
**Information technology — Multimedia
framework (MPEG-21) —**

**Part 23:
Smart Contracts for Media**

*Technologies de l'information — Cadre multimédia (MPEG-21) —
Partie 23: Contrats intelligents pour les médias*

IECNORM.COM : Click to view the full PDF of ISO/IEC 21000-23:2022



IECNORM.COM : Click to view the full PDF of ISO/IEC 21000-23:2022



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2022

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

Page

Foreword.....	v
Introduction.....	vi
1 Scope.....	1
2 Normative references.....	1
3 Terms, definitions, symbols, and abbreviated terms.....	1
3.1 Terms and definitions.....	1
3.2 Abbreviated terms.....	3
4 Conventions.....	3
4.1 Classes representation.....	3
4.2 Namespace prefixes.....	4
5 Overview.....	5
5.1 General aspects.....	5
5.2 Relationships between MPEG-21 CEL/MCO and DLTs.....	5
6 Bidirectional conversion between MPEG-21 CEL/MCO contracts and smart contracts for media.....	8
6.1 Conversion from MPEG-21 CEL/MCO contracts to smart contracts for media.....	9
6.1.1 MPEG-21 CEL/MCO parser.....	9
6.1.2 Smart contract generator.....	10
6.1.3 DLT tokens and payments manager.....	10
6.2 Conversion from smart contracts for media to MPEG-21 CEL/MCO contracts.....	11
6.2.1 Smart contract parser.....	11
6.2.2 MPEG-21 CEL/MCO generator.....	11
7 Narrative contracts.....	12
8 API for media contractual objects.....	12
8.1 Contract.....	13
8.1.1 Contract.....	13
8.1.2 Encryptable.....	17
8.2 Party.....	18
8.2.1 Party.....	18
8.2.2 Person/User.....	19
8.2.3 CELPerson.....	20
8.2.4 MCOUser.....	20
8.2.5 Organization.....	21
8.3 Deontic.....	22
8.3.1 DeonticStructuredClause/DeonticExpression.....	22
8.3.2 TextClause.....	23
8.3.3 CELDeonticStructuredBlock.....	24
8.3.4 CELDeonticStructuredClause.....	25
8.3.5 CELCondition.....	26
8.3.6 MCODEonticExpression.....	27
8.3.7 Permission.....	28
8.3.8 CELPermission.....	29
8.3.9 MCOPermission.....	29
8.4 Action.....	30
8.4.1 Act/GenericAction/Action.....	30
8.4.2 Trade.....	33
8.4.3 Provide.....	33
8.4.4 Payment.....	34
8.4.5 Notify.....	35
8.4.6 UserDefinedAction.....	36
8.5 Object.....	37

8.5.1	Object	37
8.5.2	Item	37
8.5.3	IPEntity	38
8.5.4	Event	39
8.5.5	Segment	40
8.5.6	Service	40
8.5.7	SubjectWrapperObject	41
8.5.8	Track	42
8.5.9	Interval	43
8.6	Fact	43
8.6.1	Constraint/Fact	43
8.6.2	FactComposition	46
8.6.3	ActionEventFact	47
8.6.4	TogetherWith	48
8.6.5	AccessPolicy	48
8.6.6	DeliveryModality	49
8.6.7	Device	50
8.6.8	IPEntityContext	51
8.6.9	Language	51
8.6.10	Length	52
8.6.11	MaterialFormat	52
8.6.12	Means	54
8.6.13	Runs	54
8.6.14	ServiceAccessPolicy	55
8.6.15	ServiceChannelContext	56
8.6.16	SpatialContext	57
8.6.17	TemporalContext	57
8.6.18	UserTimeAccess	58
8.6.19	UserDefinedFact	59
9	API for MPEG-21 CEL/MCO parser	60
10	API for MPEG-21 CEL/MCO generator	60
11	Reference software and conformance	60
11.1	MPEG-21 template contracts	62
11.1.1	Open Music Initiative use cases	62
11.2	MPEG-21 Contract Expression Language	63
11.2.1	MPEG-21 CEL parser	63
11.2.2	MPEG-21 CEL generator	63
11.2.3	MPEG-21 CEL contracts to smart contracts for media (forward conversion)	63
11.2.4	Smart contracts for media to MPEG-21 CEL contracts (backward conversion)	65
11.3	MPEG-21 Media Contract Ontology	66
11.3.1	MPEG-21 MCO parser	67
11.3.2	MPEG-21 MCO generator	68
11.3.3	MPEG-21 MCO contracts to smart contracts for media (forward conversion)	68
11.3.4	Smart contracts for media to MPEG-21 MCO contracts (backward conversion)	73
11.4	OpenAPI and demo	75
11.4.1	OpenAPI	75
11.4.2	MPEG-21 MCO OpenAPI server	76
11.4.3	MPEG-21 CEL server	76
11.4.4	Demo	76
	Bibliography	78

Foreword

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

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

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

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

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

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

A list of all parts in the ISO/IEC 21000 series can be found on the ISO and IEC websites.

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

Introduction

The Moving Picture Experts Group (MPEG) standards include a set of RDF ontologies for the codification of intellectual property (IP) rights information related to media. The ISO/IEC 21000-19 Media Value Chain Ontology (MVCO) which facilitates rights tracking for fair, timely, and transparent payment of royalties by capturing user roles and their permissible actions on a particular IP entity. The ISO/IEC 21000-19/AMD1 Audio Value Chain Ontology (AVCO) which extends MVCO functionality related to the description of IP entities in the audio domain (e.g. multitrack audio and time segments). The ISO/IEC 21000-21 Media Contract Ontology (MCO) which facilitates the conversion of narrative contracts to digital ones related to exploitation of IP rights, payments and notifications. With respect to the latter, an equivalent standard has also been developed but using XML schemas, known as ISO/IEC 21000-20 Contract Expression Language (CEL).

Furthermore, the axioms in these XML schemas and RDF ontologies can drive the execution of rights-related workflows in controlled environments, for example, Distributed Ledger Technologies (DLTs), where transparency and interoperability are favored toward fair trade of music and media. Thus, the aim of this document is to provide the means (e.g. protocols and application programming interfaces) for converting these XML and RDF media contracts to smart contracts executable on existing DLT environments.

By doing this conversion in a standard way for several smart contract languages it is going to ensure that MPEG schemas and ontologies prevail as the interlingua for transferring verified contractual data from one DLT to another.

IECNORM.COM : Click to view the full PDF of ISO/IEC 21000-23:2022

Information technology — Multimedia framework (MPEG-21) —

Part 23: Smart Contracts for Media

1 Scope

This document specifies the means (e.g. protocols and application programming interfaces) for converting MPEG-21 XML and RDF media contracts (ISO/IEC 21000-19, ISO/IEC 21000-20, and ISO/IEC 21000-21) to smart contracts executable on existing DLT environments.

2 Normative references

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

ISO/IEC 21000-19, *Information technology — Multimedia framework (MPEG-21) — Part 19: Media Value Chain Ontology*

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

ISO/IEC 21000-21, *Information technology — Multimedia framework (MPEG-21) — Part 21: Media contract ontology*

3 Terms, definitions, symbols, and abbreviated terms

3.1 Terms and definitions

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

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1.1

DLT

distributed ledger technology

distributed network of computers, ideally organized in a decentralized way, mutually agreeing on a common state while tolerating failures (including malicious behavior) to some extent

3.1.2

smart contract

code deployed in a DLT or the source code from which such code was compiled

Note 1 to entry: The execution of smart contract instructions is distributed among the nodes of the DLT in which it is deployed to. This execution is triggered via a DLT transaction and produces a change in the DLT state.

3.1.3

smart contract language

programming language used for creating the code of a smart contract, that is then compiled in another code deployable to a specific DLT

3.1.4

smart contract template

source code of a smart contract written using a specific smart contract language for defining a common behavior.

3.1.5

smart contract specification

set of information needed for the deployment of a smart contract and for populating the data structures that the smart contract instructions are interacting with

3.1.6

DLT address

product of a cryptographic schema operation used to represent identities in a DLT

3.1.7

DLT governance

specification indicating the set of rules followed by the specific DLT protocol

3.1.8

token

object stored in a DLT and managed through one or more smart contracts, representing unique tangible or intangible media assets, possessions, and accountable items

3.1.9

fungible token

token being changeable with other tokens

3.1.10

non-fungible token

token being non interchangeable with other tokens

3.1.11

MPEG-21 CEL/MCO contract

contract represented using ISO/IEC 21000-19, ISO/IEC 21000-20 and ISO/IEC 21000-21 elements

3.1.12

media contractual objects

set of machine-readable objects extracted from a specific MPEG-21 CEL/MCO contract

3.1.13

smart contract for media

deployed smart contract that is the result of the conversion process from a specific MPEG-21 CEL/MCO contract

3.1.14

parser

software component that extracts a set of media contractual objects from an MPEG-21 CEL/MCO contract or a smart contract for media

3.1.15

generator

software component that from a set of media contractual objects generates an MPEG-21 CEL/MCO contract or a smart contract specification

3.1.16**DLT tokens and payments manager**

component deploying a smart contract for media on a specific DLT

3.1.17**contract developer**

actor providing the means to generate an MPEG-21 CEL/MCO contract or a smart contract in a specific smart contract language (e.g. smart contract templates)

3.1.18**DLT system engineer**

actor providing the information needed to deploy a smart contract in a specific DLT (e.g. DLT addresses and governance)

3.2 Abbreviated terms

MVCO	media value chain ontology
AVCO	audio value chain ontology
CEL	contract expression language
MCO	media contract ontology
CEL/MCO	ISO/IEC 21000-19, ISO/IEC 21000-20, and ISO/IEC 21000-21
IP	intellectual property
API	application programming interface

4 Conventions**4.1 Classes representation**

The following conventions derive from the Object-Oriented Programming paradigm. In this sense Application Programming Interfaces (APIs) are represented in terms of Classes definitions and Objects.

An Object is an instantiation of a Class while a Class contains the following properties:

- Name of the represented object.
- Type of the represented object. An object Type may be:
 - Abstract which is only showing essential information with respect to an interface, but it cannot be implemented; or,
 - Concrete which is a complete specification that can be implemented.
- Hierarchy with respect to the other objects; it also introduces the sub-class which is a class that inherits the complete set of fields and methods of its super-class.
- Fields which describe the attributes associated to the represented object; Fields consist of a specific Field Type and the number of Occurrences.
- Methods which are operations performed by manipulating the object Fields; Methods accept as input a specific set of Parameter Types and provide as output a specific set of Return Types.

In the following, [Table 1](#) shows the notation for representing Classes with respect to MPEG-21 CEL/MCO objects, while [Table 2](#) describes the Types used for Fields, Parameters and Returns.

Table 1 — Classes notation with respect to MPEG-21 CEL/MCO

Class		CEL	MCO
Name	Type and Hierarchy		
ClassName1	Abstract or Concrete, sub-class of ClassName2	<i>referenceToCELObject1</i>	<i>referenceToMCOObject1</i>
Fields			
Type	Field and Description	Occ.	
FieldName1Type	FieldName1	0, 1 or 1, n	<i>referenceToCELObject2</i> <i>referenceToMCOObject2</i>
Methods			
Parameters	Method and Description		Return
ParameterType1	method1()		ReturnType1

Table 2 — Types used for Fields, Parameters and Returns

Type	Description
string	A sequence of characters
ushort	An unsigned integer number represented through 2 bytes
ulong	An unsigned integer number represented through 4 or 8 bytes
float	A floating-point number, that is a number that can contain a fractional part, represented through 4 or 8 bytes
enum	A set of enumerated named elements
boolean	A dyadic value with two possible values, True and False
typeName []	An array of elements of type typeName
map(typeName1, typeName2)	A key value mapping where the key of type typeName is used to retrieve a value of type typeName2
void	A type used to represent "no data"
idref	A type used to represent a reference to a specific object, e.g. class instance. The form of classNameldref is used to reference objects that instantiate the class className, e.g. contractIdref refers to objects that instantiate the contract class.

4.2 Namespace prefixes

Table 3 below shows the namespace prefixes for the MPEG-21 CEL/MCO standards and other related schemas together with their references.

Table 3 — Mapping of prefixes to namespaces for the MPEG-21 CEL/MCO standards and other related schemas

Prefix	Corresponding namespace	References
dc	https://purl.org/dc/elements/1.1/	ISO 15836 [1]
dii	urn:mpeg:mpeg21:2002:01-DII-NS#	ISO/IEC 21000-3[2]
vcard	http://www.w3.org/2006/vcard/ns#	IETF RFC 2426[3]
mvco	https://purl.oclc.org/NET/mvco.owl#	ISO/IEC 21000-19
avco	https://purl.oclc.org/NET/aumvco.owl#	ISO/IEC 21000-19/Amd1
cel-core	urn:mpeg:mpeg21:cel:core:2015#	ISO/IEC 21000-20:2016
cel-ipre	urn:mpeg:mpeg21:cel:ipre:2015#	ISO/IEC 21000-20:2016

Table 3 (continued)

Prefix	Corresponding namespace	References
cel-pane	urn:mpeg:mpeg21:cel:pane:2015#	ISO/IEC 21000-20:2016
cel-rele	urn:mpeg:mpeg21:cel:rele:2015#	ISO/IEC 21000-20:2016
mco-core	urn:mpeg:mpeg21:mco:core:2015#	ISO/IEC 21000-21:2017
mco-ipre	urn:mpeg:mpeg21:mco:ipre:2015#	ISO/IEC 21000-21:2017
mco-pane	urn:mpeg:mpeg21:mco:pane:2015#	ISO/IEC 21000-21:2017
mco-rele	urn:mpeg:mpeg21:mco:rele:2015#	ISO/IEC 21000-21:2017

5 Overview

5.1 General aspects

MPEG-21 CEL/MCO schemas and ontologies can be used by music and media value chain stakeholders to share and exchange, in an interoperable way, all metadata and contractual information connected to creative works, leading to transparent payment of royalties and reduced time spent searching for the right data. The latter is due to inference and reasoning capabilities inherently associated with ontologies. That is, knowledge and data can be derived by evidence and logic based on rich semantic copyright models expressed by MPEG-21 CEL/MCO schemas and ontologies. In this way, the data derived are unambiguously interpretable, facilitating efficient processing in business-to-consumer (B2C) and business-to-business (B2B) music and media value chains.

