INTERNATIONAL STANDARD

ISO/IEC 21000-5

First edition
2004-04-01

Information technology — Multimedia framework (MPEG-21) —

Part 5: Rights Expression Language

Technologies de l'information — Cadre multimédia (MPEG-21) —

Partie 5: Langage d'expression des droits

Reference number
ISO/IEC 21000-5:2004(E)

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

Published in Switzerland

Contents

Page

Foreword

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

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

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

ISO/IEC 21000-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 21000 consists of the following parts, under the general title *Information technology — Multimedia framework (MPEG-21)*:

— *Part 1: Vision, Technologies and Strategy* [Technical Report]

— *Part 2: Digital Item Declaration*

— *Part 3: Digital Item Identification*

— *Part 5: Rights Expression Language*

— *Part 6: Rights Data Dictionary*

— *Part 7: Digital Item Adaptation*

Introduction

The growth of the Internet has enabled worldwide distribution and consumption of valuable multimedia resources, reduced the cost of doing business, enabled new business models for industry participants, and provided consumers with unprecedented access to high-quality multimedia resources. Internal distribution, external distribution, and retail sales now are conducted on the Internet to establish cost effective, reliable, flexible, highly available, and secure means of managing the delivery of multimedia resources. Consumers routinely search for and download multimedia resources from sources world-wide and can conveniently redistribute those resources. This has fuelled the development of technologies to manage, secure, control, and automate the flow of multimedia resources over the Internet.

The free and convenient flow of multimedia resources through the Internet presents many challenges to content owners and distributors. Before making high-quality and valuable multimedia resources available online, content owners want to be assured that their rights to those resources are respected. In addition, the business models and contracts of content distributors often involve conditions regarding distribution, such as fees, territory restrictions, time limits, and so on.

To meet these requirements, the players involved in the online distribution and consumption of multimedia resources need to exchange information about the rights, terms, and conditions associated with each resource at each step in the multimedia resource lifecycle. For example, a publisher needs to communicate the available consumption rights and the terms and conditions under which those rights may be exercised. To use the multimedia resources, a consumer needs to know the types of usage allowed and the terms and conditions that must be met. In distribution and super distribution business models, this information needs to be communicated to each participant in the distribution chain.

Depending on the business model, expressing rights, terms, and conditions can be simple or complex. In a simple example, a consumer might pay a flat fee to obtain unlimited rights to play a video file. In a more complex example, a video publisher might grant a distributor the right to sell usage rights for classic movie titles to consumers. The distribution agreement might specify the rights that consumers may purchase, the maximum fee the distributor may charge, and a percentage of the fee that must be paid to the publisher.

In an end-to-end system, other considerations such as authenticity and integrity of Rights Expressions become important. For example, any party who issues rights to use or distribute multimedia resources must be identified and authorized. In addition, a Rights Expression may be accessed by different participants during its life cycle, which requires mechanisms and semantics for validating the authenticity and integrity of the Rights Expression.

To address many of these issues, a common Rights Expression Language that can be shared among all participants in this digital workflow is required. A common Rights Expression Language addresses important aspects of the interoperability issues inherent in digital multimedia resource distribution; the issues relating to exchanging Rights Expressions during their life cycle; and the system issues such as trust, authorization, and authentication.

This part of ISO/IEC 21000 addresses a part of the overall vision for ISO/IEC 21000, which is to define a multimedia framework to enable transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities.

Information technology — Multimedia framework (MPEG-21) —

Part 5:
Rights Expression Language

1 Scope

This part of ISO/IEC 21000 specifies the syntax and semantics of a Rights Expression Language.

This part of ISO/IEC 21000 does not give any permission, including permissions about who is legally or technically allowed to create Rights Expressions. It does not specify the security measures of trusted systems, propose specific applications, or describe the details of the systems required for accounting (monetary transactions, state transactions, and so on). It also does not specify if or when Rights Expressions shall be consulted.

However, this part of ISO/IEC 21000 does define an authorization model to specify whether the semantics of a set of Rights Expressions permit a given Principal to perform a given Right upon a given optional Resource during a given time interval based on a given authorization context and a given trust root.

Clause 1 gives the scope of this part of ISO/IEC 21000. Clause 2 gives the normative references. Clause 3 gives pertinent terms, definitions, symbols, and abbreviated terms. Clause 4 gives the namespaces and conventions. Clause 5 specifies the authorization model. Clause 6 defines architectural concepts. Clause 7 specifies the syntax and semantics integral to the architecture. Clause 8 specifies syntax and semantics peripheral to the architecture but still useful in many domains beyond multimedia. Clause 9 specifies syntax and semantics specific to multimedia. Annex A uses W3C XML Schema to normatively specify the syntax of the types and elements defined throughout this part of ISO/IEC 21000. Annex B normatively defines Qualified Names for identifying countries, regions, and currencies. Annex C gives an informative simplified equality algorithm. Annex D gives some example Rights Expressions. Annex E describes the design philosophy for extensions and profiles of this part of ISO/IEC 21000. Annex F demonstrates how to introduce new rights as an extension to this part of ISO/IEC 21000. Annex G gives an example profile of this part of ISO/IEC 21000. Annex H describes the relationship between ISO/IEC 21000-6 and this part of ISO/IEC 21000. Annex I describes the relationship between ISO/IEC 21000-2 and this part of ISO/IEC 21000. Annex J describes an example revocation mechanism and gives a walk-through of revocation.

2 Normative references

The following referenced documents are indispensable for the application 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 3166 (all parts), *Codes for the representation of names of countries and their subdivisions*

ISO 4217, *Codes for the representation of currencies and funds*

ISO/IEC 9594-8, *Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks*

ISO/IEC 10021-2, *Information technology — Message Handling Systems (MHS): Overall architecture*

ISO/IEC 21000 (all parts), *Information technology — Multimedia framework (MPEG-21)*

Request for Comments (RFC) 1034, *Domain Names — Concepts and Facilities*, The Internet Engineering Task Force (IETF), November 1987, available at <http://www.ietf.org/rfc/rfc1034>

RFC 1738, *Uniform Resource Locators (URL)*, IETF, December 1994, available at <http://www.ietf.org/rfc/rfc1738.txt>

RFC 2141, *URN Syntax*, IETF, May 1997, available at <http://www.ietf.org/rfc/rfc2141.txt>

RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, IETF, August 1998, available at <http://www.ietf.org/rfc/rfc2396.txt>

RFC 2822, *Internet Message Format*, IETF, April 2001, available at <http://www.ietf.org/rfc/rfc2822>

ROUTINGABA, *Routing Number Policy*, American Bankers Association Routing Number Administrative Board, November 1996, available at <http://www.tfp.com/text/aba_policy.pdf>

SCC14N, *Schema Centric XML Canonicalization Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS) Universal Description, Discovery, and Integration (UDDI) Technical Committee (TC) Committee Specification, 19 July 2002, available at <http://www.uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm>

UDDIV2API, *UDDI Version 2.04 API Specification*, OASIS Standard, 19 July 2002, available at <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>

UDDIV2DSR, *UDDI Version 2.03 Data Structure Reference*, OASIS Standard, 19 July 2002, available at <http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>

UDDIV3, *UDDI Version 3.0*, OASIS UDDI TC Committee Specification, 19 July 2002, available at <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>

WSDLSPEC, *Web Services Description Language (WSDL) 1.1*, Worldwide Web Consortium (W3C) Note, 15 March 2001, available at <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

XMLSPEC, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2002, available at <http://www.w3.org/TR/2000/REC-xml-20001006>

XMLDSIG, *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002, available at <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212>

XMLENC, *XML-Encryption Syntax and Processing*, W3C Recommendation, 10 December 2002, available at <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210>

XMLINFOSET, *XML Information Set*, W3C Recommendation, 24 October 2001, available at <http://www.w3.org/TR/2001/REC-xml-infoset-20011024>

XMLNAMES, *Namespaces in XML*, W3C Recommendation, 14 January 1999, available at <http://www.w3.org/TR/1999/REC-xml-names-19990114>

XMLSCHEMA, *XML Schema Part 1: Structures and Part 2: Datatypes*, W3C Recommendation, 2 May 2001, available at <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502> and <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>

XPATH, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 16 November 1999, available at <http://www.w3.org/TR/1999/REC-xpath-19991116>

3 Terms, definitions, symbols, and abbreviated terms

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

3.1
Allowable Destination Condition
allowable destination condition as mentioned in 7.4.7

3.2
Allowable Destination Delegation Control
allowable destination delegation control as mentioned in 7.4.7

3.3
Allowable Destination Principal
allowable destination principal as mentioned in 7.4.7

3.4
authorization context
set of properties having the characteristics defined in 5.3

3.5
Authorization Context Member
fifth member

NOTE This term is used with respect to an authorization request, as in "the Authorization Context Member of the authorization request."

3.6
authorization proof
authorization story having the characteristics defined in 5.7

3.7
authorization request
ordered seven-tuple as defined in 5.2

3.8
authorization story
ordered triple as defined in 5.4

3.9
authorizer
ordered five-tuple as defined in 5.5

3.10
Business Service
businessService as defined in UDDIV2DSR and UDDIV3

3.11
conceptually abstract
designated as being subject to the provisions given in 6.6

3.12
Condition
permission limitation identified by an `r:Condition`

NOTE See 4.3 for the meaning of `r:Condition`.

