcessContentRequest` message to the SP, containing a text input to be processed and the parameters for the tagging.

```
<mpegm:ProcessContentRequest immediateResponse="false">
  <mpegmb:TransactionIdentifier>77</mpegmb:TransactionIdentifier>
  <ServiceTypeEntity>
    <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:05-process-
language-NS:2012</mpegmb:ServiceTypeIdentifier>
  </ServiceTypeEntity>
  <ContentEntity>
    <didl:DIDL>
      <didl:Item>
        <didl:Component>
          <didl:Resource mimeType="text/plain"><![CDATA[
          I would like to see the movie Black Rain by the director Ridley Scott.
]]></didl:Resource>
        </didl:Component>
      </didl:Item>
    </didl:DIDL>
  </ContentEntity>
  <RequestParameters xsi:type="pl:ProcessLanguageType">
    <pl:Language>en-US</pl:Language>
    <pl:TaggingType href="urn:mpeg:mpegM:cs:03-es-NS:2012:TaggingTypeCS:1">
      <mpeg7:Name>Named Entity Recognition</mpeg7:Name>
    </pl:TaggingType>
    <pl:TaggingSet xsi:type="mpeg7:TermUseType">
      <mpeg7:Name>ETRI NE Tagger</mpeg7:Name>
    </pl:TaggingSet>
  </RequestParameters>
</mpegm:ProcessContentRequest>
```

After successfully creating a tagging output from the input content, the Service Provider sends a `ProcessContentCompletion` message to the client.

```
<ProcessContentCompletion>
  <mpegmb:TransactionIdentifier>77</mpegmb:TransactionIdentifier>
  <ProcessContentCompletionSuccess>
    <ProcessedContentEntity>
      <didl:DIDL>
        <didl:Item>
          <didl:Component>
```

```
            <didl:Resource mimeType="text/plain"><![CDATA[
              Black Rain/movie_name + Ridley Scott/director_name
]]></didl:Resource>
          </didl:Component>
        </didl:Item>
      </didl:DIDL>
    </ProcessedContentEntity>
  </ProcessContentCompletionSuccess>
</ProcessContentCompletion>
```

B.5 Translate Language

B.5.1 Introduction

This subclause specifies the "Translate Language" Service Type for the Process Content service.

The Service Type is employed for translating the input text provided in a source language into the output text in a target language.

B.5.2 Syntax of Service Type data format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xmlns:tl="urn:mpeg:mpegM:service-type:05-translate-language-NS:2012"
targetNamespace="urn:mpeg:mpegM:service-type:05-translate-language-NS:2012"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="ISO/IEC 23006-4 2nd edition" id="translate-language.xsd">
  <import namespace="urn:mpeg:mpegM:schema:01-base-NS:2012"
schemaLocation="../mpeg-m-base.xsd"/>
  <import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="../mpeg/mpeg7-
v3.xsd"/>
  <!-- Definition of Translate Language Service Type -->
  <complexType name="TranslateLanguageType">
    <complexContent>
      <extension base="mpegmb:RequestParametersBaseType">
        <sequence>
          <element name="SourceLanguage" type="language"/>
          <element name="TargetLanguage" type="language"/>
          <element name="DomainInfo" type="mpeg7:SemanticBaseType"
minOccurs="0"/>
        </sequence>
        <attribute name="requestVariants" type="boolean" use="optional"
default="false"/>
        <attribute name="linguisticStyle" use="optional" default="formal">
          <simpleType>
            <union>
              <simpleType>
                <restriction base="NMTOKEN">
                  <enumeration value="formal"/>
                  <enumeration value="informal"/>
                </restriction>
              </simpleType>
              <simpleType>
```

```
                <restriction base="mpeg7:termReferenceType"/>
              </simpleType>
            </union>
          </simpleType>
        </attribute>
        <attribute name="writerGender" use="optional" default="unspecified">
          <simpleType>
            <restriction base="NMTOKEN">
              <enumeration value="female"/>
              <enumeration value="male"/>
              <enumeration value="neuter"/>
              <enumeration value="unspecified"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name="audienceGender" use="optional" default="unspecified">
          <simpleType>
            <restriction base="NMTOKEN">
              <enumeration value="female"/>
              <enumeration value="male"/>
              <enumeration value="neuter"/>
              <enumeration value="unspecified"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name="audienceAgeSpan" use="optional" default="adult">
          <simpleType>
            <union>
              <simpleType>
                <restriction base="NMTOKEN">
                  <enumeration value="child"/>
                  <enumeration value="adult"/>
                </restriction>
              </simpleType>
              <simpleType>
                <restriction base="mpeg7:termReferenceType"/>
              </simpleType>
            </union>
          </simpleType>
        </attribute>
      </extension>
    </complexContent>
  </complexType>
</schema>
```


B.5.3 Semantics of Service Type data format

Semantics of the `TranslateLanguageType`:

Name	*Definition*
`TranslateLanguageType`	Parameters for the `ProcessContentRequest` message of Service Type "Translate Language". It contains necessary and optional parameters for the Translate Language Service Type. `TranslateLanguageType` extends `mpegmb:RequestParametersBaseType`.

Name	*Definition*
`SourceLanguage`	Indicates the language of the input text.
`TargetLanguage`	Indicates the language of the output text which is the result text of the translation from the source language.
`DomainInfo`	Domain of the content to be translated (e.g., science, weather, drama).
`requestVariants`	Indicates whether multiple possible translations of the input shall be provided. If set to *true*, the client can choose the preferred translation himself/herself.
`linguisticStyle`	Indicates the linguistic register or style of the output text such as formal style or informal style which is important for highly advanced natural translation in some languages where formality is expressed morphologically. Other values that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved. EXAMPLE The 1st person singular "you" can be translated to German as either "du" in informal style or as "Sie" in formal style.
`writerGender`	Gender of the writer for target languages that have linguistic differentiation of the author's gender. EXAMPLE The English text "I am happy" can be translated to Spanish either as "Estoy contenta" for a female author or as "Estoy contento" for a male author.
`audienceGender`	Gender of the audience for target languages that have linguistic differentiation of the audience's gender. EXAMPLE The English text "Are you happy?" can be translated to Spanish either as "¿Estás contenta?" for a female audience or as "¿Estás contento?" for a male audience.
`audienceAgeSpan`	Specifies whether the intended reader is a child for target languages that have linguistic differentiation of the audiences age span. Other types that are datatype-valid with respect to `mpeg7:termReferenceType` are reserved.

NOTE The output of translate language is provided in the `ProcessedContentEntity` element of the `ProcessContentCompletion` message.

B.5.4 Extension to SID

For the representation of Service Instance Declarations of the *Translate Language* Service Type, the following complex type is defined in the *SID XML Schema*:

```
  <!-- NOTE: This extension is defined in the SID XML Schema. -->

  <complexType name="STTranslateLanguageSIDType">
    <complexContent>
```

```
      <extension base="sid:ProcessContentSIDType">
        <sequence>
          <element name="SupportedSourceLanguage" type="language" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="SupportedTargetLanguage" type="language" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

Semantics of the `sid:STTranslateLanguageSIDType`:

Name	*Definition*
`sid:STTranslateLanguageSIDType`	Top-level type for SIDs of the *Translate Language* Service Type of the *Process Content* specific ES. `sid:STTranslateLanguageSIDType` extends `sid:ProcessContentSIDType`.
`sid:SupportedSourceLanguage`	Optional list source languages that the SP supports.
`sid:SupportedTargetLanguage`	Optional list of target languages that the SP supports.

B.5.5 Service Type XML declaration

The XML declaration of the "Translate Language" Service Type is defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpegmb:ServiceType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mpegmb="urn:mpeg:mpegM:schema:01-base-NS:2012"
xmlns:tl="urn:mpeg:mpegM:service-type:05-translate-language-NS:2012">

  <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:05-translate-
language-NS:2012</mpegmb:ServiceTypeIdentifier>
  <mpegmb:Name>Translate Language</mpegmb:Name>

<mpegmb:RequestParametersType>tl:TranslateLanguageType</mpegmb:RequestParametersT
ype>
  <!-- NOTE: This Service Type defines no extension for the
mpegmb:CompletionParametersType. -->
</mpegmb:ServiceType>
```

B.5.6 Example

For the actual processing, the client sends a `ProcessContentRequest` message to the SP, containing a text input to be processed and parameters for the translation specified by the client.

```
<ProcessContentRequest immediateResponse="true">
  <mpegmb:TransactionIdentifier>35</mpegmb:TransactionIdentifier>
  <ServiceTypeEntity>
```

```
    <mpegmb:ServiceTypeIdentifier>urn:mpeg:mpegM:service-type:05-translate-
language-NS:2012</mpegmb:ServiceTypeIdentifier>
  </ServiceTypeEntity>
  <ContentEntity>
    <didl:DIDL>
      <didl:Item>
        <didl:Component>
          <didl:Resource mimeType="text/plain">
            I am happy. How are you?
          </didl:Resource>
        </didl:Component>
      </didl:Item>
    </didl:DIDL>
  </ContentEntity>
  <RequestParameters xsi:type="tl:TranslateLanguageType" linguisticStyle="formal"
writerGender="female">
    <tl:SourceLanguage>en</tl:SourceLanguage>
    <tl:TargetLanguage>es</tl:TargetLanguage>
  </RequestParameters>
</ProcessContentRequest>
```

The SP concludes that the processing can be performed and sends an affirmative `ProcessContentResponse` message to the client.

```
<ProcessContentResponse>
  <mpegmb:TransactionIdentifier>35</mpegmb:TransactionIdentifier>
  <ProcessContentSuccess/>
</ProcessContentResponse>
```

After successfully translating the input text in a source language to the output text in a target language, the Service Provider sends a `ProcessContentCompletion` message to the client.

```
<ProcessContentCompletion>
  <mpegmb:TransactionIdentifier>35</mpegmb:TransactionIdentifier>
  <ProcessContentCompletionSuccess>
    <ProcessedContentEntity>
      <didl:DIDL>
        <didl:Item>
          <didl:Component>
            <didl:Resource mimeType="text/plain">
              Estoy contenta. ¿Como está usted?
            </didl:Resource>
          </didl:Component>
        </didl:Item>
      </didl:DIDL>
    </ProcessedContentEntity>
  </ProcessContentCompletionSuccess>
</ProcessContentCompletion>
```


B.6 Extract Sensory Information

B.6.1 Introduction

This subclause specifies the "Extract Sensory Information" Service Type for the Process Content service.