Furthermore, for contractual music and media asset trading, smart contracts can be used to encode the terms and conditions of a contract. They validate contractual agreements between stakeholders before a DLT value transfer is enabled. In other words, smart contracts could allow music and media royalties to be administered almost instantaneously and manage usage allowances and restrictions. Rather than passing through intermediaries, revenue from a stream or download could be distributed automatically to rights holders, according to agreed terms and conditions (e.g. splits), as soon as an asset is downloaded or streamed.

Therefore, the challenge is converting MPEG-21 CEL/MCO standardized schemas and ontologies to smart contracts that can be executed on existing DLT environments, thus enriching DLT environments with inference and reasoning capabilities inherently associated with ontologies. Note that this process will increase trust among music and media value chain stakeholders for sharing data in the ecosystem since the data will be cryptographically secured and verified on a DLT. By addressing this challenge in a standard and agnostic way, with respect to smart contract languages and thus DLT environments, it would also ensure that MPEG-21 CEL/MCO schemas and ontologies prevail as the interlingua for transferring verified contractual data from one DLT to another^[4].

5.2 Relationships between MPEG-21 CEL/MCO and DLTs

This subclause describes the relationships between MPEG-21 CEL/MCO elements and DLTs components, for the conversion of MPEG-21 CEL/MCO contracts to smart contracts for media and vice versa. Smart contracts for media are distinguished from generic smart contracts since they are the result of the conversion process from a specific MPEG-21 CEL/MCO contract.

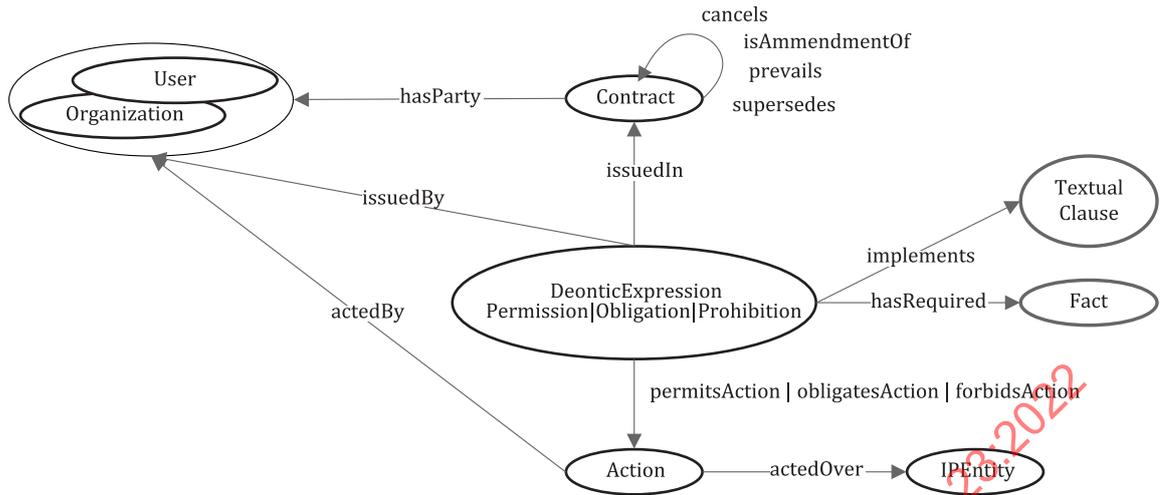
For the description of above-mentioned relationships, the main elements identified for MPEG-21 CEL/MCO are the contract, the party, the IP entity, and the deontic expression. The counterparts in a DLT-based scenario have been identified as shown in [Table 4](#).

Table 4 — Relationships between MPEG-21 CEL/MCO elements and DLTs components

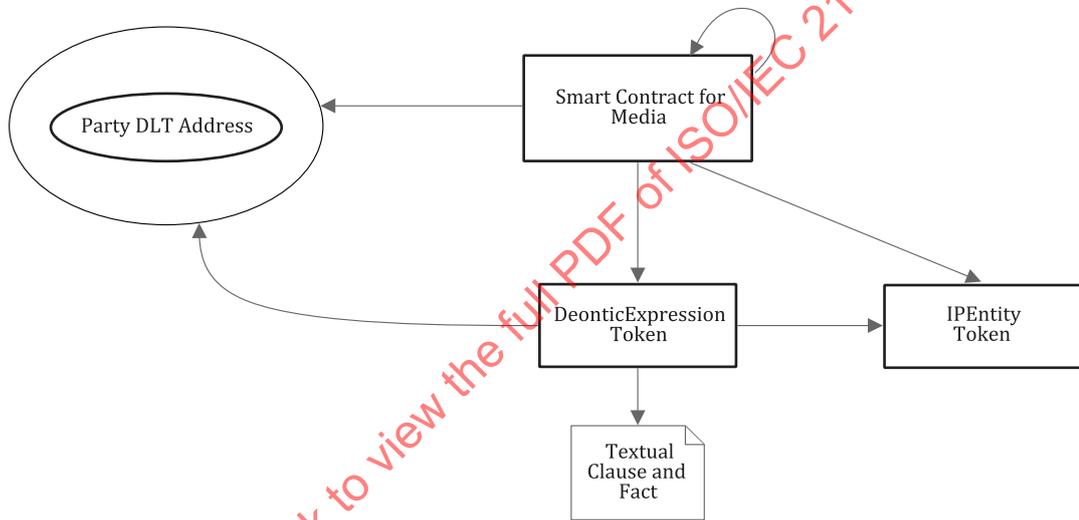
MPEG-21 CEL/MCO	DLTs
Contract	Smart contract for media
Party	DLT address
IP entity	Non-fungible token
Deontic expression	Non-fungible token

Furthermore, in [Figure 1](#), as for example, the relationships between MPEG-21 MCO and DLTs are depicted, albeit similar relationships apply between MPEG-21 CEL and DLTs. These relationships are further explained in the following.

IECNORM.COM : Click to view the full PDF of ISO/IEC 21000-23:2022



a) ISO/IEC 21000-21:2017 Media Contract Ontology



b) Relationship between ISO/IEC 21000-21:2017 Media Contract Ontology and DLTs

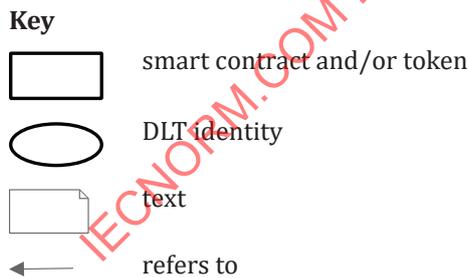


Figure 1 — Relationships between MPEG-21 MCO elements and DLTs components

- 1) **Contract - Smart contract for media:** the MPEG-21 CEL/MCO contract element is the one that includes or refers to the digitalized contractual information extracted from a narrative contract. Whilst the smart contract for media is the result of the conversion process from the MPEG-21 CEL/MCO contract. Thus, the counterpart of an instance of an MPEG-21 CEL/MCO contract is a unique smart contract for media deployed on a specific DLT.
- 2) **Party - DLT address:** a Party element is the representation of the identity of a user or organization bound by the narrative contract. Since identities in DLTs are represented through addresses, the

Party element counterpart is a DLT address. Thus, a Party identity represented by a DLT address may also be authenticated in the DLT and referenced in a smart contract for media.

- 3) **IP entity – Non-fungible token:** an IP Entity element is the representation of an asset, and the reference to this asset can be stored on a DLT. This representation of an asset may be serialized according to the concept of non-fungible tokens. Thus, in smart contracts an IP Entity may be represented by a token. Then, the entire set of information related to a specific IP Entity is linkable to the associated token. Two reasons support this approach:
 - i) the linkage between IP Entities and related smart contracts for media is maintained at a high level, particularly when DLTs offer append-only data storage and not a more complex one;
 - ii) it makes feasible the process of auditing, exploiting at best the immutability feature of DLTs; the history of all operations executed over an IP Entity, indeed, can be found in one place.
- 4) **Deontic expression – Non-fungible token:** a Deontic Expression encompasses the properties of an agreed machine-readable contract clause regulating the actions of the Parties (e.g. obligations, permissions, and prohibitions). This representation of a clause may also be serialized according to the concept of non-fungible tokens. The reasons for supporting this approach are:
 - i) it enables a unique way for storing clauses on DLTs, that is also beneficial in terms of interoperability, for sharing these clauses with other DLT-based applications;
 - ii) it allows the transfer of value in the form of obligations, permissions and prohibitions, similarly to how cryptocurrency transfers are done.

6 Bidirectional conversion between MPEG-21 CEL/MCO contracts and smart contracts for media

This clause describes the bidirectional conversion processes between MPEG-21 CEL/MCO contracts and smart contracts for media. In [Figure 2](#) it is shown the forward conversion from MPEG-21 CEL/MCO contracts to smart contracts for media, while, in [Figure 3](#) it is shown the backward conversion from smart contracts for media to MPEG-21 CEL/MCO contracts.

Both processes interact with several actors and DLTs where a smart contract for media would be (forward conversion) or has been (backward conversion) deployed. In the following subclauses, a set of interrelated components are described, each of which consists of a grouping of related functionality encapsulated behind a well-defined interface (e.g. inputs and outputs).

The smart contract for media may store instances of the MPEG-21 CEL/MCO contract elements either:

- in data structures of the smart contract for media; or
- in non-fungible tokens referenced by the smart contract for media, which are stored on the same DLT but managed through a different smart contract.

By storing these elements in that way, this document also facilitates the backward conversion from smart contracts for media to MPEG-21 CEL/MCO contracts in the XML^[5]/RDF^[6] form. In turn, MPEG-21 CEL/MCO contracts may be transformed into narrative contracts.

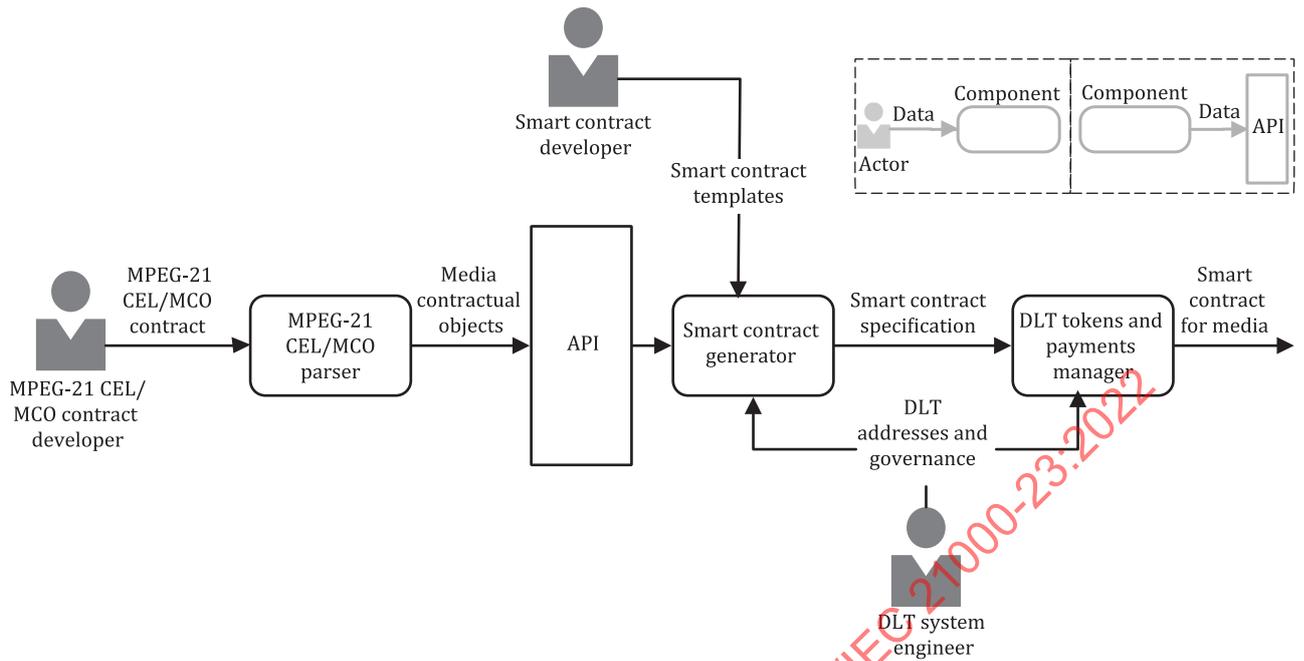


Figure 2 — Conversion workflow from MPEG-21 CEL/MCO contracts to smart contracts for media

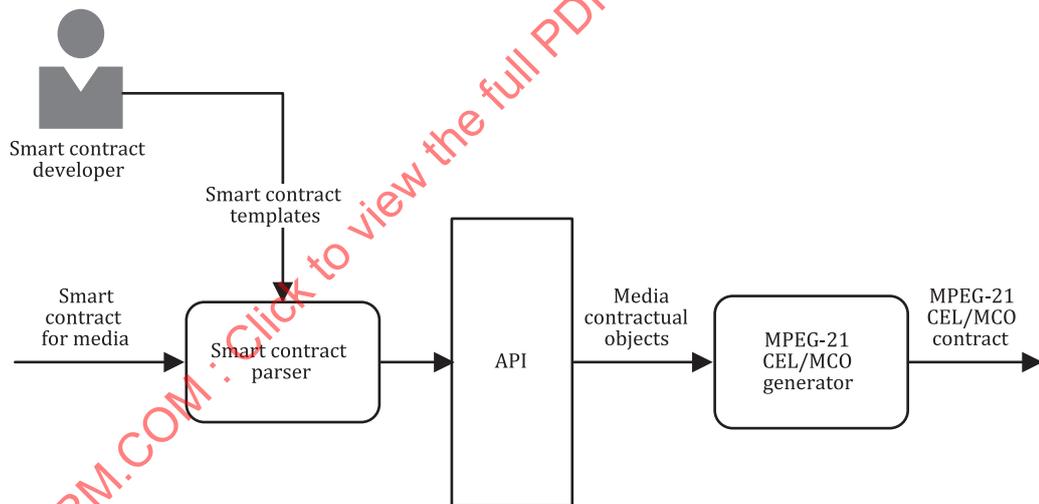


Figure 3 — Conversion workflow from smart contracts for media to MPEG-21 CEL/MCO contracts

6.1 Conversion from MPEG-21 CEL/MCO contracts to smart contracts for media

The process of conversion from an MPEG-21 CEL/MCO contract to a smart contract for media involves the execution of several components and the interaction with three actors and a DLT. This process is graphically illustrated in [Figure 2](#).