3.13
Conditionally
in a manner subject to a Condition

3.14
Core Namespace
Namespace for the names defined in Clauses 5, 6, and 7

3.15
Derived
derived as defined in 6.9.1

3.16
Equal
equal as defined in 6.1

3.17
License
Rights Expression
expression that is created by Principals to Conditionally or Unconditionally permit the same or other Principals to perform Rights upon Resources

3.18
Match
match as mentioned in 7.4.6.1

3.19
Multimedia Extension Namespace
Namespace for the names defined in Clause 9

3.20
Namespace
XML namespace as defined in XMLNAMES

3.21
performance
carrying out or execution

3.22
primitive grant
`r:Grant` that has no child `r:forAll` element

3.23
Principal
system entity identified by an `r:Principal`

NOTE 1 See 4.3 for the meaning of `r:Principal`.

NOTE 2 Many Users (as introduced in ISO/IEC 21000-1) are Principals.

3.24
Qualified Name
qualified name as defined in XMLNAMES

3.25
Registry
Universal Description, Discovery, and Integration registry as defined in UDDIV2API and UDDIV3

3.26
repository
Principal that can hold Resources

EXAMPLE Personal systems, on-line storefronts, libraries, and archives are examples of repositories.

3.27
Resource
entity, quality, event, state, concept, substance, or anything else referred to by a noun and identified by an `r:Resource`

NOTE 1 See 4.3 for the meaning of `r:Resource`.

NOTE 2 ISO/IEC 21000-2 defines *resource* in a narrower sense than Resource is defined here. The relation might be characterized by the expression "a *resource* is a Resource that is also an asset."

3.28
resource attribute set definition element
element that is one of those defined in 9.3

3.29
Resource Member
third member

NOTE This term is used with respect to an authorization request, as in "the Resource Member of the authorization request."

3.30
revocable
ordered pair as defined in 6.7

3.31
Right
act identified by an `r:Right`

NOTE See 4.3 for the meaning of `r:Right`.

3.32
Right Member
second member

NOTE This term is used with respect to an authorization request, as in "the Right Member of the authorization request."

3.33
Satisfied
satisfied as mentioned in 7.4.5

3.34
Standard Extension Namespace
Namespace for the names defined in Clause 8

3.35
Surpasses
surpasses as defined in 6.8.1

3.36
service
system entity that provides functionality

NOTE 1 There is no requirement that a service be located on a physically different machine than a client.

NOTE 2 There is no requirement that a service be part of a different executable than a client.

3.37
system entity
entity that is capable of acting in a system

EXAMPLE An automated process, a subsystem, and a person or group of persons are examples of system entities.

3.38
Trust Root Member
seventh member

NOTE This term is used with respect to an authorization request, as in "the Trust Root Member of the authorization request."

3.39
Unconditionally
in a manner not subject to any Condition

3.40
URI
Uniform Resource Identifier

[RFC 2396]

3.41
Variable
variable as defined in 6.5.1.1

3.42
WSDL
Web Services Description Language

[WSDLSPEC]

3.43
XML
Extensible Markup Language

[XMLSPEC]

3.44
XPath
XML Path Language

[XPATH]

3.45
I
the `r:issue` element

3.46
P
the `r:possessProperty` element

4 Namespaces and conventions

4.1 Namespaces

The Core Namespace shall be `urn:mpeg:mpeg21:2003:01-REL-R-NS`. The Standard Extension Namespace shall be `urn:mpeg:mpeg21:2003:01-REL-SX-NS`. The Multimedia Extension Namespace shall be `urn:mpeg:mpeg21:2003:01-REL-MX-NS`. In each of these, the `01` represents a serial number that is expected to change as this part of ISO/IEC 21000 evolves.

4.2 Namespace conventions

Throughout this part of ISO/IEC 21000, Qualified Names are written with a namespace prefix followed by a colon followed by the local part of the Qualified Name as shown in the following example.

EXAMPLE `r:grant`

For clarity, throughout this part of ISO/IEC 21000, consistent namespace prefixes are used. Table 1 gives these prefixes and the corresponding namespace. Licenses compliant with this part of ISO/IEC 21000 are not limited to the namespace prefixes in Table 1 for the corresponding namespace and may use any appropriate namespace prefix that is declared in the License.

Table 1 — Namespace prefixes

Namespace prefix	Namespace
r	urn:mpeg:mpeg21:2003:01-REL-R-NS
sx	urn:mpeg:mpeg21:2003:01-REL-SX-NS
mx	urn:mpeg:mpeg21:2003:01-REL-MX-NS
dsig	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
didl	urn:mpeg:mpeg21:2002:01-DIDL-NS
dii	urn:mpeg:mpeg21:2002:01-DII-NS
wsdl	http://schemas.xmlsoap.org/wsdl/
soap	http://schemas.xmlsoap.org/wsdl/soap/
acme	This namespace prefix is used for demonstration only.

4.3 Special typographical conventions

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

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

Specific linguistic constructions are used in this part of ISO/IEC 21000 with respect to these conventions in order to more concisely convey the intention. They are as follows:

— A passage that mentions an element, as in "the `r:grant`," is using the word in a technical sense to refer to the notion of that element as an XML Schema element.

— A passage that mentions a type and prefixes the type with the word *type*, as in "the type `r:Grant`," is using the word in a technical sense to refer to the notion of that type as an XML Schema type.

— A passage that mentions a type and prefixes the type with an article, such as *a* or *the*, as in "an `r:Grant`," is using the word in a technical sense to refer to any element whose type is that type or any derivation thereof (or, in the case of a simple type, any simple content whose type is that simple type or any derivation thereof). Semantics assigned to a type in this way shall not be overridden by type derivations or elements using the type; type derivations or elements that use the type may alter the semantics only as long as all the statements made about the type in these passages still hold for the type derivations and elements that use the type.

4.4 XML structural notation conventions

To refer to child elements of an element represented by a variable, the variable name is followed by a forward slash and the name of the particle against which the child elements validate as shown in Example 1.

EXAMPLE 1 Suppose g is the following `r:grant`:

```
<r:grant>
   <r:keyHolder>...</r:keyHolder>
   <acme:extensionRight/>
   <r:keyHolder>...</r:keyHolder>
</r:grant>
```

In this case, g/`r:principal` would refer to the first `r:keyHolder` child because it validates against the `r:principal` particle, which comes before the `r:right` particle in an `r:grant`. The second `r:keyHolder` validates against the `r:resource` particle, which comes after the `r:right` in an `r:grant`.

NOTE For more details about the syntax of `r:grant`, see Annex A.

To refer to child elements of child elements of an element represented by a variable, another forward slash is used as shown in Example 2.

EXAMPLE 2 c/`sx:location`/`sx:country`

To refer to attributes of an element represented by a variable, the variable name is followed by a forward slash, an at sign (@), and then the name of the attribute as shown in Example 3.

EXAMPLE 3 l/@`r:licenseId`

To avoid confusion with XPath notation, when XPath notation is used in this part of ISO/IEC 21000, its use is clearly indicated with wording such as "the following XPath location path."

4.5 Authorization context conventions

To refer to the value of a property of an authorization context variable, the variable name is followed by a full stop followed by the property name as shown in Example 1 and Example 2.

EXAMPLE 1 Σ.acme:a1() refers to the value of the acme:a1 property of the authorization context Σ.

EXAMPLE 2 If the value of x is 123, then Σ.acme:a2(x) refers to the value of the acme:a2(123) property of the authorization context Σ.

NOTE For more details about authorization contexts, see 5.3.

5 Authorization model

5.1 General

This Clause specifies the authorization model.

NOTE The authorization model makes use of an authorization request, an authorization context, an authorization story, and an authorizer. These are used to create authorization proofs which help to define the semantics of Licenses with respect to authorization requests. Figure 1 illustrates the relationship between the components of the authorization model.