6.1.1 MPEG-21 CEL/MCO parser

The MPEG-21 CEL/MCO parser component gets as input an MPEG-21 CEL/MCO contract, provided by an MPEG-21 CEL/MCO contract developer, and produces a set of media contractual objects. It is expected that the MPEG-21 CEL/MCO contract has been checked to be syntactically and semantically

valid. Otherwise, the validation result provided by the MPEG-21 CEL/MCO parser returns two levels of information:

1. **Errors:** syntactic or semantic errors are identified.
2. **Warnings:** elements of information that may be lost during the conversion process.

Input:

- **MPEG-21 CEL/MCO contract:** it consists of XML/RDF documents containing one or several MPEG-21 CEL/MCO elements that represent a contract.

Output:

- **Media contractual objects:** a structured set of information related to the deontic expressions, actions, entities, and constraints extracted from an MPEG-21 CEL/MCO contract.

These media contractual objects are only dependent upon the MPEG-21 CEL/MCO standards and as such are agnostic with respect to any specific DLT.

6.1.2 Smart contract generator

The smart contract generator component produces a smart contract specification by combining information related to an MPEG-21 CEL/MCO contract in the form of media contractual objects with some specific smart contracts templates. This component is meant to be dependent on the smart contract language.

Inputs:

- **Media contractual objects:** a set of media contractual objects is needed to define the information that is contained in the smart contract for media.
- **Smart contract templates:** the DLT smart contract developer (e.g. the person skilled in the specific DLT smart contract language) elaborates a set of specific smart contract templates to be utilized in the conversion process.

Output:

- **Smart contract specification:** a set of elements that represent the information needed by the DLT tokens and payments manager component to deploy the smart contract for media.

This smart contract specification includes information produced based on the objects found while traversing the set of media contractual objects. If MPEG-21 CEL/MCO obligations including payments are found, then the smart contract generator produces the information needed for creating revenue functions for each party involved. For instance, if a party is obliged to share its revenue with another party, then a smart contract method performs the revenue sharing function (e.g. royalties flow). Moreover, for each MPEG-21 CEL/MCO deontic expression and IP entity found in the media contractual objects, the smart contract generator produces the information needed for creating a new non-fungible token.

6.1.3 DLT tokens and payments manager

The purpose of the DLT tokens and payments manager component is to deploy the smart contract for media that derives from a smart contract specification, a set of DLT addresses and a DLT governance protocol. Both the DLT addresses and the DLT governance protocol are required for the operation in a specific DLT environment and are provided by a DLT system engineer.

Inputs:

- **Smart contract specification:** this specification enables the creation of a smart contract for media instance, which includes specific tokens and payments information.

- **DLT addresses:** this input refers to the bindings between MPEG-21 CEL/MCO contract parties and DLT addresses.
- **DLT governance:** the protocol the specific DLT adheres to that allows the DLT tokens and payments manager component to update the ledger.

Output:

- **Smart contract for media:** the result of the execution of the DLT tokens and payments manager component is the deployed smart contract for media.

The DLT tokens and payments manager component performs some checks on whether each token referenced in the smart contract specification has already been in existence on the DLT. If this is not the case, it creates a new token and stores its reference in the smart contract for media.

The DLT tokens and payments manager component may store an MPEG-21 CEL/MCO element in a token, or store in a token an immutable reference of it and then the element itself outside of the DLT. The second mechanism is preferred when the element information needs to be kept private or when the DLT disincentives data storage.

6.2 Conversion from smart contracts for media to MPEG-21 CEL/MCO contracts

The process of conversion from a smart contract for media to an MPEG-21 CEL/MCO contract involves the execution of two components and the interaction with a DLT and a DLT smart contract developer. This process is graphically illustrated in [Figure 3](#).

6.2.1 Smart contract parser

The smart contract parser is a component that fetches the data of a deployed smart contract for media, which uses to produce a set of media contractual objects.

Inputs:

- **Smart contract for media:** the parser fetches the data from a deployed smart contract for media.
- **Smart contract template:** a template is required for decoding the data structures within the smart contract for media.

Output:

- **Media contractual objects:** a set of media contractual objects extracted from the information contained in the smart contract for media.

6.2.2 MPEG-21 CEL/MCO generator

The MPEG-21 CEL/MCO generator is a component performing the backward operation with respect to the MPEG-21 CEL/MCO parser described previously. That is, the MPEG-21 CEL/MCO generator gets as input a set of media contractual objects and produces an MPEG-21 CEL/MCO contract.

Input:

- **Media contractual objects:** a set of media contractual objects extracted from the information contained in the smart contract for media.

Output:

- **MPEG-21 CEL/MCO contract:** an XML/RDF document containing one or several MPEG-21 CEL/MCO elements derived from the media contractual objects.

7 Narrative contracts

One shortcoming is that, generally, there is no way to deduce from a smart contract the clauses that the smart contract contains. Publishing the narrative contract does not ensure that the clauses of the narrative contract correspond to the clauses of the smart contract. There should be a way that allows the other party of the smart contract to know beyond doubt what the clauses stored in the smart contract express.

Thus, an important feature of this document is that it offers the possibility to bind, through persistent links, the clauses of a smart contract to the corresponding ones of the narrative contract and vice versa, e.g. the narrative clause x “user A pays \$1 to user B” is bound to its counterpart smart contract clause x “Transfer UserA UserB \$1”. In the latter, if the beneficiary of the payment is not clear, the link to its corresponding narrative clause could be handy. In the following, the MPEG-21 CEL/MCO elements enabling such bindings are further described.

For example, the narrative version of contracts is preserved in MPEG-21 CEL contracts, either in plain text (`cel-core:TextVersion`) or encrypted (`cel-core:EncryptedTextVersion`). Further structuring is possible by means of `cel-core:TextClause` and `cel-core:TextParagraph` (in clear text or encrypted). These text elements can be referenced with the attribute 'idrefs' from the operative part of the contracts, maintaining the isomorphism (e.g. the one-to-one relationship between an operative clause to its narrative counterpart) explicitly.

Equivalently, MPEG-21 MCO contracts also preserve the narrative version of the contract, either in a complete form (using the `mco-core:TextVersion` property) or through a number of textual clauses which can also be referenced from the operative part. These clauses are instances of the `mco-core:TextualClause`, with `mco-core:Text` as a data property.

In that way, the narrative version of a contract and its clauses is represented in MPEG-21 CEL/MCO contracts, and these relationships are also maintained all the way through the media contractual objects (`textVersion` and `textClauses`) to smart contracts and vice versa. In turn, this ensures the parties signing a smart contract to know beyond doubt what the clauses stored in the smart contract express with respect to the clauses of the narrative contract.

By doing this conversion in a standard way for several smart contract languages would ensure MPEG-21 CEL/MCO contracts prevail as the interlingua for transferring verified contractual data from one DLT to another.

8 API for media contractual objects

This API, by facilitating the creation and handling of media contractual objects, is fundamental for the bidirectional conversion from MPEG-21 CEL/MCO contracts to smart contracts for media. Thus, this clause specifies the media contractual objects API. This API has been derived and shall be used in conjunction with MPEG-21 XML and RDF media contracts (ISO/IEC 21000-19, ISO/IEC 21000-20, and ISO/IEC 21000-21). The specific XML schemas and RDF ontologies contained in these standards are shown in [Table 5](#). Furthermore, the presentation of this API follows the conventions (e.g. classes representation and namespace prefixes) as described in [Clause 4](#).

Table 5 — Schemas and ontologies contained in MPEG-21 CEL/MCO

MPEG-21 CEL/MCO	Schemas and Ontologies
ISO/IEC 21000-19 Media Value Chain Ontology	MVCO
ISO/IEC 21000-19/AMD1 Audio Value Chain Ontology	AVCO

Table 5 (continued)

MPEG-21 CEL/MCO	Schemas and Ontologies
ISO/IEC 21000-21:2017 Media Contract Ontology	MCO-CORE, MCO Intellectual Property Rights Extension (MCO-IPRE), MCO Payments and Notifications Extension (MCO-PANE), MCO Right Expression Language Extension (MCO-RELE)
ISO/IEC 21000-20:2016 Contract Expression Language	CEL-CORE, CEL Intellectual Property Rights Extension (CEL-IPRE), CEL Payments and Notifications Extension (CEL-PANE), CEL Right Expression Language Extension (CEL-RELE)

8.1 Contract

8.1.1 Contract

Class		CEL	MCO
Name	Type and Hierarchy		
Contract	Concrete, sub-class of Encryptable	<i>cel-core:Contract</i>	<i>mco-core:Contract</i>
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the contract	1	<i>cel-core:Contract</i> <i>contractId (attribute)</i> Internationalized Resource Identifier (IETF Standard is RFC 3987)
string	governingLaw Indicating the jurisdiction and applicable law	0, 1	<i>cel-core:Contract</i> <i>governingLaw (attribute)</i> <i>mco-core:hasGoverningLaw</i>
string	court Has exclusive jurisdiction over any dispute related to the contract's terms and conditions	0, 1	<i>cel-core:Contract</i> <i>court (attribute)</i> <i>mco-core:hasCourt</i>
boolean	isCourtJurisdictionExclusive	0, 1	<i>isCourtJurisdictionExclusive</i> <i>mco-core:isCourtJurisdictionExclusive</i>
string	textVersion Whole narrative contract text	0, 1	<i>cel-core:TextVersion</i> <i>mco-core:TextVersion</i>
string	encryptedTextVersion An encrypted version of the whole narrative contract text	0, 1	<i>cel-core:EncryptedTextVersion</i> <i>mco-core:encryptedContractPart</i>
map (string, string)	metadata For giving information about the contract itself.	0, n	<i>cel-core:Metadata</i> Dublin Core (ISO 15836) or others Dublin Core (ISO 15836) or others

Class		CEL	MCO
Name	Type and Hierarchy		
map(contractIdref, enum(contractRelations))	contractRelations Used to relate the contract to other identified contracts. The enum of the map value must be one of the following: 0 = supersedes 1 = cancels 2 = prevailsOver 3 = isAmendmentOf	0, n	<i>cel-core:ContractsRelated</i> <i>mco-core:contractObjectProperty</i>
map(partyIdref, Party)	parties Persons or organizations for which the contract is binding	0, n	<i>cel-core:Party</i> <i>mco-core:Party</i>
map(personuserIdref, PersonUser)	otherPersonUsers Referenced Persons/Users for which the contract is not binding	0, n	<i>cel-core:Person</i> <i>cel-core:User</i>
map(deonticIdref, Deontic)	deontics Machine-readable operative parts of the contract used to represent agreements	1, n	list of <i>cel-core:DeonticStructuredClause</i> list of <i>mco-core:DeonticExpression</i>
map(actionIdref, Action)	actions Specifies the rights that are permitted / obligated / prohibited to parties	1, n	list of <i>cel-core:Act</i> within a <i>cel-core:DeonticStructuredClause</i> list of <i>mco-core:GenericAction</i> within a <i>mco-core:DeonticExpression</i>
map(objectIdref, Object)	objects Resources against which the deontics will apply	0, n	list of <i>cel-core:Object</i> list of <i>mco-core:actedOver</i> <i>mco-core:IPEntity</i> <i>mco-core:Service</i>
map(factIdref, Fact)	facts Conditions, restrictions and constraints within each deontic	0, n	<i>cel-core:Constraint</i> <i>mvco:Fact</i>
map(textIdref, TextClause)	textClauses Text clauses to reference from a deontic expression to narrative contract excerpts of which it makes the operative part	1, n	list of <i>cel-core:TextClause</i> list of <i>mco-core:TextualClause</i>
map(encryptedIdref, string)	encryptedContractParts For addressing the encryption needs regarding a part, or the entirety, of a contract document. The key value of the map must be the identifier of the part.	0, n	<i>cel-core:EncryptedContract</i> <i>cel-core:EncryptedParty</i> <i>cel-core:EncryptedClause</i> <i>cel-core:EncryptedBlock</i> list of <i>mco-core:encryptedContractPart</i>