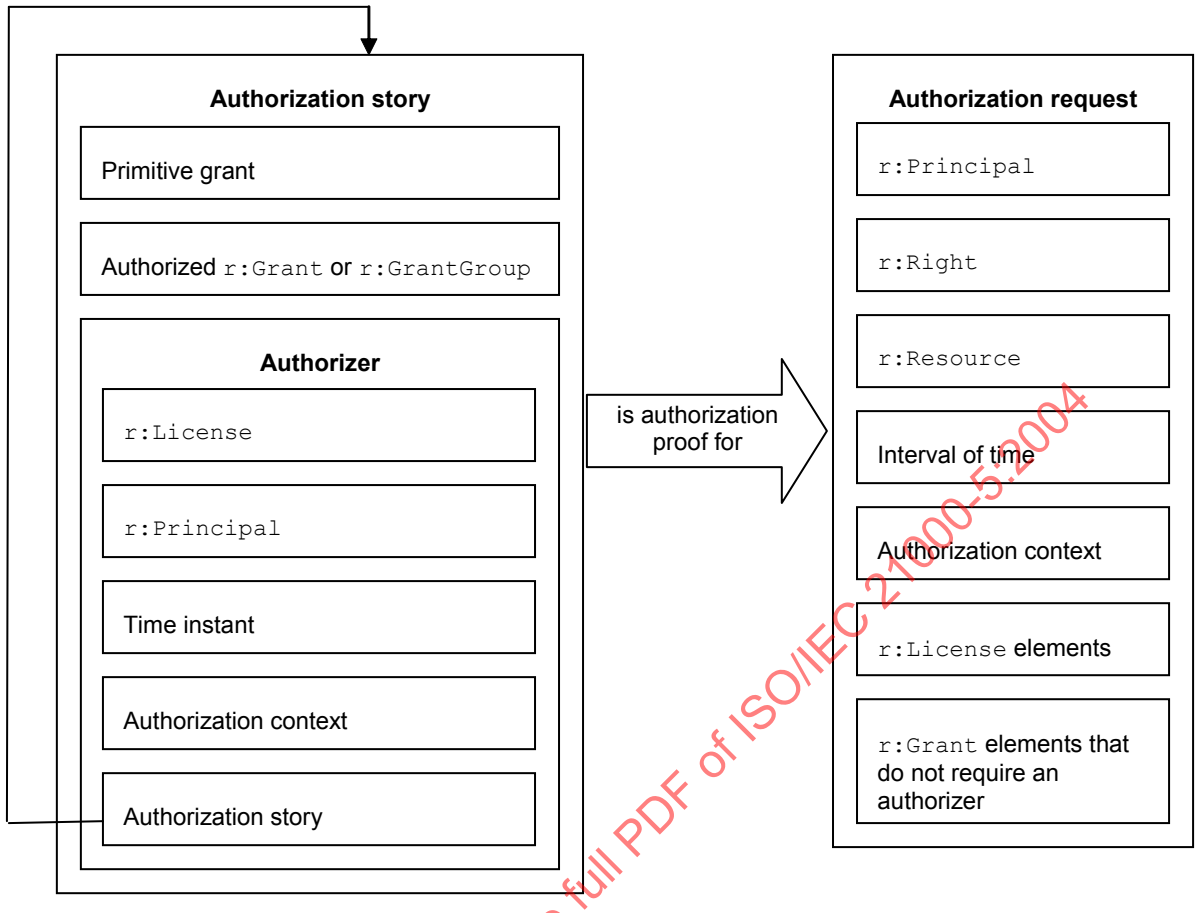

Figure 1 — Authorization Model

5.2 Authorization request

An authorization request is an ordered seven-tuple containing the following members, in order:

a) optionally, the `r:Principal` identifying the Principal for which permission is requested,

b) the `r:Right` identifying the Right the performance of which is requested to be permitted,

c) optionally, the `r:Resource` identifying the Resource upon which permission is requested,

d) the interval of time in which the requested performance of b) by a) upon c) is to occur,

e) the authorization context containing properties representing statements that are to be considered true for the purposes of establishing the requested permission,

f) the set of `r:License` elements that may be consulted to establish the requested permission, and

g) the set of `r:Grant` elements that do not require an authorizer for the purposes of establishing the requested permission.

NOTE 1 An authorization request can be conceptualized as representing the question "is it permitted for a given Principal to perform a given Right upon a given Resource during a given time interval based on a given authorization context, a given set of Licenses, and a given trust root?"

Each of the `r:License` elements *l* in the set mentioned in f) shall have exactly one *l*/`r:issuer` child.

NOTE 2 The semantics of an `r:License` *l* with a different number of *l*/`r:issuer` children is specified in terms of `r:License` elements with one such child each (see 6.4 for the specifics).

5.3 Authorization context

An authorization context is a set of properties having the following characteristics:

— each property consists of exactly one name and one value,

— a property name consists of exactly one Qualified Name and zero or more parameters,

— no two properties in an authorization context share the same property name, and

— each property represents a statement.

NOTE 5.6, 7.2, 8.2, and 9.2 define authorization context properties and the statements they represent.

5.4 Authorization story

An authorization story is an ordered triple containing the following members, in order:

a) a primitive grant,

b) either an `r:Grant` or an `r:GrantGroup`, and

c) optionally, an authorizer.

NOTE Conceptually, the first member of an authorization story is used to demonstrate to which authorization requests the authorization story applies. The second member of an authorization story represents the actual `r:Grant` or `r:GrantGroup` that is authorized by the third member of the authorization story.

5.5 Authorizer

An authorizer is an ordered five-tuple containing the following members, in order:

a) an r:License,

b) an r:Principal,

c) a time instant,

d) an authorization context, and

e) an authorization story.

NOTE Conceptually, if an `r:Grant` or `r:GrantGroup` is authorized, it is part of a License and is authorized in that License by some Principal at some time instant based on some authorization context and there is an authorization story explaining why that Principal was permitted to make that authorization.

5.6 Authorization model authorization context properties

Table 2 specifies the authorization context properties relating specifically to the authorization model and the statements they represent. If a property has the name given in the first column of Table 2 and the value given in the second column of Table 2, then the statement represented by that property is the statement given in the third column of Table 2.

Table 2 — Authorization model authorization context properties

Property name	Property value	Statement represented
r:issueTime(l, p)	i	l is an `r:License`, p is an `r:Principal`, i is a time instant, and p issued l at i. NOTE 1 The `r:timeOfIssue` field in an `r:IssuerDetails` (see 7.3.1.6.3) can be useful in determining when p issued l. However, it is wise to give consideration as to whether the issuer is trustworthy and, when in doubt, to seek additional proof, such as in the form of signatures and countersignatures. NOTE 2 The `r:Issuer` can be useful in determining whether p issued l. However, it is wise to give consideration as to whether the information given in the `r:Issuer` is trustworthy and, when in doubt, to seek additional proof, such as in the form of signatures and countersignatures.
r:issueContext(l, p, h, Σ)	true	l is an `r:License`, p is an `r:Principal`, h is either an `r:Grant` or an `r:GrantGroup`, Σ is an authorization context, and the statements represented by the properties in Σ are all true for the purposes of establishing the permission for the Principal identified by p to include an `r:Grant` or `r:GrantGroup` that is Equal to h as l/`r:grant` or l/`r:grantGroup` when issuing l.

5.7 Authorization proof

A finite authorization story (g, h, e) is said to be an authorization proof for an authorization request (p, r, t, v, Σ, L, R) if and only if all of the following are true:

— either p Surpasses g/`r:principal` or g/`r:principal` is absent,

— g/`r:right` is Equal to r,

— either g/`r:resource` is Equal to t or both are absent,

— either g/`r:condition` is Satisfied with respect to authorization request (p, r, t, v, Σ, L, R) and authorization story (g, h, e) or g/`r:condition` is absent,

— g is Derived from h with respect to authorization request (p, r, t, v, Σ, L, R) and authorizer e,

— if e is absent, then h is a member of R, and

— if e is present, then, letting (l, π, i, σ, s) be the ordered five-tuple representation of e, all of the following are true:

  — l is a member of L,

  — h is Equal to one of the l/`r:Grant` or l/`r:GrantGroup` children of l,

  — Σ.r:issueTime(l, π) is i,

  — Σ.r:issueContext(l, π, h, σ) is true,

  — the time instant i is no later in time than the start of time interval v, and

  — s is an authorization proof for the authorization request (π, I, h, ψ, σ, L, R) where ψ is a time interval of zero length starting at i.

5.8 License Semantics

If there is an authorization proof for an authorization request (p, r, t, v, Σ, L, R), then the semantics of the set of `r:License` elements L with respect to that authorization request is that they permit the optional Principal identified by p to perform the Right identified by r over the optional Resource identified by t during the time interval v based on the authorization context Σ and the set R of `r:Grant` elements that do not require an authorizer.

6 Architectural concepts

6.1 Equality

Let a and b be XML elements. Let α and β be the result of the execution of the Schema Centric Canonicalization algorithm (see SCC14N) on an XML Information Set (see XMLINFOSET) whose document information item contains in its [children] property the item a or b, respectively. Then a is Equal to b if and only if the bits strings α and β are bit-for-bit identical.

NOTE Annex C presents an approach to efficiently testing for equality.

6.2 License syntax

Licenses shall be valid `r:License` elements.

6.3 License parts

The following provisions apply to `r:License` elements that contain `r:LicensePart` elements having the `r:licensePartId` or `r:licensePartIdRef` attributes.

a) An `r:LicensePart` shall not have both the `r:licensePartId` attribute and the `r:licensePartIdRef` attribute.

b) For a given `r:LicensePartId` value v and `r:License` l, l shall not contain more than one `r:LicensePart` having an `r:licensePartId` attribute with the value v.

c) If an `r:LicensePart` has an `r:licensePartIdRef` attribute, then it shall have empty content. As a corollary, types that are derivations of the type `r:LicensePart` should allow their content to be empty (for otherwise their `r:licensePartIdRef` attribute shall not be used).

d) If an `r:LicensePart` a has an `r:licensePartIdRef` attribute with a certain value v, then there shall exist some (other) `r:LicensePart` b in the same `r:License` as a such that all of the following are true:

   1) b has an `r:licensePartId` attribute with value v,

   2) the expanded element name of b exactly matches that of a, and

   3) b is not an ancestor of a.

e) With the exception of signature verification and License issuer resolution (see 6.4), the following two processing steps shall be carried out before the other License processing steps defined in this part of ISO/IEC 21000. In particular, they shall be carried out before the evaluation of Variable references or the testing of equality.

   1) If an `r:LicensePart` a has an `r:licensePartIdRef` attribute with a certain value v, and b is the `r:LicensePart` in the same `r:License` as a that has an `r:licensePartId` attribute with value v, then the semantics of the `r:License` containing a and b are as if

      i) a were removed from the `r:License` and replaced with an element α that is Equal to b,

ii) the `r:licensePartId` attributes were removed from α and all of its descendants, and

iii) any preserved attributes that α has were removed from α, and any preserved attributes that a has were added to α, where here a preserved attribute is

I) any attribute of type `xsd:ID` or

II) any attribute for which `id` is the local part of its Qualified Name.