Class		CEL	MCO
Name	Type and Hierarchy		
map(deonticStructuredBlockIdref, CELDeonticStructuredBlock)	<p>operativePart</p> <p>The contract part which contains the deontic expressions of the operative part.</p> <p>It embodies the set of related deontic clauses grouped as a nested structure (Specific for CEL)</p>	0, n	<i>cel-core:OperativePart</i> <i>cel-core:DeonticStructuredBlock</i>
personuserIdref[]	<p>signatories</p> <p>A signatory of a contract when different from a party (Specific for MCO)</p>	0, n	<i>mco-core:isSignedBy</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getGoverningLaw()	string
void	<p>getCourt()</p> <p>Has exclusive jurisdiction over any dispute related to the contract's terms and conditions</p>	string
void	getIsCourtJurisdictionExclusive()	boolean
void	getTextVersion()	string
void	getEncryptedTextVersion()	string
void	getMetadata()	map (string, string)
string	getMetadataBy()	string
void	getContractRelations()	map (contractIdref, enum(contractRelations))
contractIdref	getContractRelationsBy()	enum(contractRelations)
void	getParties()	map(partyIdref, Parties)
partyIdref	getPartyBy()	Party
personuserIdref	<p>getPersonUserBy()</p> <p>This method searches and possibly returns a PersonUser from the parties' map field</p>	PersonUser
void	getOtherPersonUsers()	map(personuserIdref, PersonUser)
personuserIdref	getOtherPersonUsersBy()	PersonUser
void	getDeontics()	map(deonticIdref, Deontic)
deonticIdref	getDeonticBy()	Deontic
void	getActions()	map(actionIdref, Action)
actionIdref	getActionBy()	Action
void	getObjects()	map(objectIdref, Object)
objectIdref	getObjectBy()	Object

Methods		
Parameters	Method and Description	Return
itemIdref	getItemBy() This method searches and possibly returns an Item from the objects map field	Item
ipentityIdref	getIPEntityBy() This method searches and possibly returns an IPEntity from the objects map field	IPEntity
void	getFacts()	map(factIdref, Fact)
factIdref	getFactBy()	Fact
void	getTextClauses()	map(textIdref, TextClause)
textIdref	getTextClausesBy()	TextClause
void	getEncryptedContractParts()	map(encryptedIdref, string)
encryptedIdref	getEncryptedContractPartsBy()	string
void	getOperativePart()	map(deonticStructuredBlockIdref, CELDeonticStructuredBlock)
deonticStructuredBlockIdref	getDeonticStructuredBlock() This method returns information about a deontic structured block	CELDeonticStructuredBlock
void	getConstraints() This method consists in a call to the method getFacts()	map(factIdref, Fact)
void	getExpressions() This methods consists in a call to the method getDeontics()	deonticIdref[]
void	getSignatories()	personuserIdref[]
string	Contract() This method constructs a new object and gets as input only the identifier	void
string, map(deonticIdref, Deontic), map(actionIdref, Action), map(textIdref, TextClause)	Contract() This method constructs a new object and gets as input all the required fields	void
string	setIdentifier()	void
string	setGoverningLaw()	void
string	setCourt()	void
boolean	setIsCourtJurisdictionExclusive()	void
string	setTextVersion()	void
string	setEncryptedTextVersion()	void
map (string, string)	setMetadata()	void
string, string	addMetadata()	void
map(contractIdref, enum(contractRelations))	setContractRelations()	void
contractIdref, enum(contractRelations)	addContractRelations()	void
map(partyIdref, Parties)	setParties()	void
partyIdref, Parties	addParties()	void

Methods		
Parameters	Method and Description	Return
map(deonticIdref, Deontic)	setDeontics()	void
deonticIdref, Deontic	addDeontics()	void
map(actionIdref, Action)	setActions()	void
actionIdref, Action	addActions()	void
map(objectIdref, Object)	setObjects()	void
objectIdref, Object	addObjects()	void
map(factIdref, Fact)	setFacts()	void
factIdref, Fact	addFacts()	void
map(textIdref, TextClause)	setTextClauses()	void
textIdref, TextClause	addTextClauses()	void
map(encryptedIdref, string)	setEncryptedContractParts()	void
encryptedIdref, string	addEncryptedContractParts()	void
map(deonticStructuredBlockIdref, CELDeonticStructuredBlock)	setOperativePart()	void
deonticStructuredBlockIdref, CELDeonticStructuredBlock	addOperativePart()	void
deonticStructuredBlockIdref	setDeonticStructuredBlock()	void
map(factIdref, Fact)	setConstraints()	void
factIdref, Fact	addConstraints()	void
deonticIdref[]	setExpressions()	void
deonticIdref	addExpressions()	void
personuserIdref[]	setSignatories()	void
personuserIdref	addSignatories()	void

8.1.2 Encryptable

Class		CEL	MCO
Name	Type and Hierarchy		
Encryptable	Abstract	cel-core: EncryptedContract cel-core: EncryptedParty cel-core: EncryptedClause cel-core: EncryptedBlock	mco-core:encryptedContractPart
Fields			
Type	Field and Description	Occ.	
encryptedIdref	encryptedRepresentation A full version of the current contract part in an encrypted representation	0, 1	/

Methods		
Parameters	Method and Description	Return
void	getEncryptedRepresentation()	encryptedIdref

8.2 Party

8.2.1 Party

Class		CEL	MCO
Name	Type and Hierarchy		
Contract	Abstract, sub-class of Encryptable	<i>cel-core:Party</i>	<i>mco-core:Party</i>
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the party	1	<i>cel-core:Party</i> id (attribute) <i>rdf:about</i>
string	name The name of the party	1	<i>cel-core:Name</i> /
map (string, string)	details To provide further detailed contact information	0, n	<i>cel-core:Details</i> <i>xmlns:vCard</i> <i>mco-core:hasVCard</i>
map (string, string)	metadata For giving information about the contract itself.	0, n	<i>dc:description</i> <i>dc:identifier</i> Dublin Core (ISO 15836) or others <i>Dublin Core (ISO 15836) or others</i>
string	address Party address as free text	0, 1	<i>cel-core:Address</i> <i>mco-core:Address</i>
deonticIdref[]	deonticsIssued Machine-readable operative parts of the contract issued by the party	0, n	list of <i>cel-core:Issuer</i> relations <i>list of mco-core:issuedBy</i> relations
actionIdref[]	actionsIsSubject The acts that the party may / must / must not / did execute, specified in a Deontic	0, n	<i>cel-core:Act</i> and <i>cel-core:Subject</i> relation within a <i>cel-core:DeonticStructuredClause</i> <i>mco-core:actedBy</i> for a <i>mco-core:GenericAction</i> in a <i>mco-core:DeonticExpression</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getName()	string
void	getDetails()	map (string, string)
string	getDetailBy()	string
void	getMetadata()	map (string, string)
string	getMetadataBy()	string
void	getAddress()	string

Methods		
Parameters	Method and Description	Return
void	getDeonticsIssued()	deonticIdref[]
void	getActionsIsSubject()	actionIdref[]
string	Party() This method constructs a new object and gets as input only the identifier	void
string, string	Party() This method constructs a new object and gets as input all the required fields	void
string	setIdentifier()	void
string	setName()	void
map (string, string)	setDetails()	void
string, string	addDetails()	void
map (string, string)	setMetadata()	void
string, string	addMetadata()	void
string	setAddress()	void
deonticIdref[]	setDeonticsIssued()	void
deonticIdref	addDeonticsIssued()	void
actionIdref[]	setActionsIsSubject()	void
actionIdref	addActionsIsSubject()	void

8.2.2 Person/User

Class		CEL	MCO
Name	Type and Hierarchy		
PersonUser	Abstract, sub-class of Party	<i>cel-core:Person</i>	<i>mco-core:User</i>
Fields			
Type	Field and Description	Occ.	
string	signature Contract binding signature	0, 1	<i>dsig:Signature</i> <i>mco-core:Signature</i>

Methods		
Parameters	Method and Description	Return
void	getSignature()	string
string	PersonUser() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	PersonUser() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string	setSignature()	void

8.2.3 CELPerson

Class		CEL	MCO
Name	Type and Hierarchy		
CELPerson	Concrete, sub-class of PersonUser	<i>cel-core:Person</i> <i>cel-core:Signatory</i>	/
Fields			
Type	Field and Description	Occ.	
/	/	/	/

Methods		
Parameters	Method and Description	Return
string	CELPerson() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	CELPerson() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void

8.2.4 MCOUser

Class		CEL	MCO
Name	Type and Hierarchy		
User	Concrete, sub-class of Person	/	<i>mco-core:User</i>
Fields			
Type	Field and Description	Occ.	
enum (role)	role The user role. The enum value must be: 0 = User (simple) 1 = Creator 2 = Adaptor 3 = Instantiator 4 = Producer 5 = Distributor 6 = EndUser 7 = Collective	1	/
string	socialTag	0, 1	/
personuserIdref[]	actOnBehalfOf	0, n	/
personuserIdref[]	belongsToCollective	0, n	/
ipentityIdref[]	isRightsOwnerOf	0, n	/

Methods		
Parameters	Method and Description	Return
void	getRole()	enum (role)
void	getSocialTag()	string
void	getActOnBehalfOf()	personuserIdref[]
void	getBelongsToCollective()	personuserIdref[]
void	getIsRightsOwnerOf()	ipentityIdref[]
string	MCOUser() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string, enum (role)	MCOUser() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (role)	setRole()	void
string	setSocialTag()	void
personuserIdref[]	setActOnBehalfOf()	void
personuserIdref	addActOnBehalfOf()	void
personuserIdref[]	setBelongsToCollective()	void
personuserIdref	addBelongsToCollective()	void
ipentityIdref[]	setIsRightsOwnerOf()	void
ipentityIdref	addIsRightsOwnerOf()	void

8.2.5 Organization

Class		CEL	MCO
Name	Type and Hierarchy		
Organization	Concrete, sub-class of Party	<i>cel-core:</i> Organization	<i>mco-core:</i> Organization
Fields			
Type	Field and Description	Occ.	
personuserIdref	signatory A signatory person optionally given for an Organization	0, 1	<i>cel-core:</i> Signatory <i>mco-core:</i> hasSignatory

Methods		
Parameters	Method and Description	Return
void	getSignatory()	personuserIdref
string	Organization() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	Organization() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
personuserIdref	setSignatory()	void

8.3 Deontic

8.3.1 DeonticStructuredClause/DeonticExpression

Class		CEL	MCO
Name	Type and Hierarchy		
Deontic	Abstract, sub-class of Encryptable	<i>cel-core</i> :DeonticStructuredClause	<i>mco-core</i> :DeonticExpression
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the deontic	1	<i>cel-core</i> :DeonticStructuredClause id (<i>attribute</i>) <i>rdf:about</i>
enum (type)	type 0 = Statement/Simple 1 = Permission 2 = Obligation 3 = Prohibition	1	<i>cel-core</i> :Statement, <i>cel-core</i> :Permission, <i>cel-core</i> :Obligation, <i>cel-core</i> :Prohibition <i>mco-core</i> :DeonticExpression, <i>mco-core</i> :Permission, <i>mco-core</i> :Obligation, <i>mco-core</i> :Prohibition
textIdref[]	textClauses The text clauses that represents	0, n	<i>cel-core</i> :DeonticStructuredClause idrefs (<i>attribute</i>) <i>mco-core</i> :implements
map (string, string)	metadata For giving information about the contract itself.	0, n	<i>cel-core</i> : Metadata Dublin Core (ISO 15836) or others Dublin Core (ISO 15836) or others
actionIdref	act Specifies the right that is permitted / obligated / prohibited	1	<i>cel-core</i> :Act <i>mco-core</i> :obligatesAction <i>mco-core</i> :permitsAction <i>mco-core</i> :forbidsAction
partyIdref	actedBySubject Party to which the deontic clause applies	1	<i>cel-core</i> :Act and <i>cel-core</i> :Subject relation within a <i>cel-core</i> :DeonticStructuredClause <i>mco-core</i> :actedBy for a <i>mco-core</i> :GenericAction in a <i>mco-core</i> :DeonticExpression
objectIdref[]	actObjects Resources against which the deontic expression will apply	0, n	<i>cel-core</i> :Object <i>mco-core</i> :actedOver <i>mvco</i> :Action <i>mco-core</i> :IPEntity <i>mco-core</i> :Service
itemIdref[]	resultantObject Resource resultant of applying the act over the actObjects	0, n	<i>cel-core</i> :ResultantObject inverse of <i>mvco</i> :ResultFrom in the relation between <i>mvco</i> :Action and <i>mvco</i> :IPEntity
factIdref[]	constraints	0, n	<i>cel-core</i> :Constraint <i>mco-core</i> :hasRequired
partyIdref	issuer	0, 1	<i>cel-core</i> :Issuer <i>mco-core</i> :issuedBy