2) If an `r:License` contains no `r:LicensePart` elements with an `r:licensePartIdRef` attribute, then the semantics of that `r:License` are as if the `r:licensePartId` attribute were removed from all `r:LicensePart` elements in that `r:License` with an `r:licensePartId` attribute.

NOTE 1 It is the intent of e)1)iii)II) to enable the useful definition and incorporation of other identification systems on `r:LicensePart` elements beyond the document-global `xsd:ID`-typed identifiers.

NOTE 2 Circular references such as that shown in Example 1 are not allowed because, after one application of provision e), the semantics will be as shown in Example 2, which violates provision d)3).

EXAMPLE 1

```
<r:license>
   <r:grantGroup licensePartId="x">
      <r:grantGroup licensePartIdRef="y"/>
   </r:grantGroup>
   <r:grantGroup licensePartId="y">
      <r:grantGroup licensePartIdRef="x"/>
   </r:grantGroup>
</r:license>
```

EXAMPLE 2

```
<r:license>
   <r:grantGroup licensePartId="x">
      <r:grantGroup>
         <r:grantGroup licensePartIdRef="x"/>
      </r:grantGroup>
   </r:grantGroup>
   <r:grantGroup licensePartId="y">
      <r:grantGroup>
         <r:grantGroup licensePartIdRef="y"/>
      </r:grantGroup>
   </r:grantGroup>
</r:license>
```

6.4 License issuers

Let l be an `r:License`. Let k be the number of l/`r:issuer` children of l. If k is not 1, then the semantics of l is the same as the semantics of k independent `r:License` elements, one for each of the k l/`r:issuer` children of l. Each of these k `r:License` elements λ is Equal to l, except that λ has only one λ/`r:issuer` child and that child is Equal to the respective l/`r:issuer` child.

With the exception of signature verification, this processing step shall be carried out before the other License processing steps defined in this part of ISO/IEC 21000. In particular, it shall be carried out before `r:licensePartId` references, Variable references, and the testing of equality.

NOTE If k is 0, it follows that the semantics of l is the same as the semantics of no `r:License` at all. In colloquial terms, if a License has no issuer, it has not been issued and so has no effect.

6.5 Variables

6.5.1 Variable definition

6.5.1.1 General

A Variable is declared using an `r:forAll` element. A Variable has a name, a set of bindings, a scope, and a set of eligible bindings.

6.5.1.2 Variable name

Let f be an `r:forAll` element. Let y be the Variable declared by f. Then the name of y is the value of f/@`r:varName`.

6.5.1.3 Variable bindings

Let X be the universe of XML elements. Let q be an authorization request. Let e be an authorizer. Let f be an `r:forAll` element. Let y be the Variable declared by f. Then the set of bindings of y with respect to q and e is defined to be that subset of X such that x in X is in the set of bindings of y with respect to q and e if and only if a new XML document containing the element x as the root element Matches each of the f/`r:anXmlPatternAbstract` children of f with respect to q and e.

6.5.1.4 Variable scope

For any XML element x, let following(x) be that set of XPath nodes selected by the XPath location path

```
following-sibling::*/descendent-or-self::node()
```

when evaluated with x as the contextual XPath node. For an `r:forAll` element x, let redeclarations(x) be that set of XPath nodes selected by the XPath location path

```
following-sibling::*/descendent-or-self::r:forAll[@r:varName=$fVarName]
```

when evaluated with x as the contextual XPath node and `$fVarName` as the value of x/@`r:varName`. Let f be an `r:forAll` element. Let y be the Variable declared by f. Then the scope of y is defined to be following(f) less the union of all following(d) where each XPath node d is in redeclarations(f).

6.5.1.5 Eligible Variable bindings

Let X be the universe of XML elements. Let q be an authorization request. Let e be an authorizer. Let f be an `r:forAll` element. Let y be the Variable declared by f. Then the set of eligible bindings of y with respect to q and e is defined to be that subset of X such that x in X is in the set of eligible bindings of y with respect to q and e if and only if x is in the set of bindings of y with respect to q and e and for all elements χ in the scope of y where χ/@`r:varRef` is Equal to f/@`r:varName` all of the following hold:

— either the expanded element name of x exactly matches that of χ or x is substitutable for χ using substitution groups,

— either the type of x exactly matches that of χ or the type of x derives, through any number of levels, from the type of χ using type derivation, and

— if χ is removed from its document and replaced with an element Equal to x, that document is still valid.

6.5.2 Variable referencing

Let a be an `r:LicensePart`. If a/@`r:varRef` exists, then a shall have empty content and the value of a/@`r:varRef` shall be the name of some Variable whose scope includes a.

NOTE Conceptually, the contents of a are determined by the bindings of the Variable it references, so a doesn't need to have its own content.

6.6 Conceptually abstract elements and types

If a conceptually abstract element appears in an `r:License`, it shall either have a type that is not conceptually abstract or appear in the form of a Variable reference, as described in 6.5.2.

If a conceptually abstract type appears in an `r:License`, it shall belong to an element that either is not conceptually abstract or appears in the form of a Variable reference, as described in 6.5.2.

6.7 Revocable

A revocable is an ordered pair containing the following members, in order:

a) a document and

b) the `r:Principal` identifying the issuer of that document.

6.8 Principal surpassing

6.8.1 General

Let a and b be `r:Principal` elements. Let A be the set representation of a as defined in 6.8.2. Let B be the set representation of b as defined in 6.8.2. Then b Surpasses a if and only if for every α in A there exists a β in B such that α is Equal to β.

6.8.2 Principal set representation

Let p be an `r:Principal`. If p is an `r:AllPrincipals` element, then the set representation of p is the union of the set representations of each p/`r:principal` child of p. If p is not an `r:AllPrincipals` element, then the set representation of p is the one-member set containing only p.

6.9 Derived grants and grant groups

6.9.1 General

Let a be an `r:Grant` or an `r:GrantGroup`. Let b be an `r:Grant` or an `r:GrantGroup`. Let q be an authorization request. Let e be an authorizer. Then a is Derived from b with respect to q and e if and only if either of the following are true:

— a is one-step-derived from b, or

— there exists some `r:Grant` or `r:GrantGroup` ϕ such that all of the following are true:

  — a is one-step-derived from ϕ with respect to q and e as defined in 6.9.2.1, and

  — ϕ is Derived from b with respect to q and e.

6.9.2 One-step-derived grants and grant groups

6.9.2.1 General

Let a be an `r:Grant` or an `r:GrantGroup`. Let b be an `r:Grant` or an `r:GrantGroup`. Let q be an authorization request. Let e be an authorizer. Then a is one-step-derived from b with respect to q and e if and only if at least one of the following is true:

— a is Equal to b,

— a is one-step-derived from b with respect to q and e as defined in 6.9.2.2,

— a is one-step-derived from b with respect to q and e as defined in 6.9.2.3, or

— a is one-step-derived from b with respect to q and e as defined in 6.9.2.4.

6.9.2.2 One-step-derived grants and grant groups resulting from a Variable

Let a be an `r:Grant` or an `r:GrantGroup`. Let b be an `r:Grant` or an `r:GrantGroup`. Let q be an authorization request. Let e be an authorizer. Then a is one-step-derived from b with respect to q and e if and only if both b/`r:forAll` is not absent and, letting f be the first b/`r:forAll` child of b, a is Equal to b except that

— f is not present in a and

— there exists an x in the eligible bindings of the Variable declared by f with respect to q and e such that, throughout the scope of that Variable in b, all elements having references to that Variable are replaced in a by x.

6.9.2.3 One-step-derived grants and grant groups resulting from a grant group

Let a be an `r:Grant` or an `r:GrantGroup`. Let b be an `r:Grant` or an `r:GrantGroup`. Let q be an authorization request. Let e be an authorizer. Then a is one-step-derived from b with respect to q and e if and only if both b is an `r:GrantGroup` that has no child `r:forAll` element and there exists a b/`r:grant` or b/`r:grantGroup` β such that a is Equal to β except that

— a/`r:principal` is Equal to an `r:allPrincipals` whose children are, in order, b/`r:principal` (if present) and β/`r:principal` (if present) and

— a/`r:condition` is Equal to an `r:allConditions` whose children are, in order, b/`r:condition` (if present) and β/`r:condition` (if present).

6.9.2.4 One-step-derived grants resulting from a delegation control

Let a be an `r:Grant` or an `r:GrantGroup`. Let b be an `r:Grant` or an `r:GrantGroup`. Let q be an authorization request. Let e be an authorizer. Then a is one-step-derived from b with respect to q and e if and only if all of the following are true:

— b/`r:delegationControl` is present,

— for each b/`r:forAll` f, b/`r:delegationControl`, b/`r:principal`, and all of their descendent elements that are within the scope of the Variable declared by f do not have references (see 6.5.2) to the Variable declared by f,

— a has no child `r:forAll` elements,

— a/`r:delegationControl` is absent,

— a/`r:principal` is Equal to b/`r:principal`,

— a/`r:right` is Equal to I,

— letting α be a/`r:resource`, α is Equal to b except that

  — α/`r:delegationControl` is an allowable destination delegation control of b/`r:delegationControl` with respect to q, e, and b as defined in 7.3.4.3,

— a/r:principal is an allowable destination principal of b/r:delegationControl with respect to q, e, and b as defined in 7.3.4.1, and

— a/r:condition is an allowable destination condition of b/r:delegationControl with respect to q, e, and b as defined in 7.3.4.2, and

— a/r:condition is absent.

NOTE By placing an r:delegationControl in an r:Grant or r:GrantGroup, the effect of delegation of that r:Grant or r:GrantGroup can be achieved. If Principal A issues a License to Principal B and enables delegation using an r:delegationControl, Principal B can then issue a similar License to Principal C who can then issue a similar License to Principal D, who might be the same as Principal B. If the Conditions in the License from Principal A to Principal B expire (they are no longer Satisfied), the Licenses to Principal C and Principal D are still valid. If this is not the intent of Principal A, Principal A can use appropriate Conditions and place appropriate r:DcConstraint elements in the r:delegationControl so that the Conditions applicable to Principal C and Principal D are at least as restrictive as (meaning they will expire no later than) those applicable to Principal B.

6.10 Encrypted Elements

In cases where an r:License, r:Grant, or r:GrantGroup has a child validating against the r:encryptedLicense, r:encryptedGrant, or r:encryptedGrantGroup particle, respectively, that child shall represent (see 7.3.6) the encryption of the contents of the r:License, r:Grant, or r:GrantGroup, and this part of ISO/IEC 21000 excepting 6.10 shall apply to the clear-text form of the r:License, r:Grant, or r:GrantGroup.

7 Core

7.1 General

This Clause specifies the syntax and semantics integral to the architecture.

7.2 Core authorization context properties

Table 3 specifies the authorization context properties relating to Clause 7 and the statements they represent. If a property has the name given in the first column of Table 3 and the value given in the second column of Table 3, then the statement represented by that property is the statement given in the third column of Table 3.

Table 3 — Core authorization context properties

Property name	Property value	Statement represented
r:exerciseMechanism()	m	m is an r:ExerciseMechanism, and the mechanism indicated by m is used to effect the requested performance.
r:freshness(m, b, τ)	true	m is an r:RevocationMechanism, b is a revocable, τ is a time instant, and the mechanism indicated by m for effecting revocation of a revocable guarantees that no revocation of b has been, is, or will be in effect before or at τ.
r:fulfiller()	p	p is an r:Principal, and p identifies the fulfiller (according to the semantics of the Right Member of the authorization request) for the requested performance.

7.3 General core elements and types

7.3.1 License

7.3.1.1 General

Additional normative semantics of `r:License` is given in 5.8, 6.3, 6.4, 6.6, and 6.10.

NOTE Example Licenses can be found in Annex D and G.5.

7.3.1.2 Title

For an `r:License` *l*, each of the *l*/`r:title` elements that are present shall provide a descriptive phrase about *l* that is intended for human consumption.

EXAMPLE The content of an `r:title` element is displayed in a user interface.

Automated processors shall not interpret semantically the contents of such *l*/`r:title` elements.

7.3.1.3 Inventory

An `r:Inventory` is a syntactic container of `r:LicensePart` elements. Automated processors shall not interpret semantically the contents of an `r:Inventory` element.

NOTE An `r:Inventory` element of an `r:License` provides a convenient place to define `r:LicensePart` elements without the definition site associating any particular semantics with the `r:LicensePart` elements. Usefully and usually, `r:LicensePart` elements in an `r:Inventory` will have `r:licensePartId` attributes so that they can be referenced elsewhere in the `r:License`.

7.3.1.4 OtherInfo

For an `r:License` *l*, the *l*/`r:otherInfo` element, if present, shall provide additional information. Automated processors need not interpret semantically the contents of an *l*/`r:otherInfo` element.

EXAMPLE A License creator might include in the `r:otherInfo` element some information that is peripherally related to some authentication or authorization process.

7.3.1.5 License attributes

For an `r:License` *l*, the *l*/@`r:licenseId` attribute, if present, shall provide the URI that identifies *l*. Automated processors need not interpret semantically the contents of any `r:License` attributes.

7.3.1.6 Issuer

7.3.1.6.1 In a License

Let *l* be an `r:License` with one *l*/`r:issuer` *u* (see 6.4 for additional semantics of `r:issuer` in Licenses). *u* shall give information about the issuer of *l*.

7.3.1.6.2 In general

Let *u* be an `r:Issuer`. *u*/`dsig:Signature`, if present, shall be a signature (XMLDSIG) that may be used to establish which Principal the issuer is.

NOTE If a License has been tampered with so that the digest value in the signature does not match the digest of the License, then the signature probably does not establish which Principal the issuer is, and therefore the tampered License does not follow the semantics of this subclause. In this way, signatures also serve as a way to ensure integrity and authenticity of Licenses.

u/r:principal, if present, shall identify which Principal the issuer is. *u*/r:details, if present, shall give details about the circumstances under which the issuer issues.

7.3.1.6.3 IssuerDetails

Let ε be an r:IssuerDetails. ε/r:timeOfIssue, if present, shall indicate the specific date and time at which the issuer claims to have issued. Each ε/r:revocationMechanism, if any are present, shall indicate a mechanism by which the issuer shall, if he later wishes to revoke his revocable for the document whose issuer information is given by the parent of ε, effect his revocation of that revocable.

EXAMPLE
```
<r:details>
   <r:timeOfIssue>2003-07-01T00:00:00</r:timeOfIssue>
   <r:revocationMechanism>
      <r:revocationService>
         <r:serviceReference licensePartIdRef="uddiService"/>
      </r:revocationService>
   </r:revocationMechanism>
</r:details>
```

7.3.1.6.4 RevocationMechanism

Let *m* be an r:RevocationMechanism. The child of *m* shall indicate a mechanism for effecting revocation of a revocable. *m*/r:revocationService, if present, indicates that the service reference identified by *m*/r:revocationService/r:serviceReference is the mechanism for effecting revocation of a revocable.

7.3.1.6.5 Signature recommendations

When a dsig:Signature *s* is used in an r:Issuer in an r:License, there should be an *s*/dsig:SignedInfo/dsig:Reference element *r* such that

— *r*/@dsig:URI is omitted and

— *r*/dsig:Transforms is present and contains exactly two child dsig:Transform elements, both dsig:Transform children are empty, the dsig:Algorithm attribute of the first dsig:Transform child has the value urn:mpeg:mpeg21:2003:01-REL-R-NS:licenseTransform, and the dsig:Algorithm attribute of the second dsig:Transform child has the value urn:uddi-org:schemaCentricC14N:2002-07-10.

NOTE 1 This practice simplifies the process of determining exactly which pieces of an r:License have actually been signed. It also ensures that signatures are consistent with the Equal operation.

NOTE 2 There can also be other *s*/dsig:SignedInfo/dsig:Reference elements that reference items (such as metadata) external to the License. Such references would be included in the signature. The use of such references is beyond the scope of this specification.

EXAMPLE
```
<dsig:Signature>
   <dsig:SignedInfo>
      <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
      <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference>
         <dsig:Transforms>
            <dsig:Transform Algorithm="urn:mpeg:mpeg21:2003:01-REL-R-
NS:licenseTransform"/>
            <dsig:Transform Algorithm="urn:uddi-org:schemaCentricC14N:2002-07-10"/>
         </dsig:Transforms>
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>Jk9QbKOQCo941tTExbj1/Q==</dsig:DigestValue>
```

```
      </dsig:Reference>
   </dsig:SignedInfo>
   <dsig:SignatureValue>DFgqOhh5QQ==</dsig:SignatureValue>
   <dsig:KeyInfo>
      <dsig:KeyValue>
         <dsig:RSAKeyValue>
            <dsig:Modulus>gOyM4ccyzA==</dsig:Modulus>
            <dsig:Exponent>AQABAA==</dsig:Exponent>
         </dsig:RSAKeyValue>
      </dsig:KeyValue>
   </dsig:KeyInfo>
</dsig:Signature>
```

7.3.1.6.6 License transform

Let t be a `dsig:Transform` element with a `dsig:Algorithm` attribute whose value is `urn:mpeg:mpeg21:2003:01-REL-R-NS:licenseTransform`. Let l be the most immediate `r:License` ancestor of t. Then t emits as output l with all its l/`r:issuer` children wholly removed except for that l/`r:issuer` that contains t, which is kept, removing its `dsig:Signature` child instead.

7.3.1.7 Recommendation on declaration of Namespaces used in a License

An `r:License` should not rely on Namespace declarations to be imported from its context in its surrounding XML document.

NOTE Following this recommendation greatly facilitates the ability to manipulate an `r:License` as a self-contained unit within Applications. It also helps ensure that namespace declarations are included in signatures.

7.3.2 LicenseGroup

An `r:LicenseGroup` is a syntactic container of `r:License` elements. There is no normative semantics for `r:LicenseGroup`. No semantics is conveyed by the presence of two `r:License` elements within the same `r:LicenseGroup`.

NOTE This type exists merely due to the observation that it is often convenient to be able to use it as a container in XML documents. No use of it is made in this part of ISO/IEC 21000.

7.3.3 ForAll

The semantics of `r:forAll` is given in 6.5.