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getType()	enum (type)
void	getTextClauses()	textIdref[]
void	getMetadata()	map (string, string)
string	getMetadataBy()	string
void	getAct()	actionIdref
void	getActedBySubject()	partyIdref
void	getActObjects()	objectIdref[]
void	getResultantObject()	itemIdref[]
void	getConstraints()	factIdref[]
void	getIssuer()	partyIdref
string	Deontic() This method constructs a new object and gets as input only the identifier.	void
string, enum (type), actionIdref, partyIdref	Deontic() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
enum (type)	setType()	void
textIdref[]	setTextClauses()	void
textIdref	addTextClauses()	void
map (string, string)	setMetadata()	void
string, string	addMetadata()	void
actionIdref	setAct()	void
partyIdref	setActedBySubject()	void
objectIdref[]	setActObjects()	void
objectIdref	addActObjects()	void
itemIdref[]	setResultantObject()	void
itemIdref	addResultantObject()	void
factIdref[]	setConstraints()	void
factIdref	addConstraints()	void
partyIdref	setIssuer()	void

8.3.2 TextClause

Class		CEL	MCO
Name	Type and Hierarchy		
TextClause	Concrete	cel-core:TextClause	mco-core:TextualClause
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the party	1	cel-core:TextClause id (attribute) rdf:about

Class		CEL	MCO
Name	Type and Hierarchy		
textIdref[]	innerTextClauses TextClauses objects considered within this one	0, n	cel-core:TextClause /
map(textIdref, string)	paragraphs Containing the actual text	0, n	cel-core:TextParagraph /
encrypted-Idref[]	encryptedParagraphs Encrypted representation of a paragraph	0, 1	cel-core:EncryptedTextParagraph /
string	text Optional representation of the clause as a whole text instead of paragraph subdivision	0, 1	/ mco-core:Text

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getInnerTextClauses()	textIdref[]
void	getParagraphs()	map (textIdref, string)
string	getParagraphsBy()	textIdref
void	getEncryptedParagraphs()	encryptedIdref[]
void	getText()	string
string	setTextClause() This method constructs a new object and gets as input only the identifier.	void
string	setIdentifier()	void
textIdref[]	setInnerTextClauses()	void
textIdref	addInnerTextClauses()	void
map(textIdref, string)	setParagraphs()	void
textIdref, string	addParagraphs()	void
encryptedIdref[]	setEncryptedParagraphs()	void
encryptedIdref	addEncryptedParagraphs()	void
string	setText()	void

8.3.3 CELDeonticStructuredBlock

Class		CEL	MCO
Name	Type and Hierarchy		
CELDeontic-Structured-Block	Concrete, sub-class of Encryptable		cel-core:DeonticStructuredBlock /
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the party	1	cel-core:DeonticStructuredBlock id (attribute) /

Class		CEL	MCO
Name	Type and Hierarchy		
textIdref[]	textClauses The text clauses that represent this block	0, n	cel-core:DeonticStructuredBlockIdrefs (attribute) /
deonticStructuredBlockIdref[]	innerDeonticStructuredBlocks DeonticStructuredBlock objects considered within this one	0, n	cel-core:DeonticStructuredBlock /
deonticIdref[]	deontics Deontics within a block	0, n	cel-core:DeonticStructuredClause /

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getTextClauses()	textIdref[]
void	getInnerDeonticStructuredBlocks()	deonticStructuredBlockIdref[]
void	getDeontics()	deonticIdref[]
string	CELDeonticStructureBlock() This method constructs a new object and gets as input only the identifier.	void
string	setIdentifier()	void
textIdref[]	setTextClauses()	void
textIdref	addTextClauses()	void
deonticStructuredBlockIdref[]	setInnerDeonticStructuredBlocks()	void
deonticStructuredBlockIdref	addInnerDeonticStructuredBlocks()	void
deonticIdref[]	setDeontics()	void
deonticIdref	addDeontics()	void

8.3.4 CELDeonticStructuredClause

Class		CEL	MCO
Name	Type and Hierarchy		
CELDeonticStructuredClause	Concrete, sub-class of Deontic	cel-core:DeonticStructuredClause	/
Fields			
Type	Field and Description	Occ.	
string	number	0, 1	cel-core:DeonticStructuredClause number (attribute) /
string[]	context Contextual information of any type that can be added to a deontic structured clause	0, n	cel-core:Context /

Class		CEL	MCO
Name	Type and Hierarchy		
map(condition-Idref, CELCondition)	preCondition Pre-conditions of the deontic clause	0, n	<i>cel-core</i> :PreCondition /
CELCondition	postCondition Post-condition of the deontic clause	0, 1	<i>cel-core</i> : PostCondition /

Methods		
Parameters	Method and Description	Return
void	getNumber()	string
void	getContext()	string[]
void	getPreCondition()	map(conditionIdref,CELCondition)
conditionIdref	getPreConditionBy()	CELCondition
void	getPostCondition()	CELCondition
string	CELDeonticStructuredClause() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum (type), actionIdref, party-Idref	CELDeonticStructuredClause() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string	setNumber()	void
string[]	setContext()	void
string	addContext()	void
map(conditionIdref,CELCondition)	setPreCondition()	void
conditionIdref,CEL-Condition	addPreCondition()	void
CELCondition	setPostCondition()	void

8.3.5 CELCondition

Class		CEL	MCO
Name	Type and Hierarchy		
CELCondition	Concrete	<i>cel-core</i> : PreCondition <i>cel</i> <i>-core</i> :PostCondition	/
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the condition	1	<i>cel-core</i> : PreCondition <i>cel-core</i> : PostCondition <i>id (attribute)</i> /

Class		CEL	MCO
Name	Type and Hierarchy		
enum (action-Status)	actionStatus Indicates the status of the action in the deontic clause to which the condition refers. The enum value must be: 0 = ActionStarted 1 = ActionDone	0, 1	actionStatus (<i>attribute</i>) /
string	withDelay The elapsed time after which the deontic clause has to be considered valid	0, 1	withDelay (<i>attribute</i>) /
string	validity the time of validity of the deontic clause	0, 1	validity (<i>attribute</i>) /

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getActionStatus()	enum (actionStatus)
void	getWithDelay()	string
void	getValidity()	string
string	CELCondition() This method constructs a new object and gets as input only the identifier.	void
string	setIdentifier()	void
enum (actionStatus)	setActionStatus()	void
string	setWithDelay()	void
string	setValidity()	void

8.3.6 MCODEonticExpression

Class		CEL	MCO
Name	Type and Hierarchy		
MCODEonticExpression	Concrete, sub-class of Deontic	/	mco-core: DeonticExpression
Fields			
Type	Field and Description	Occ.	
/	/	/	/

Methods		
Parameters	Method and Description	Return
void	getRequiredFacts() This method calls the super-method getConstraints().	map(factIdref, Fact)
string	MCODEonticExpression() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void

Methods		
Parameters	Method and Description	Return
string, enum (type), actionIdref, partyIdref	MCODeonticExpression() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
map(factIdref, Fact)	setRequiredFacts() This method calls the super-method setConstraints().	void
factIdref, Fact	addRequiredFacts() This method calls the super-method addConstraints().	void

8.3.7 Permission

Class		CEL	MCO
Name	Type and Hierarchy		
Permission	Abstract, sub-class of Deontic	<i>cel-ipre:Permission</i>	<i>mco:Permission</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 1 = Permission (super-class field override)	1	<i>cel-core:Permission</i> <i>mco-core:Permission</i>
float	percentage Indicates when the permission is shared with other parties	0, 1	<i>cel-core:Permission</i> <i>percentage (attribute)</i> <i>mco-ipre:hasPercentage</i>
float	incomePercentage Indicates when the income of the exploitation of the permission has to be shared	0, 1	<i>cel-core:Permission</i> <i>incomePercentage (attribute)</i> <i>mco-ipre:hasIncomePercentage</i>
boolean	isExclusive Indicates if this exploitation might be granted by the issuer to multiple licensees in the same context or not	0, 1	<i>cel-core:Permission</i> <i>isExclusive (attribute)</i> <i>mco-ipre:isExclusive</i>
boolean	hasSublicenseRight Indicates if it is possible to sublicense a granted right	0, 1	<i>cel-core:Permission</i> <i>sublicenseRight (attribute)</i> <i>mco-ipre:hasSublicenseRight</i>

Methods		
Parameters	Method and Description	Return
void	getPercentage()	float
void	getIncomePercentage()	float
void	getIsExclusive()	boolean
void	getHasSublicenseRight()	boolean
string	Permission() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void

Methods		
Parameters	Method and Description	Return
string, actionIdref, partyIdref	Permission() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
float	setPercentage()	void
float	setIncomePercentage()	void
boolean	setIsExclusive()	void
boolean	setHasSublicenseRight()	void

8.3.8 CELPermission

Class		CEL	MCO
Name	Type and Hierarchy		
Permission	Concrete, sub-class of Permission, sub-class of CELDeonticStructuredClause	<i>cel-ipre:Permission</i>	/
Fields			
Type	Field and Description	Occ.	
/	/	/	/

Methods		
Parameters	Method and Description	Return
string	CELPermission() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, actionIdref, partyIdref	CELPermission() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void

8.3.9 MCOPermission

Class		CEL	MCO
Name	Type and Hierarchy		
MCOPermission	Concrete, sub-class of Permission, sub-class of MCOPermission	/	<i>mvco:Permission</i>
Fields			
Type	Field and Description	Occ.	
/	/	/	/

Methods		
Parameters	Method and Description	Return
string	MCOPermission() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void

Methods		
Parameters	Method and Description	Return
string, actionIdref, partyIdref	MCOPermission() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void

8.4 Action

8.4.1 Act/GenericAction/Action

Class		CEL	MCO
Name	Type and Hierarchy		
Action	Concrete	<i>cel-core:Act</i>	<i>mco-core:GenericAction</i> <i>mvco:Action</i>
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the action	1	<i>cel-core:Act id (attribute)</i> <i>rdf:about</i>
enum (type)	type 0 = Action (simple) 1 = CreateWork 2 = Distribute 3 = EndUserAction 4 = ModifyCopy 5 = Dub 6 = MoveContent 7 = Render 8 = MakeAdaptation 9 = MakeCopy (Duplicate)	1	<i>cel-core</i> <i>cel-ipre</i> <i>cel-pane</i> <i>cel-rele</i> <i>mco-core</i> <i>mco-ipre</i> <i>mco-pane</i> <i>mco-rele</i> <i>mvco</i>

Class		CEL	MCO
Name	Type and Hierarchy		
	10 = MakeAdaptationInstanceCopy		
	11 = MakeAdaptationManifestationCopy		
	12 = MakeWorkInstanceCopy		
	13 = MakeWorkManifestationCopy		
	14 = MakeInstance (Fixate)		
	15 = MakeAdaptationInstance		
	16 = MakeWorkInstance		
	17 = MakeManifestation		
	18 = MakeAdaptationManifestation		
	19 = MakeWorkManifestation		
	20 = Produce		
	21 = PublicCommunication		
	22 = Broadcast		
	23 = Download		
	24 = Stream		
	25 = CommunicationToThePublic		
	26 = PublicPerformance		
	27 = Synchronise		
	28 = GenericAction		
	29 = Reuse		
	30 = ExploitIPRights		
	31 = Transform		
	32 = Translate		
	33 = MakeCutAndEdit		
	34 = MakeExcerpt		
	35 = MakeRadioProduct		
	36 = Remix		
	37 = CreativeTransform		
	38 = Novelization		
	39 = Prequel		
	38 = Sequel		
	39 = Remake		
	40 = Spinoff		

IECNORM.COM Click to view the full PDF of ISO/IEC 21000-23:2022

Class		CEL	MCO
Name	Type and Hierarchy		
	41 = RelAct 42 = Issue 43 = Obtain 44 = PossessProperty 45 = Revoke 46 = Rel-mx-Act 47 = Rel-sx-Act 48 = Adapt 49 = Delete 50 = Diminish 51 = Embed 52 = Enhance 53 = Enlarge 54 = Execute 55 = Install 56 = Modify 57 = Move 58 = Play 59 = Print 60 = Reduce 61 = Uninstall 62 = RightUri 63 = Trade 64 = Consume 65 = Match 66 = Provide 67 = Payment 68 = Notify 69 = UserDefinedAction		
actionIdref[]	impliesAlso Other actions implied	0, n	/
personuserIdref[]	rightGivenBy The Persons/Users that provide the right to perform the action	0, n	/
			<i>mvco:impliesAlso</i>
			<i>mvco:rightGivenBy</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getType()	enum (type)

Methods		
Parameters	Method and Description	Return
void	getImpliesAlso()	actionIdref[]
void	getRightGivenBy()	personuserIdref[]
string	Action() This method constructs a new object and gets as input only the identifier.	void
string, enum (type)	Action() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
enum (type)	setType()	void
actionIdref[]	setImpliesAlso()	void
actionIdref	addImpliesAlso()	void
personuserIdref[]	getRightGivenBy()	void
personuserIdref	addRightGivenBy()	void

8.4.2 Trade

Class		CEL	MCO
Name	Type and Hierarchy		
Trade	Concrete, sub-class of Action	/	<i>mco-core:Trade</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 63 = Trade (super-class field override)	1	/
deonticIdref	sellsDeontic	1	/
			<i>mco-core:Trade</i>
			<i>mco-ipre:sellsDeontic</i>

Methods		
Parameters	Method and Description	Return
void	getSellsDeontic()	deonticIdref
string	Trade() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, deonticIdref	Trade() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
deonticIdref	setSellsDeontic()	void