7.3.4 DelegationControl

7.3.4.1 Allowable destination principals

Let p be an `r:Principal`, d be an `r:delegationControl`, q be an authorization request, e be an authorizer, and g be an `r:Grant` or `r:GrantGroup`. Then p is an allowable destination principal of d with respect to q, e, and g if and only if p is an Allowable Destination Principal of each d/`r:dcConstraint` child of d with respect to q, e, and g.

NOTE Because an Allowable Destination Principal does not appear in a License, it and its descendents cannot have any references to Variables not declared within the Allowable Destination Principal itself.

7.3.4.2 Allowable destination conditions

Let c be an `r:Condition`, d be an `r:delegationControl`, q be an authorization request, e be an authorizer, and g be an `r:Grant` or `r:GrantGroup`. Then c is an allowable destination condition of d with respect to q, e, and g if and only if c is an Allowable Destination Condition of each d/`r:dcConstraint` child of d with respect to q, e, and g.

NOTE Because an Allowable Destination Condition does not appear in a License, it and its descendents cannot have any references to Variables not declared within the Allowable Destination Condition itself.

7.3.4.3 Allowable destination delegation controls

Let δ be an `r:delegationControl`, d be an `r:delegationControl`, q be an authorization request, e be an authorizer, and g be an `r:Grant` or `r:GrantGroup`. Then δ is an allowable destination delegation control of d with respect to q, e, and g if and only if δ is an Allowable Destination Delegation Control of each d/`r:dcConstraint` child of d with respect to q, e, and g.

NOTE Because an Allowable Destination Delegation Control does not appear in a License, it and its descendents cannot have any references to Variables not declared within the Allowable Destination Delegation Control itself.

7.3.5 NonSecureReference

The semantics and processing associated with the type `r:NonSecureReference` are identical to those of `dsig:ReferenceType`, except that the semantics and processing associated with the `dsig:DigestMethod` and `dsig:DigestValue` elements found in `dsig:ReferenceType` are omitted.

7.3.6 EncryptedContent

The type `r:EncryptedContent` modifies the semantics of its base type, `xenc:EncryptedDataType`. An `r:EncryptedContent` shall have an `xenc:Type` attribute with a value of `http://www.w3.org/2001/04/xmlenc#Content`. (See XMLENC.)

NOTE This value is the type associated with encrypting XML element content.

The plaintext of an `r:EncryptedContent` element shall replace semantically the `r:EncryptedContent` element and thus become the content of said element's parent. In doing so, the plaintext shall conform to the schema of the parent as a whole.

7.4 Core conceptually abstract elements and types

7.4.1 LicensePart

The element `r:licensePart` is conceptually abstract. The type `r:LicensePart` is conceptually abstract. Because of the requirements given in 6.3 c) and 6.5.2, if the syntax of a particular derivation of the type `r:LicensePart` declares its content model as optional, the semantics of that optional content model, unless specified otherwise, is that it shall not be omitted in an `r:LicensePart` of that derivation unless that `r:LicensePart` has an `r:licensePartIdRef` or `r:varRef` attribute.

7.4.2 Principal

The element `r:principal` is conceptually abstract. The type `r:Principal` is conceptually abstract. An `r:Principal` shall identify a system entity.

7.4.3 Right

The element `r:right` is conceptually abstract. The type `r:Right` is conceptually abstract. An `r:Right` shall identify an act. The semantic specification of each different particular kind of `r:Right` should indicate which kinds of `r:Resource`, if any, are to be used as the Resource Member of an authorization request having that kind of `r:Right` as the Right Member.

7.4.4 Resource

The element `r:resource` is conceptually abstract. The type `r:Resource` is conceptually abstract. An `r:Resource` shall identify an entity, quality, event, state, concept, substance, or anything else referred to by a noun.

7.4.5 Condition

The element `r:condition` is conceptually abstract. The type `r:Condition` is conceptually abstract. The semantics specification of each different particular kind of `r:Condition` shall indicate whether or not an `r:Condition` of that kind is Satisfied with respect to a given authorization request and authorization story. An application that encounters an `r:Condition` of which it lacks semantic knowledge shall not consider the `r:Condition` Satisfied.

7.4.6 AnXmlPatternAbstract

7.4.6.1 General

The element `r:anXmlPatternAbstract` is conceptually abstract. The type `r:AnXmlPatternAbstract` is conceptually abstract. The semantics specification of each different particular kind of `r:AnXmlPatternAbstract` shall indicate whether or not a given XML document Matches an `r:AnXmlPatternAbstract` of that kind with respect to a given authorization request and authorizer. An application that encounters an `r:AnXmlPatternAbstract` of which it lacks semantic knowledge shall not consider anything as Matching that `r:AnXmlPatternAbstract`.

7.4.6.2 PrincipalPatternAbstract

The element `r:principalPatternAbstract` is conceptually abstract. The type `r:PrincipalPatternAbstract` is conceptually abstract. An XML document shall not Match an `r:PrincipalPatternAbstract` with respect to any given authorization request and authorizer unless that XML document contains exactly one `r:Principal` as the root element.

7.4.6.3 RightPatternAbstract

The element `r:rightPatternAbstract` is conceptually abstract. The type `r:RightPatternAbstract` is conceptually abstract. An XML document shall not Match an `r:RightPatternAbstract` with respect to any given authorization request and authorizer unless that XML document contains exactly one `r:Right` as the root element.

7.4.6.4 ResourcePatternAbstract

The element `r:resourcePatternAbstract` is conceptually abstract. The type `r:ResourcePatternAbstract` is conceptually abstract. An XML document shall not Match an `r:ResourcePatternAbstract` with respect to any given authorization request and authorizer unless that XML document contains exactly one `r:Resource` as the root element.

7.4.6.5 ConditionPatternAbstract

The element `r:conditionPatternAbstract` is conceptually abstract. The type `r:ConditionPatternAbstract` is conceptually abstract. An XML document shall not Match an `r:ConditionPatternAbstract` with respect to any given authorization request and authorizer unless that XML document contains exactly one `r:Condition` as the root element.

7.4.7 DcConstraint

The element `r:dcConstraint` is conceptually abstract. The type `r:DcConstraint` is conceptually abstract.

The semantics specification of each different particular kind of `r:DcConstraint` shall indicate whether or not a given `r:Principal` is an Allowable Destination Principal of an `r:DcConstraint` of that kind with respect to a given authorization request, authorizer, and `r:Grant` or `r:GrantGroup`. An application that encounters an `r:DcConstraint` of which it lacks semantic knowledge shall not consider anything as an Allowable Destination Principal of that `r:DcConstraint`.

The semantics specification of each different particular kind of `r:DcConstraint` shall indicate whether or not a given `r:Condition` is an Allowable Destination Condition of an `r:DcConstraint` of that kind with respect to a given authorization request, authorizer, and `r:Grant` or `r:GrantGroup`. An application that encounters an `r:DcConstraint` of which it lacks semantic knowledge shall not consider anything as an Allowable Destination Condition of that `r:DcConstraint`.

The semantics specification of each different particular kind of `r:DcConstraint` shall indicate whether or not a given `r:delegationControl` is an Allowable Destination Delegation Control of an `r:DcConstraint` of that kind with respect to a given authorization request, authorizer, and `r:Grant` or `r:GrantGroup`. An application that encounters an `r:DcConstraint` of which it lacks semantic knowledge shall not consider anything as an Allowable Destination Delegation Control of that `r:DcConstraint`.

7.4.8 TrustRoot

The element `r:trustRoot` is conceptually abstract. The type `r:TrustRoot` is conceptually abstract. The semantics specification of each different particular kind of `r:TrustRoot` shall indicate the set of `r:Grant` elements identified by an `r:TrustRoot` of that kind with respect to a given authorization request and authorizer. An application that encounters an `r:TrustRoot` of which it lacks semantic knowledge shall consider that `r:TrustRoot` as representing the empty set.

7.4.9 ServiceDescription

The element `r:serviceDescription` is conceptually abstract. The type `r:ServiceDescription` is conceptually abstract. An `r:ServiceDescription` shall describe a service. An `r:ServiceDescription` should indicate the address of, the interface to, the rules for interacting with, and the functionality provided by the service. An `r:ServiceDescription` may include additional information about a service.

NOTE See 8.12 for some examples of concrete types derived from `r:ServiceDescription`.

7.4.10 PropertyAbstract

The element `r:propertyAbstract` is conceptually abstract. The type `r:PropertyAbstract` is conceptually abstract. An `r:PropertyAbstract` shall identify a property that can be possessed by a Principal.

7.5 Core principals

7.5.1 AllPrincipals

Let p be an `r:AllPrincipals`. Then p identifies that system entity that is identified by each of the p/`r:principal` children of p.

EXAMPLE
```
<r:allPrincipals>
   <r:keyHolder licensePartIdRef="Alice"/>
   <r:keyHolder licensePartIdRef="Alice2"/>
</r:allPrincipals>
```

NOTE If p has no children, then, according to 6.8, any `r:Principal` Surpasses p.

7.5.2 KeyHolder

Let p be an `r:KeyHolder`. Then p identifies that system entity that possesses the cryptographic key indicated by the p/`r:info` element as determined by the semantics of `dsig:KeyInfoType` given in XMLDSIG.

EXAMPLE 1 A Principal using public-key cryptography can be identified as that Principal that possesses the private key that corresponds to the public key given in an `r:KeyHolder`.