8.4.3 Provide

Class		CEL	MCO
Name	Type and Hierarchy		
Provide	Concrete, sub-class of Action	<i>cel-core:Provide</i>	<i>mco-core:Provide</i>

Class		CEL	MCO
Name	Type and Hierarchy		
Fields			
Type	Field and Description	Occ.	
enum (type)	type 66 = Provide (super-class field override)	1	/
boolean	isOnLoan	0, 1	<i>cel-core</i> :Provide isOnLoan (attribute)
partyRef[]	recipients	0, n	<i>cel-core</i> :Provide recipients (attribute)
			<i>mco-core</i> : Provide <i>mco-core</i> :isOnLoan <i>mco-core</i> : hasRecipient

Methods		
Parameters	Method and Description	Return
void	getIsOnLoan()	boolean
void	getRecipients()	partyRef[]
string	Provide() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
boolean	setIsOnLoan()	void
partyRef[]	setRecipients()	void
partyRef	addRecipients()	void

8.4.4 Payment

Class		CEL	MCO
Name	Type and Hierarchy		
Payment	Concrete, sub-class of Action	<i>cel-pane</i> :Payment	<i>mco-pane</i> :Payment
Fields			
Type	Field and Description	Occ.	
enum (type)	type 67 = Payment (super-class field override)	1	<i>cel-pane</i> :Payment
partyIdref[]	beneficiaries	1, n	<i>cel-pane</i> :Beneficiary
actionIdref[]	incomeSources	0, n	<i>cel-pane</i> : IncomeSource
float	amount	0, 1	<i>cel-pane</i> :Payment amount (attribute)
string	currency	0, 1	<i>cel-pane</i> :Payment currency (attribute)
			<i>mco-pane</i> : Payment <i>mco-pane</i> :hasBeneficiary <i>mco-pane</i> :hasIncomeSource <i>mco-pane</i> :hasAmount <i>mco-pane</i> :hasCurrency

Class		CEL	MCO
Name	Type and Hierarchy		
float	incomePercentage	0, 1	<i>cel-pane:Payment</i> incomePercentage (attribute) <i>mco-pane:</i> hasIncomePercentage

Methods		
Parameters	Method and Description	Return
void	getBeneficiaries()	partyIdref[]
void	getIncomeSources()	actionIdref[]
void	getAmount()	float
void	getCurrency()	string
void	getIncomePercentage()	float
string	Payment() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, partyIdref[]	Payment() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
partyIdref[]	setBeneficiaries()	void
partyIdref	addBeneficiaries()	void
actionIdref[]	setIncomeSources()	void
actionIdref	addIncomeSources()	void
float	setAmount()	void
string	setCurrency()	void
float	setIncomePercentage()	void

8.4.5 Notify

Class		CEL	MCO
Name	Type and Hierarchy		
Notify	Concrete, sub-class of Action	<i>cel-pane:Notify</i>	<i>mco-pane:Notify</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 68 = Notify (super-class field override)	1	<i>cel-pane:Notify</i> <i>mco-pane:</i> Notify
partyIdref[]	recipients	1, n	<i>cel-pane:Recipient</i> <i>mco-pane:</i> hasRecipient
actionIdref[]	isAbout	0, n	<i>cel-pane>About</i> <i>mco-pane:isAbout</i>

Methods		
Parameters	Method and Description	Return
void	getRecipients()	partyIdref[]

Methods		
Parameters	Method and Description	Return
void	getIsAbout()	actionIdref[]
string	Notify() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, partyIdref[]	Notify() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
partyIdref[]	setRecipients()	void
partyIdref	addRecipients()	void
actionIdref[]	setIsAbout()	void
actionIdref	addIsAbout()	void

8.4.6 UserDefinedAction

Class		CEL	MCO
Name	Type and Hierarchy		
UserDefinedAction	Concrete, sub-class of Action	<i>cel-core:</i> UserDefinedAction	/
Fields			
Type	Field and Description	Occ.	
enum (type)	type 69 = UserDefinedAction (super-class field override)	1	<i>cel-core:</i> UserDefinedAction
string	href	0, 1	<i>cel-core:</i> UserDefinedAction href (attribute)
string	name	1	<i>cel-core:</i> Name
string	standardReference	0, 1	<i>cel-core:</i> StandardReference
string	definition	0, 1	<i>cel-core:</i> Definition

Methods		
Parameters	Method and Description	Return
void	getHref()	string
void	getName()	string
void	getStandardReference()	string
void	getDefinition()	string
string	UserDefinedAction() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	UserDefinedAction() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void

Methods		
Parameters	Method and Description	Return
string	setHref()	void
string	setName()	void
string	setStandardReference()	void
string	setDefinition()	void

8.5 Object

8.5.1 Object

Class		CEL	MCO
Name	Type and Hierarchy		
Object	Abstract	<i>cel-core:Object</i>	/
Fields			
Type	Field and Description	Occ.	
/	/	/	/

Methods		
Parameters	Method and Description	Return
/	/	/

8.5.2 Item

Class		CEL	MCO
Name	Type and Hierarchy		
Item	Concrete, sub-class of Object	<i>cel-core:Item</i>	/
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the item	0, 1	<i>dii:Identifier</i> dii:Identifier
string[]	relatedIdentifiers	0, n	<i>dii:RelatedIdentifier</i> dii:RelatedIdentifier
string	name	0, 1	<i>cel-core:Item</i> name (attribute)

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getRelatedIdentifiers()	string[]
void	getName()	string
string	Item() This method constructs a new object and gets as input only the identifier.	void
string	setIdentifier()	void
string[]	setRelatedIdentifiers()	void

Methods		
Parameters	Method and Description	Return
string	addRelatedIdentifiers()	void

8.5.3 IPEntity

Class		CEL	MCO	
Name	Type and Hierarchy			
IPEntity	Concrete, sub-class of Item	/	mvco:IPEntity	
Fields				
Type	Field and Description	Occ.		
enum (type)	type 0 = IPEntity (simple) 1 = Adaptation 2 = Excerpt 3 = Copy 4 = AdaptationInstanceCopy 5 = AdaptationManifestationCopy 6 = WorkInstanceCopy 7 = WorkManifestationCopy 8 = Instance 9 = AdaptationInstance 10 = WorkInstance 11 = Manifestation 12 = AdaptationManifestation 13 = WorkManifestation 14 = Product 15 = Work 16 = Event 17 = Segment	1	/	mvco mco-core:Event avco:Segment
map (string, string)	metadata For giving information about the contract itself.	0, n	cel-core:Metadata Dublin Core (ISO 15836) or others	Dublin Core (ISO 15836) or others
string	socialTag	0, 1	/	mvco:hasSocialTag
boolean	isDigital	0, 1	/	avco:hasSocialTag
personuserIdref[]	rightsOwners	0, n	/	mvco:hasRightsOwner
ipentityIdref[]	isMadeUpOf	0, n	/	mvco:isMadeUpOf
actionIdref[]	resultedFrom	0, n	/	mvco:resultedFrom
boolean	isAudio	0, 1	/	avco:isAudio
ipentityIdref[]	segments	0, n	/	avco:hasSegment
trackIdref[]	tracks	0, n	/	avco:hasTrack
intervalIdref[]	intervals	0, n	/	avco:interval

Methods		
Parameters	Method and Description	Return
void	getType()	enum (type)
void	getMetadata()	map (string, string)
string	getMetadataBy()	string
void	getSocialTag()	string
void	getIsDigital()	boolean
void	getRightsOwners()	personuserIdref[]
void	getIsMadeUpOf()	ipentityIdref[]
void	getResultFrom()	actionIdref[]
void	getIsAudio()	boolean
void	getSegments()	ipentityIdref[]
void	getTracks()	trackIdref[]
void	getIntervals()	intervalIdref[]
string	IPEntity() This method constructs a new object and gets as input only the identifier.	void
string, enum (type)	IPEntity() This method constructs a new object and gets as input all the required fields.	void
enum (type)	setType()	void
map (string, string)	setMetadata()	void
string, string	addMetadata()	void
string	setSocialTag()	void
boolean	setIsDigital()	void
personuserIdref[]	setRightsOwners()	void
personuserIdref	addRightsOwners()	void
ipentityIdref[]	setIsMadeUpOf()	void
ipentityIdref	addIsMadeUpOf()	void
actionIdref[]	setResultFrom()	void
actionIdref	addResultFrom()	void
boolean	setIsAudio()	void
ipentityIdref[]	setSegments()	void
ipentityIdref	addSegments()	void
trackIdref[]	setTracks()	void
trackIdref	addTracks()	void
intervalIdref[]	setIntervals()	void
intervalIdref	addIntervals()	void

8.5.4 Event

Class		CEL	MCO
Name	Type and Hierarchy		
Event	Concrete, sub-class of IPEntity	<i>cel-core:Event</i>	<i>mco-core:Event</i>
Fields			
Type	Field and Description	Occ.	

Class		CEL	MCO
Name	Type and Hierarchy		
enum (type)	type 16 = Event (super-class field override)	1	<i>cel-core:Event</i> <i>mco-core:Event</i>

Methods		
Parameters	Method and Description	Return
string	Event() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void

8.5.5 Segment

Class		CEL	MCO
Name	Type and Hierarchy		
Segment	Concrete, sub-class of IPEntity	/	<i>avco:Segment</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 17 = Segment (super-class field override)	1	/
ipentityIdref	segmentOf	0, 1	/
ipentityIdref[]	contains	0, n	/
trackIdref[]	onTrack	0, n	/

Methods		
Parameters	Method and Description	Return
void	getSegmentOf()	ipentityIdref
void	getContains()	ipentityIdref[]
void	getOnTrack()	trackIdref[]
string	Segment() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
ipentityIdref	setSegmentOf()	void
ipentityIdref[]	setContains()	void
ipentityIdref	addContains()	void
trackIdref[]	setOnTrack()	void
trackIdref	addOnTrack()	void

8.5.6 Service

Class		CEL	MCO
Name	Type and Hierarchy		
Service	Concrete, sub-class of Object	<i>cel-core:Service</i>	<i>mco-core:Service</i>

Class		CEL	MCO
Name	Type and Hierarchy		
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the service	1	/
enum (type)	type 0 = Service (simple) 1 = Authenticate 2 = Deliver 3 = Describe 4 = Identify 5 = InteractWith 6 = Package 7 = Post 8 = Present 9 = Process 10 = Store 11 = Verify	1	<i>cel-core:Service</i> / <i>mco-core:Service</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getType()	enum (type)
string	Service() This method constructs a new object and gets as input only the identifier.	void
string, enum (type)	Service() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
enum	setType()	void

8.5.7 SubjectWrapperObject

Class		CEL	MCO
Name	Type and Hierarchy		
SubjectWrapperObject	Concrete, sub-class of Object	<i>cel-core:Subject</i>	<i>mco-core:Party</i>
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the SubjectWrapperObject	1	/

Class		CEL	MCO
Name	Type and Hierarchy		
partyIdref	partyRefersTo The party referred as object	1	<i>cel-core:Subject</i> / <i>mco-core:Party</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getPartyRefersTo()	partyIdref
string	SubjectWrapperObject() This method constructs a new object and gets as input only the identifier.	void
string, partyIdref	SubjectWrapperObject() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
partyIdref	setPartyRefersTo()	void

8.5.8 Track

Class		CEL	MCO
Name	Type and Hierarchy		
Track	Concrete, sub-class of Object	/	<i>avco:Track</i>
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the track	1	/ / <i>rdf:about</i>
ulong	trackNumber	1	/ / <i>avco:trackNumber</i>

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getTrackNumber()	ulong
string	Track() This method constructs a new object and gets as input only the identifier.	void
string, ulong	Track() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
ulong	setTrackNumber()	void

8.5.9 Interval

Class		CEL	MCO
Name	Type and Hierarchy		
Interval	Concrete, sub-class of Object		/
			<i>avco</i> :Interval
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the interval	1	/
			<i>rdf</i> :about
string	start	1	/
			<i>avco</i> :start
string	end	1	/
			<i>avco</i> :end
string	duration	1	/
			<i>avco</i> :duration
string	onTimeline	0, 1	/
			<i>avco</i> :onTimeline <i>avco</i> :Timeline

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getStart()	string
void	getEnd()	string
void	getDuration()	string
void	getOnTimeline()	string
string	Interval() This method constructs a new object and gets as input only the identifier.	void
string, string, string, string	Interval() This method constructs a new object and gets as input all the required fields.	void
string	setIdentifier()	void
string	setStart()	void
string	setEnd()	void
string	setDuration()	void
string	setOnTimeline()	void

8.6 Fact

8.6.1 Constraint/Fact

Class		CEL	MCO
Name	Type and Hierarchy		
Fact	Concrete	<i>cel-core</i> :Constraint, <i>cel-core</i> :Fact	<i>mvco</i> :Fact
Fields			
Type	Field and Description	Occ.	
string	identifier Uniquely identifies the interval	1	/
			<i>rdf</i> :about

Class		CEL	MCO
Name	Type and Hierarchy		
	29 = ExistsRight		
	30 = FeeFlat		
	31 = FeeMetered		
	32 = FeePerInterval		
	33 = FeePerUse		
	34 = FeePerUsePrePay		
	35 = Fulfiller		
	36 = Helper		
	37 = IsMarked		
	38 = Mark		
	39 = PrerequisiteRight		
	40 = ProhibitedAttributeChanges		
	41 = ResourceSignedBy		
	42 = RevocationFreshness		
	43 = SeekApproval		
	44 = Source		
	45 = Territory		
	46 = TrackQuery		
	47 = TrackReport		
	48 = Transaction		
	49 = TransferControl		
	50 = ValidityInterval		
	51 = ValidityIntervalFloating		
	52 = ValidityIntervalStartsNow		
	53 = ValidityTimeMetered		
	54 = ValidityTimePeriodic		
boolean	isTrue	0, 1	/ mvco:isTrue

Methods		
Parameters	Method and Description	Return
void	getIdentifier()	string
void	getType()	enum (type)
void	getIsTrue()	boolean
string	Fact() This method constructs a new object and gets as input only the identifier.	void
string, enum (type)	Fact() This method constructs a new object and gets as input all the required fields.	void

Methods		
Parameters	Method and Description	Return
string	setIdentifier()	void
enum (type)	setType()	void
boolean	setIsTrue()	void

8.6.2 FactComposition

Class		CEL	MCO
Name	Type and Hierarchy		
FactComposition	Concrete, sub-class of Fact	<i>cel-core:FactNegation</i> <i>cel-core:FactIntersection</i> <i>cel-core:FactUnion</i>	<i>mco-core:FactComposition</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 1 = FactComposition (super-class field override)	1	/
enum (compositionType)	compositionType 0 = Negation 1 = Union 2 = Intersection	1	<i>cel-core:FactNegation</i> <i>cel-core:FactIntersection</i> <i>cel-core:FactUnion</i>
factIdref[]	composedFacts	1..n	/
			<i>mco-core:hasFact</i>

Methods		
Parameters	Method and Description	Return
void	getCompositionType()	enum(composition-Type)
void	getComposedFacts()	factIdref[]
string	FactComposition() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(compositionType), factIdref[]	FactComposition() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum(composition-Type)	setCompositionType()	void
factIdref[]	setComposedFacts()	void
factIdref	addComposedFacts()	void

8.6.3 ActionEventFact

Class		CEL	MCO
Name	Type and Hierarchy		
ActionEventFact	Concrete, sub-class of Fact	<i>cel-core:</i> ActionEventFact	<i>mco-core:</i> ActionEventFact
Fields			
Type	Field and Description	Occ.	
enum (type)	type 2 = ActionEventFact (super-class field override)	1	<i>cel-core:</i> ActionEventFact
string	validity	0, 1	<i>cel-core:</i> ActionEventFact validity (attribute)
string	withDelay	0, 1	<i>cel-core:</i> ActionEventFact withDelay (attribute)
enum (status)	status Indicates the status of the action or the event to which the fact refers. The enum value must be: 0 = Started 1 = Done	1	<i>cel-core:</i> ActionEventFact status (attribute)
ipentitYIdref	eventThatMakesTrue	0, 1	<i>cel-core:</i> ActionEventFact ref (attribute)
actionIdref	actionThatMakesTrue	0, 1	<i>cel-core:</i> ActionEventFact ref (attribute)

Methods		
Parameters	Method and Description	Return
void	getValidity()	string
void	getWithDelay()	string
void	getStatus()	enum (status)
void	getEventThatMakesTrue()	ipentitYIdref
void	getActionThatMakesTrue()	actionIdref
string	ActionEventFact() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void

Methods		
Parameters	Method and Description	Return
string, enum(status)	ActionEventFact() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string	setValidity()	void
string	setWithDelay()	void
enum (status)	setActionStatus()	void
ipentityIdref	setEventThatMakesTrue()	void
actionIdref	setActionThatMakesTrue()	void

8.6.4 TogetherWith

Class		CEL	MCO
Name	Type and Hierarchy		
TogetherWith	Concrete, sub-class of Fact	<i>cel-core:</i> TogetherWith	<i>mco-core:</i> TogetherWith
Fields			
Type	Field and Description	Occ.	
enum (type)	type 3 = TogetherWith (super-class field override)	1	<i>cel-core:</i> TogetherWith
ipentityIdref	withIPEntity	0, 1	<i>mco-ipre:</i> withIPEntity

Methods		
Parameters	Method and Description	Return
void	getWithIPEntity()	ipentityIdref
string	TogetherWith() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
ipentityIdref	setWithIPEntity()	void

8.6.5 AccessPolicy

Class		CEL	MCO
Name	Type and Hierarchy		
AccessPolicy	Concrete, sub-class of Fact	<i>cel-ipre:</i> AccessPolicy	<i>mco-ipre:</i> AccessPolicy
Fields			
Type	Field and Description	Occ.	
enum (type)	type 5 = AccessPolicy (super-class field override)	1	<i>cel-ipre:</i> AccessPolicy
			<i>mco-ipre:</i> AccessPolicy

Class		CEL	MCO
Name	Type and Hierarchy		
enum (restriction)	restriction 0 = FreeOfCharge 1 = Pay 2 = PayPerPackage 3 = PayPerView 4 = Subscription	1	<i>cel-ipre:</i> AccessPolicyType <i>mco-ipre:</i> AccessPolicy sub-classes

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)
string	AccessPolicy() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	AccessPolicy() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void

8.6.6 DeliveryModality

Class		CEL	MCO
Name	Type and Hierarchy		
DeliveryModality	Concrete, sub-class of Fact	<i>cel-ipre:</i> DeliveryModality	<i>mco-ipre:</i> DeliveryModality
Fields			
Type	Field and Description	Occ.	
enum (type)	type 7 = DeliveryModality (super-class field override)	1	<i>cel-ipre:</i> DeliveryModality <i>mco-ipre:</i> DeliveryModality
enum (restriction)	restriction 0 = Linear 1 = Broadcasting 2 = Webcasting 3 = NonLinear 4 = OnDemandBasis 5 = OnDemandDownload 6 = OnDemandStreaming	1	<i>cel-ipre:</i> DeliveryModalityType <i>mco-ipre:</i> DeliveryModality sub-classes

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)

Methods		
Parameters	Method and Description	Return
string	DeliveryModality() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	DeliveryModality() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void

8.6.7 Device

Class		CEL	MCO
Name	Type and Hierarchy		
Device	Concrete, sub-class of Fact	<i>cel-ipre:Device</i>	<i>mco-ipre:Device</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 8 = Device (super-class field override)	1	<i>cel-ipre:Device</i> <i>mco-ipre:Device</i>
enum (restriction)	restriction 0 = Computer 1 = MobileDevice 2 = MobileBroadcastDevice 3 = MobileTelecommunicationDevice 4 = RobotDevice 5 = StorageDevice 6 = TelevisionDevice 7 = TelevisionSet	1	<i>cel-ipre:DeviceType</i> <i>mco-ipre:Device</i> sub-classes

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)
string	Device() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	Device() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void

8.6.8 IPEntityContext

Class		CEL	MCO
Name	Type and Hierarchy		
IPEntityContext	Concrete, sub-class of Fact	<i>cel-ipre:</i> IPEntityContext	<i>mco-ipre:</i> IPEntityContext
Fields			
Type	Field and Description	Occ.	
enum (type)	type 9 = IPEntityContext (super-class field override)	1	<i>cel-ipre:</i> IPEntityContext
ipentityIdref[]	partOf	1	<i>mco-ipre:</i> partOf

Methods		
Parameters	Method and Description	Return
void	getPartOf()	ipentityIdref[]
string	IPEntityContext() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, ipentityIdref[]	IPEntityContext() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
ipentityIdref[]	setPartOf()	void
ipentityIdref	addPartOf()	void

8.6.9 Language

Class		CEL	MCO
Name	Type and Hierarchy		
Language	Concrete, sub-class of Fact	<i>cel-ipre:</i> Language	<i>mco-ipre:</i> Language
Fields			
Type	Field and Description	Occ.	
enum (type)	type 10 = Language (super-class field override)	1	<i>cel-ipre:</i> Language
string[]	languages	1	<i>cel-ipre:</i> Language languages (attribute) <i>mco-ipre:</i> hasLanguages

Methods		
Parameters	Method and Description	Return
void	getLanguages()	string[]

Methods		
Parameters	Method and Description	Return
string	Language() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string[]	Language() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string[]	setLanguages()	void
string	addLanguages()	void

8.6.10 Length

Class		CEL	MCO
Name	Type and Hierarchy		
Length	Concrete, sub-class of Fact	<i>cel-ipre:Length</i>	<i>mco-ipre:Length</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 11 = Length (super-class field override)	1	<i>cel-ipre:Length</i> <i>mco-ipre:Length</i>
string	maxLength	1	<i>cel-ipre:Length</i> maxLength (attribute) <i>mco-ipre:hasMaxLength</i>

Methods		
Parameters	Method and Description	Return
void	getMaxLength()	string
string	Length() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	Length() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string	setMaxLength()	void

8.6.11 MaterialFormat

Class		CEL	MCO
Name	Type and Hierarchy		
Material-Format	Concrete, sub-class of Fact	<i>cel-ipre:MaterialFormat</i>	<i>mco-ipre:MaterialFormat</i>
Fields			
Type	Field and Description	Occ.	

Class		CEL	MCO
Name	Type and Hierarchy		
enum (type)	type 12 = MaterialFormat (super-class field override)	1	<i>cel-ipre:</i> MaterialFormat <i>mco-ipre:</i> MaterialFormat
string	matchesFormatComplianceProfile	0, 1	<i>cel-ipre:</i> MatchesFormatComplianceProfile <i>mco-ipre:</i> matchesFormatComplianceProfile
string	aspectRatio	0, 1	<i>cel-ipre:</i> AspectRatio <i>mco-ipre:</i> hasAspectRatio
string	audioFormat	0, 1	<i>cel-ipre:</i> AudioFormat <i>mco-ipre:</i> hasAudioFormat
string	format	0, 1	<i>cel-ipre:</i> Format <i>mco-ipre:</i> hasFormat
ulong	maxBitrate	0, 1	<i>cel-ipre:</i> MaxBitrate <i>mco-ipre:</i> hasMaxBitrate
ulong	maxLines	0, 1	<i>cel-ipre:</i> MaxLines <i>mco-ipre:</i> hasMaxLines
ulong	minBitrate	0, 1	<i>cel-ipre:</i> MinBitrate <i>mco-ipre:</i> hasMinBitrate
ulong	minLines	0, 1	<i>cel-ipre:</i> MinLines <i>mco-ipre:</i> hasMinLines
string	videoFormat	0, 1	<i>cel-ipre:</i> VideoFormat <i>mco-ipre:</i> hasVideoFormat

Methods		
Parameters	Method and Description	Return
void	getMatchesFormatComplianceProfile()	string
void	getAspectRatio()	string
void	getAudioFormat()	string
void	getFormat()	string
void	getMaxBitrate()	ulong
void	getMaxLines()	ulong
void	getMinBitrate()	ulong
void	getMinLines()	ulong
void	getVideoFormat()	string
string	MaterialFormat() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string	setMatchesFormatComplianceProfile()	void
string	setAspectRatio()	void
string	setAudioFormat()	void
string	setFormat()	void
ulong	setMaxBitrate()	void
ulong	setMaxLines()	void
ulong	setMinBitrate()	void
ulong	setMinLines()	void
string	setVideoFormat()	void

8.6.12 Means

Class		CEL	MCO
Name	Type and Hierarchy		
Means	Concrete, sub-class of Fact		<i>cel-ipre:Means</i> <i>mco-ipre:Means</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 13 = Means (super-class field override)	1	<i>cel-ipre:Means</i> <i>mco-ipre:Means</i>
enum (restriction)	restriction 0 = Videogram 1 = TransmissionTechnology 2 = BroadcastTechnology 3 = Cable 4 = IPNetwork 5 = MobileBroadcastTechnology 6 = Satellite 7 = Terrestrial 8 = Internet 9 = MobileTechnology 10 = MobileTelecommunicationTechnology	1	<i>cel-ipre:MeansType</i> <i>mco-ipre:Means</i> sub-classes

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)
string	Means() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	Means() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void

8.6.13 Runs

Class		CEL	MCO
Name	Type and Hierarchy		
Runs	Concrete, sub-class of Fact		<i>cel-ipre:Runs</i> <i>mco-ipre:Runs</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 14 = Runs (super-class field override)	1	<i>cel-ipre:Runs</i> <i>mco-ipre:Runs</i>

Class		CEL	MCO
Name	Type and Hierarchy		
ulong	numberOfRuns	1	<i>cel-ipre:Runs</i> numberOfRuns (attribute) <i>mco-ipre:</i> hasNumberOfRuns
string	validity	0, 1	<i>cel-ipre:Runs</i> validity (attrib- ute) <i>mco-ipre:</i> hasValidity
ulong	numberOfRepetitions	0, 1	<i>cel-ipre:Runs</i> numberOfRepeti- tions (attribute) <i>mco-ipre:</i> ha sNumberOfR epetitions

Methods		
Parameters	Method and Description	Return
void	getNumberOfRuns()	ulong
void	getValidity()	string
void	getNumberOfRepetitions()	ulong
string	Runs() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, ulong	Runs() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
ulong	setNumberOfRuns()	void
string	setValidity()	void
ulong	setNumberOfRepetitions()	void

8.6.14 ServiceAccessPolicy

Class		CEL	MCO
Name	Type and Hierarchy		
ServiceAccessPolicy	Concrete, sub-class of Fact		<i>cel-ipre:</i> ServiceAccessPolicy <i>mco-ipre:</i> ServiceAccessPolicy
Fields			
Type	Field and Description	Occ.	
enum (type)	type 15 = ServiceAccessPolicy (super-class field override)	1	<i>cel-ipre:</i> ServiceAccessPolicy <i>mco-ipre:</i> ServiceAccessPolicy