EXAMPLE 2
```
<r:keyHolder licensePartId="Alice">
   <r:info>
      <dsig:KeyValue>
         <dsig:RSAKeyValue>
            <dsig:Modulus>AliM4ccyzA==</dsig:Modulus>
            <dsig:Exponent>AQABAA==</dsig:Exponent>
         </dsig:RSAKeyValue>
      </dsig:KeyValue>
   </r:info>
</r:keyHolder>
```

7.6 Core rights

7.6.1 Issue

Let r be an `r:issue`. Then r identifies the act of including an `r:Grant` or `r:GrantGroup` as *//*`r:grant` or *//*`r:grantGroup` when issuing an `r:License` l.

If r is used as the Right Member of an authorization request, then the Resource Member of that authorization request shall be present and shall be an `r:Grant` or an `r:GrantGroup`.

NOTE Several authorization requests apply to issuing an `r:License` l, one for each *//*`r:grant` or *//*`r:grantGroup` therein.

7.6.2 Obtain

Let r be an `r:Obtain`. Then r identifies the act of obtaining an `r:Grant` or `r:GrantGroup` as *//*`r:grant` or *//*`r:grantGroup` in an issued `r:License` l.

If r is used as the Right Member of an authorization request, then the Resource Member of that authorization request shall be present and shall be an `r:Grant` or an `r:GrantGroup` and, letting Σ be the Authorization Context Member of that authorization request, Σ.r:fulfiller() shall identify the Principal who is requested to be the issuer of l.

NOTE 1 The use of an `r:Obtain` can be conceptualized as an offer or advertisement possibly for the sale of some `r:Grant` or `r:GrantGroup`. Suppose there is an authorization proof for an authorization request having an `r:Obtain` as the Right Member, an accurate Authorization Context Member Σ, and a Trust Root Member R. If R represents a sole trusted root issuer that is the same Principal identified by Σ.r:fulfiller(), it is generally expected that that Principal indeed will fulfill the request to obtain, since he is ultimately responsible for permitting the act of obtaining.

NOTE 2 If a Principal is permitted to obtain an `r:Grant` from another Principal, there is no implication that the other Principal is permitted to issue that `r:Grant`.

NOTE 3 If a Principal is permitted to obtain an `r:Grant`, there is no implication that the Principal is permitted to do the things mentioned in that `r:Grant` until he actually obtains it, at which time the normal License semantics apply to the License that has the obtained `r:Grant`.

EXAMPLE
```
<r:grant>
   <r:keyHolder licensePartIdRef="Alice"/>
   <r:obtain/>
   <r:grantGroup licensePartIdRef="playAndAdapt"/>
</r:grant>
```

7.6.3 PossessProperty

Let r be an `r:PossessProperty`. Then r identifies the act of claiming ownership of a property.

If *r* is used as the Right Member of an authorization request, then the Resource Member of that authorization request shall be present and shall be an `r:PropertyAbstract`.

NOTE An `r:PossessProperty` can be used to model the authorized bindings of names and other attributes to Principals as is done in X.509 (ISO/IEC 9594-8) and other public key certificates. An `r:PossessProperty` can also be used to model group membership and roles.

7.6.4 Revoke

Let *r* be an `r:Revoke`. Then *r* identifies the act of utilizing a mechanism to effect the revocation of a revocable.

If *r* is used as the Right Member of an authorization request, then the Resource Member of that authorization request shall be present and shall identify a revocable.

NOTE 1 Issuing a License having an `r:Grant` having an `r:Revoke` does not effect any revocation. Rather, one typically issues a License having an `r:Grant` having an `r:Revoke` in order to allow other Principals to effect the revocation of a revocable.

NOTE 2 Suppose one License allows a user to do something. Suppose a second License allows a second user to revoke a revocable for the first License. Typically the first user will not be involved with the second License. Rather, there will be a revocation mechanism indicated in the first license, and that revocation mechanism will rely on the second License to determine if (within the trust roots of that revocation mechanism) the second user's revoke is permitted. Then, the first user will rely on that revocation mechanism to determine whether the first License is still in force.

NOTE 3 It is expected that many revocation mechanisms will have trust roots installed that allow an issuer (and only the issuer) of a License to revoke the revocable for his License and freely delegate that ability to others.

NOTE 4 Because a revocation mechanism does not revoke but simply records the revocation, users do not check if the revocation mechanism is permitted to revoke.

EXAMPLE

```
<r:grant>
   <r:keyHolder licensePartIdRef="Alice"/>
   <r:revoke/>
   <r:revocable licensePartIdRef="licenseToBeRevoked"/>
</r:grant>
```

7.7 Core resources

7.7.1 DigitalResource

Let *t* be an `r:DigitalResource`. Then *t* identifies an arbitrary sequence of digital bits *b*, determined as follows:

— If *t*/`r:anXml` is present, then *b* is the character string that is the sequence of zero or more XML elements contained within *t*/`r:anXml`.

— If *t*/`r:binary` is present, then *b* is the bit sequence whose base64 encoding is the value of *t*/`r:binary`.

— If *t*/`r:secureIndirect` is present, then *b* is the bit sequence to which that element refers according to the semantics and processing associated with the type `dsig:ReferenceType` (XMLDSIG).

— If *t*/`r:nonSecureIndirect` is present, then *b* is the bit sequence to which that element refers according to the semantics and processing associated with the type `r:NonSecureReference`.

— Otherwise, *b* is the bit sequence identified by the child of *t*, according to the semantics of that child.

NOTE When using `r:nonSecureIndirect`, the user should be comfortable with the fact that different executions of the process for determining the sequence of bits might yield different results. No cryptographic measures are in place to ensure the yielded sequence of bits is the expected one.

EXAMPLE
```
<r:digitalResource licensePartId="video">
   <r:nonSecureIndirect URI="http://acme.org/myVideo"/>
</r:digitalResource>
```

7.7.2 Grant

Let *t* be an `r:Grant`. Then *t* identifies the Resource that is an XML element Equal to *t*.

NOTE Additional semantics related to `r:Grant` elements is given in 5.7, 6.9, and 6.10.

7.7.3 GrantGroup

Let *t* be an `r:GrantGroup`. Then *t* identifies the Resource that is an XML element Equal to *t*.

NOTE Additional semantics related to `r:GrantGroup` elements is given in 5.7, 6.9, and 6.10.

EXAMPLE
```
<r:grantGroup licensePartId="playAndAdapt">
   <r:keyHolder licensePartIdRef="Alice"/>
   <r:grant>
      <mx:play/>
      <r:digitalResource licensePartIdRef="video"/>
   </r:grant>
   <r:grant>
      <mx:adapt/>
      <r:digitalResource licensePartIdRef="video"/>
   </r:grant>
</r:grantGroup>
```

7.7.4 Revocable

Let *t* be an `r:Revocable`. Then *t* identifies a revocable (*l*, *p*), determined as follows:

— If *t*/`dsig:SignatureValue` is present, then, letting *s* be the `dsig:SignatureType` having a child Equal to *t*/`dsig:SignatureValue`, both *l* is the document to which *s* applies and *p* is the `r:Principal` identifying that issuer of *l* that is established by *s*.

— If *t*/`dsig:Reference` is present, then both of the following are true:

  — *t*/`dsig:Reference` shall reference a `dsig:SignatureType` and,

  — letting *s* be that `dsig:SignatureType`, both *l* is the document to which *s* applies and *p* is the `r:Principal` identifying that issuer of *l* that is established by *s*.

— If *t*/`r:licenseId` is present, then both *p* is Equal to *t*/`r:principal` and *l* is the `r:License` where all of the following are true:

  — the value of *l*/@`r:licenseId` is the value of *t*/`r:licenseId`,

  — there is one *l*/`r:issuer`, and

  — *l*/`r:issuer` corresponds to *p*.

EXAMPLE
```
<r:revocable licensePartId="licenseToBeRevoked">
   <r:licenseId>urn:acme:license:id:12345</r:licenseId>
   <r:keyHolder licensePartIdRef="acme"/>
</r:revocable>
```

7.7.5 ServiceReference

Let t be an `r:ServiceReference`. Then t identifies a reference to a service. The service referred to by t is described by t/`r:serviceDescription`. If t/`r:serviceParameters` is present, the values of the reference-specific parameters needed to interact with the service according to t/`r:serviceDescription` are determined according to the following paragraph, letting p be t/`r:serviceParameters`. If t/`r:serviceParameters` is not present, no reference-specific parameters are needed to interact with the service.

For each integer k between 1 and the number of p/`r:datum` elements, let d_k be the k^{th} p/`r:datum` element. If there are no elements following d_k or if the element immediately following d_k is not an `r:transforms`, then the value of the k^{th} reference-specific parameter needed to interact with the service according to t/`r:serviceDescription` is the child of d_k. If there is an element immediately following d_k and that element is an `r:transforms`, then the value of the k^{th} reference-specific parameter needed to interact with the service according to t/`r:serviceDescription` is determined by the semantics of that `r:transforms` as given in XMLDSIG for `dsig:TransformsType` and modified as follows:

— the input to the first `dsig:Transform` is an XPath node-set containing the child of d_k in the context of a new XML document containing that child as the root element, and

— the output of the last `dsig:Transform` is the value of the k^{th} reference-specific parameter needed to interact with the service according to t/`r:serviceDescription`.

NOTE The interpretation, detailed processing, and passing to the service of the reference-specific parameters is necessarily service-specific and is thus not defined here.