Class		CEL	MCO
Name	Type and Hierarchy		
enum (restriction)	restriction 0 = Open 1 = Restricted 2 = Hotel 3 = PublicPerformanceHalls 4 = Transportations 5 = Airplanes 6 = BusesMetro 7 = Ships 8 = Trains	1	<i>cel-ipre:ServiceAccessPolicyType</i> <i>mco-ipre:ServiceAccessPolicy</i> sub-classes

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)
string	ServiceAccessPolicy() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	ServiceAccessPolicy() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void

8.6.15 ServiceChannelContext

Class		CEL	MCO
Name	Type and Hierarchy		
ServiceChannelContext	Concrete, sub-class of Fact	<i>cel-ipre:ServiceChannelContext</i>	<i>mco-ipre:ServiceChannelContext</i>
Fields			
Type	Field and Description	Occ.	
enum (type)	type 16 = ServiceChannelContext (super-class field override)	1	<i>cel-ipre:ServiceChannelContext</i> <i>mco-ipre:ServiceChannelContext</i>
string[]	servicesAndChannels	0, 1	<i>cel-ipre:ServiceChannelContext</i> <i>mco-ipre:hasServicesAndChannels</i> servicesAndChannels (attribute)

Methods		
Parameters	Method and Description	Return
void	getServicesAndChannels()	string[]
string	ServiceChannelContext() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string[]	setServicesAndChannels()	void
string	addServicesAndChannels()	void

8.6.16 SpatialContext

Class		CEL	MCO
Name	Type and Hierarchy		
SpatialContext	Concrete, sub-class of Fact	<i>cel-ipre:</i> SpatialContext	<i>mco-ipre:</i> SpatialContext
Fields			
Type	Field and Description	Occ.	
enum (type)	type 17 = SpatialContext (super-class field override)	1	<i>cel-ipre:</i> SpatialContext <i>mco-ipre:</i> SpatialContext
string[]	countries	1, n	<i>cel-ipre:</i> Country <i>mco-ipre:</i> inCountry
string[]	regions	0, n	<i>cel-ipre:</i> Region /

Methods		
Parameters	Method and Description	Return
void	getCountries()	string[]
void	getRegions()	string[]
string	SpatialContext() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string[]	SpatialContext() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string[]	setCountries()	void
string	addCountries()	void
string[]	setRegions()	void
string	addRegions()	void

8.6.17 TemporalContext

Class		CEL	MCO
Name	Type and Hierarchy		
TemporalContext	Concrete, sub-class of Fact	<i>cel-ipre:</i> TemporalContext	<i>mco-ipre:</i> TemporalContext
Fields			
Type	Field and Description	Occ.	

Class		CEL	MCO
Name	Type and Hierarchy		
enum (type)	type 18 = TemporalContext (super-class field override)	1	<i>cel-ipre:</i> TemporalContext <i>mco-ipre:</i> TemporalContext
string	afterDate	0, 1	<i>cel-ipre:</i> TemporalContext afterDate (attribute) <i>mco-ipre:</i> afterDate
string	beforeDate	0, 1	<i>cel-ipre:</i> TemporalContext beforeDate (attribute) <i>mco-ipre:</i> beforeDate

Methods		
Parameters	Method and Description	Return
void	getAfterDate()	string
void	getBeforeDate()	string
string	TemporalContext() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string	setAfterDate()	void
string	setBeforeDate()	void

8.6.18 UserTimeAccess

Class		CEL	MCO
Name	Type and Hierarchy		
UserTime-Access	Concrete, sub-class of Fact	<i>cel-ipre:</i> UserTimeAccess	<i>mco-ipre:</i> UserTimeAccess
Fields			
Type	Field and Description	Occ.	
enum (type)	type 19 = UserTimeAccess (super-class field override)	1	<i>cel-ipre:</i> UserTimeAccess <i>mco-ipre:</i> UserTimeAccess
enum (restriction)	restriction 0=Unlimited 1=Limited	1	<i>cel-ipre:</i> Limited <i>cel-ipre:</i> Unlimited <i>mco-ipre:</i> Limited <i>mco-ipre:</i> Unlimited
string	validity	0, 1	<i>cel-ipre:</i> Limited validity (attribute) <i>mco-ipre:</i> hasValidity

Methods		
Parameters	Method and Description	Return
void	getRestriction()	enum (restriction)
void	getValidity()	string

Methods		
Parameters	Method and Description	Return
string	UserTimeAccess() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, enum(restriction)	UserTimeAccess() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
enum (restriction)	setRestriction()	void
string	setValidity()	void

8.6.19 UserDefinedFact

Class		CEL	MCO
Name	Type and Hierarchy		
UserDefinedFact	Concrete, sub-class of Fact	cel-core:UserDefinedFact	/
Fields			
Type	Field and Description	Occ.	
enum (type)	type 20 = UserDefinedAction (super-class field override)	1	cel-core:UserDefinedFact /
string	href	0, 1	cel-core:UserDefinedFact href (attribute) /
string	name	1	cel-core:Name /
string	standardReference	0, 1	cel-core:StandardReference /
string	definition	0, 1	cel-core:Definition /

Methods		
Parameters	Method and Description	Return
void	getHref()	string
void	getName()	string
void	getStandardReference()	string
void	getDefinition()	string
string	UserDefinedFact() This method constructs a new object and gets as input only the identifier. This constructor invokes the superclass constructor.	void
string, string	UserDefinedFact() This method constructs a new object and gets as input all the required fields. This constructor invokes the superclass constructor.	void
string	setHref()	void
string	setName()	void
string	setStandardReference()	void

Methods		
Parameters	Method and Description	Return
string	setDefinition()	void

9 API for MPEG-21 CEL/MCO parser

This clause specifies the API for the MPEG-21 CEL/MCO parser. This API shall be used in conjunction with MPEG-21 XML and RDF media contracts (ISO/IEC 21000-19, ISO/IEC 21000-20, and ISO/IEC 21000-21).

Methods		
Parameters	Method and Description	Return
string	getContractFromCEL() Returns a set of media contractual objects as defined in 7.1.1 from an MPEG-21 CEL contract	Contract
string	getContractFromMCO() Returns a set of media contractual objects as defined in 7.1.1 from an MPEG-21 MCO contract	Contract
string	validateCELContract() Returns a conformance report of the MPEG-21 CEL contract given as input.	string
string	validateMCOContract() Returns a conformance report of the MPEG-21 MCO contract given as input.	string

10 API for MPEG-21 CEL/MCO generator

This clause specifies the API for the MPEG-21 CEL/MCO generator. This API shall be used in conjunction with MPEG-21 XML and RDF media contracts (ISO/IEC 21000-19, ISO/IEC 21000-20, and ISO/IEC 21000-21).

Methods		
Parameters	Method and Description	Return
Contract	getCELFromContract() Returns an MPEG-21 CEL contract from a set of media contractual objects as defined in 7.1.1	string
Contract	getMCOFromContract() Returns an MPEG-21 MCO contract from a set of media contractual objects as defined in 7.1.1	string

11 Reference software and conformance

The reference software is organized with the following structure:

- **MPEG-21 template contracts**
 - *XML*
 - *RDF*

- *JSON-LD*
- **MPEG-21 Contract Expression Language (CEL)**
 - *MPEG-21 CEL parser*
 - *MPEG-21 CEL generator*
 - *MPEG-21 CEL contracts to smart contracts for media (forward conversion)*
 - Solidity/Ethereum¹⁾
 - Michelson/Tezos¹⁾
 - *Smart contracts for media to MPEG-21 CEL contracts (backward conversion)*
 - Solidity/Ethereum
 - Michelson/Tezos
- **MPEG-21 Media Contracts Ontology (MCO)**
 - *MPEG-21 MCO parser*
 - *MPEG-21 MCO generator*
 - *MPEG-21 MCO contracts to smart contracts for media (forward conversion)*
 - Solidity/Ethereum
 - TEAL/Algorand¹⁾
 - *Smart contracts for media to MPEG-21 MCO contracts (backward conversion)*
 - Solidity/Ethereum
 - TEAL/Algorand
- **OpenAPI and demo**
 - *OpenAPI*
 - *MPEG-21 MCO OpenAPI server*
 - *MPEG-21 CEL server*
 - *Demo*

The reference software workflow is shown in [Figure 4](#), including its modules for the bidirectional conversion from MPEG-21 CEL/MCO contracts to smart contracts for media. The complete description of the modules is given in the following subclauses, while the associated ISO/IEC 21000-23 Smart contracts for media reference software can be downloaded at <https://standards.iso.org/iso-iec/21000/-23/ed-1/en/>.

1) Solidity/Ethereum, Michelson/Tezos and TEAL/Algorand are examples of suitable products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of these products.

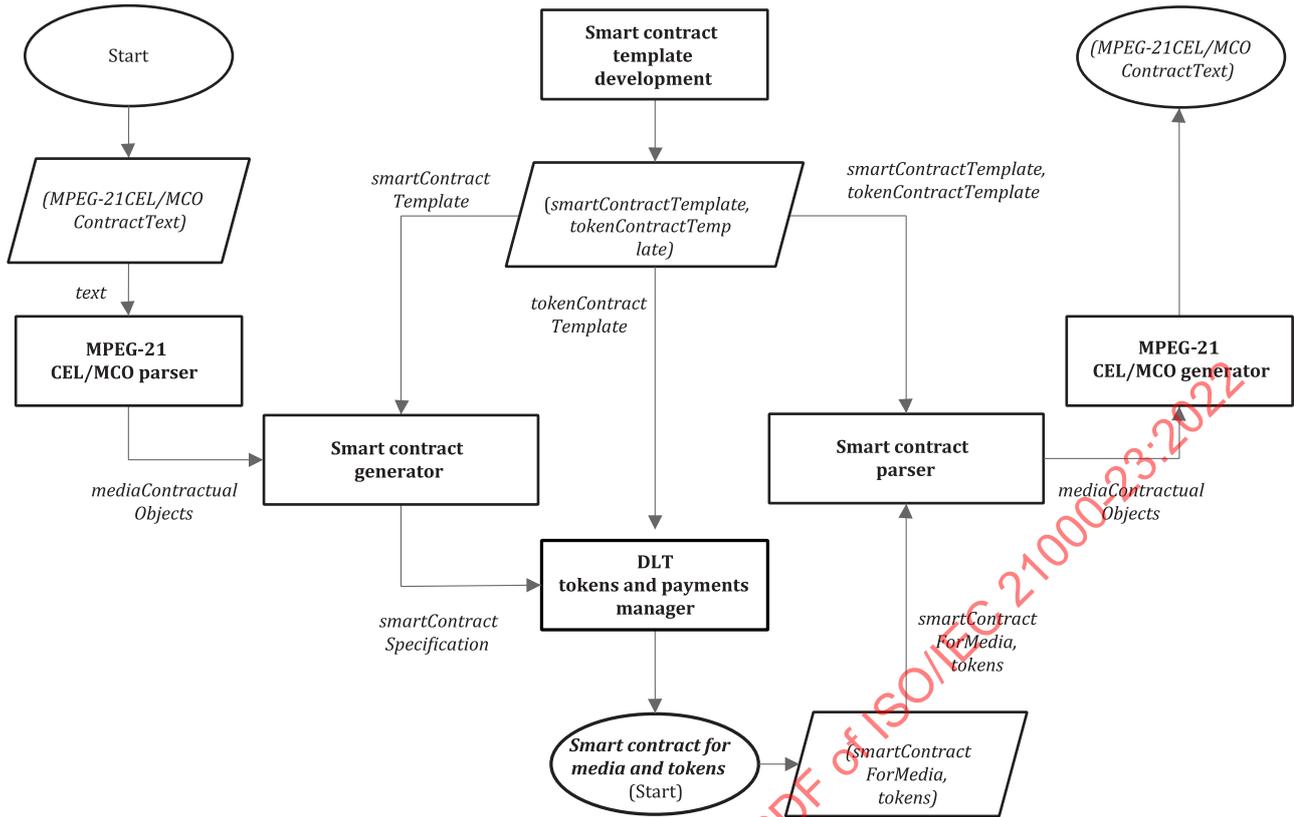


Figure 4 — Reference software workflow modules including inputs and outputs for the bidirectional conversion from MPEG-21 CEL/MCO contracts to smart contracts for media

11.1 MPEG-21 template contracts

The template contracts within the “MPEG-21 template contracts” folder are used as examples for conversion to smart contracts for media. However, these contracts have only informative status. That is, technology providers may similarly create new template contracts or adapt the ones provided as it fits to support their business models.

The collection of MPEG-21 template contracts derives from the Open Music Initiative (OMI) use cases^[7]. The subfolders are:

- **XML** - Contains template contracts expressed in ISO/IEC 21000-20:2016 Contract Expression Language, formatted as XML files.
- **RDF** - Contains template contracts expressed using the ISO/IEC 21000-21:2017 Media Contracts Ontology, formatted as RDF/TURTLE files.
- **JSON-LD** - Contains template contracts expressed using the ISO/IEC 21000-21:2017 Media Contracts Ontology, formatted as JSON-LD files.

11.1.1 Open Music Initiative use cases

Each subfolder in the “MPEG-21 template contracts” folder contains a representation in CEL/MCO of the following contracts:

- **On demand stream “Big labels”** - For record labels that have a direct deal with services.
- **On demand stream “Indie labels”** - For record labels that are represented by a digital aggregator/distributor.