EXAMPLE 1
```
<r:serviceReference licensePartId="uddiService">
   <sx:uddi>
      <sx:serviceKey>
         <sx:uuid>D04951E4-332C-4693-B7DB-D3D1D1C21111</sx:uuid>
      </sx:serviceKey>
   </sx:uddi>
</r:serviceReference>
```

EXAMPLE 2
```
<r:serviceReference licensePartId="anonymousService">
   <sx:anonymousStateService/>
   <r:serviceParameters>
      <r:datum>
         <sx:stateDistinguisher>5001EB7E-80BF-43f7-9065-
7E98CE52279E</sx:stateDistinguisher>
      </r:datum>
   </r:serviceParameters>
</r:serviceReference>
```

7.8 Core conditions

7.8.1 AllConditions

Let c be an `r:AllConditions`. Let q be an authorization request. Let s be an authorization story. Then c is Satisfied with respect to q and s if and only if each c/`r:condition` child of c is Satisfied with respect to q and s.

EXAMPLE
```
<r:allConditions>
   <r:validityInterval licensePartIdRef="firstMonth"/>
   <r:exerciseMechanism>
      <r:exerciseService>
         <r:serviceReference licensePartIdRef="uddiService"/>
      </r:exerciseService>
```

```
   </r:exerciseMechanism>
</r:allConditions>
```

NOTE If c has no children, then c is Satisfied.

7.8.2 ExerciseMechanism

Let c be an `r:ExerciseMechanism`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if c is Equal to Σ.r:exerciseMechanism().

The child of c shall indicate a mechanism for effecting a performance. c/`r:exerciseService`, if present, indicates that the service reference indicated by c/`r:exerciseService`/`r:serviceReference` is the mechanism for effecting a performance.

NOTE An `r:ExerciseMechanism` is particularly useful when used in conjunction with an `r:Obtain` (see Example 1).

EXAMPLE 1 In a super distribution scenario, it is common for the users who wish to obtain an `r:Grant` to be very removed from the original channels of distribution. An `r:ExerciseMechanism` could direct the user back to an official distribution channel.

EXAMPLE 2 A clerk may be permitted to insert records into a database only if he uses a particular (error-checking) user interface form designed for that purpose.

EXAMPLE 3 An airline company may permit its frequent flyers to ticket for a reduced fare when ticketing via a particular online travel service.

EXAMPLE 4

```
<r:exerciseMechanism>
   <r:exerciseService>
      <r:serviceReference licensePartIdRef="uddiService"/>
   </r:exerciseService>
</r:exerciseMechanism>
```

7.8.3 ExistsRight

Let c be an `r:ExistsRight`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let η be the c/`r:grant` or c/`r:grantGroup` child of c, whichever is present. Let Θ be

— the set of `r:Grant` elements identified by c/`r:trustRoot` with respect to (p, r, t, v, Σ, L, R) and authorizer e, if c/`r:trustRoot` is present, or

— R, if c/`r:trustRoot` is not present.

Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if there is an authorizer ε such that both of the following are true:

— if ε is absent, then η is a member of Θ, and

— if ε is present, then, letting (l, π, i, σ, s) be the ordered five-tuple representation of ε, all of the following are true:

  — l is a member of L,

  — η is Equal to one of the l/`r:Grant` or l/`r:GrantGroup` children of l,

  — Σ.r:issueTime(l, π) is i,

  — Σ.r:issueContext(l, π, η, σ) is true,

— the time instant i is no later in time than the start of time interval v, and

— s is an authorization proof for the authorization request (π, I, η, ψ, σ, L, Θ) where ψ is a time interval of zero length starting at i.

EXAMPLE
```
<r:existsRight>
   <r:grant>
      <r:keyHolder licensePartIdRef="Alice"/>
      <mx:play/>
      <r:digitalResource licensePartIdRef="video"/>
      <r:validityInterval licensePartIdRef="firstMonth"/>
   </r:grant>
</r:existsRight>
```

7.8.4 Fulfiller

Let c be an `r:Fulfiller`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if c/`r:principal` is Equal to Σ.r:fulfiller().

EXAMPLE
```
<r:fulfiller>
   <r:keyHolder licensePartIdRef="Alice"/>
</r:fulfiller>
```

7.8.5 PrerequisiteRight

Let c be an `r:PrerequisiteRight`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let Θ be

— the set of `r:Grant` elements identified by c/`r:trustRoot` with respect to (p, r, t, v, Σ, L, R) and authorizer e, if c/`r:trustRoot` is present, or

— R, if c/`r:trustRoot` is not present.

Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if there is an authorization proof for the authorization request (c/`r:principal`, c/`r:right`, c/`r:resource`, v, Σ, L, Θ).

EXAMPLE
```
<r:prerequisiteRight>
   <r:keyHolder licensePartIdRef="Alice"/>
   <mx:play/>
   <r:digitalResource licensePartIdRef="video"/>
</r:prerequisiteRight>
```

7.8.6 RevocationFreshness

Let c be an `r:RevocationFreshness`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if one of the following is true:

— e is absent, or

— both e is present and, letting (l, π, i, σ, s) be the ordered five-tuple representation of e and letting b be the revocable (l, π), there exists an `r:RevocationMechanism` m and a time instant τ such that all of the following are true:

— m is Equal to one of the l/`r:issuer`/`r:details`/`r:revocationMechanism` elements,

— τ is greater than or equal to the start of v less the duration c/r:priorToStart, and

— Σ.r:freshness(m, b, τ) is true.

NOTE 1 Having an r:priorToStart of 0 would make it difficult, if not impossible, to have a revocation mechanism that is queried in real time because of the trouble with synchronizing the revocation query response with the start of the performance. On the other hand, an r:priorToStart of 0 does not pose much difficulty if the revocation mechanism is a list that is sent out every midnight and is valid until the next midnight, as freshness is guaranteed for any start time between the two midnights. When using a revocation mechanism that is queried in real time, it is generally expected that the r:priorToStart will be set to an appropriate value for that mechanism.

NOTE 2 r:priorToStart can be negative. A negative r:priorToStart does not work if the revocation mechanism only guarantees freshness through the time at which it is queried. However, if the revocation mechanism can guarantee freshness through some time in the future, a negative r:priorToStart can be used to require that freshness is guaranteed for some duration after the start of the performance.

EXAMPLE
```
<r:revocationFreshness licensePartId="revFreshness">
   <r:priorToStart>P1D</r:priorToStart>
</r:revocationFreshness>
```

7.8.7 ValidityInterval

Let c be an r:ValidityInterval. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if both of the following are true:

— if c/r:notBefore is present, the start of v is greater than or equal to the instant in time represented by the value of c/r:notBefore, and

— if c/r:notAfter is present, the end of v is less than or equal to the instant in time represented by the value of c/r:notAfter.

NOTE If r is I and t is an r:Grant or r:GrantGroup. Then, c constrains when t can be included in an r:License l. It does not speak to the validity of l. An r:Grant or r:GrantGroup element in l can contain a different r:ValidityInterval from c.

EXAMPLE
```
<r:validityInterval licensePartId="firstMonth">
   <r:notBefore>2003-01-01T00:00:00</r:notBefore>
   <r:notAfter>2003-01-31T23:59:59</r:notAfter>
</r:validityInterval>
```

7.9 Core patterns

7.9.1 General patterns

7.9.1.1 AnXmlExpression

Let a be an r:AnXmlExpression. Let x be an XML document. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if the expression contained in a evaluates to true over x according to the semantics of the expression language indicated by a/@r:lang, if present.

When a/@r:lang is absent or its value is http://www.w3.org/TR/1999/REC-xpath-19991116, the expression contained in a is written in XPath and, if that expression is not of XPath type boolean, it is to be converted to such as if the XPath function boolean were applied.

Applications that support the use of any form of patterns at all should support the use of the http://www.w3.org/TR/1999/REC-xpath-19991116 expression language in r:AnXmlExpression elements.

EXAMPLE
```
<r:forAll varName="acmeVideos">
   <r:anXmlExpression>mx:diReference/mx:identifier[starts-with( .,
"urn:acme:video:")]</r:anXmlExpression>
</r:forAll>
```

7.9.1.2 PatternFromLicensePart

Let a be an `r:PatternFromLicensePart`. Let x be an XML document. Let m be the root element contained in x. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if m is Equal to a/`r:licensePart`.

7.9.2 Principal patterns – PropertyPossessor

Let a be an `r:PropertyPossessor`. Let x be an XML document. Let m be the root element contained in x. Let (p, r, t, v, Σ, L, R) be an authorization request. Let e be an authorizer. Let Θ be

— the set of `r:Grant` elements represented by a/`r:trustRoot` with respect to (p, r, t, v, Σ, L, R) and e, if a/`r:trustRoot` is present, or

— R, if a/`r:trustRoot` is not present.

Then x Matches a with respect to (p, r, t, v, Σ, L, R) and e if and only if both m is an `r:Principal` and there is an authorization proof for the authorization request (m, P, a/`r:propertyAbstract`, v, Σ, L, Θ).

EXAMPLE
```
<r:propertyPossessor licensePartId="acmeSubscriber">
   <sx:propertyUri definition="urn:acme:subscriber"/>
   <r:trustedRootIssuers licensePartIdRef="acmeCA"/>
</r:propertyPossessor>
```

7.9.3 Right patterns

No other right patterns are defined in the Core Namespace.

7.9.4 Resource patterns

7.9.4.1 GrantGroupPattern

Let a be an `r:GrantGroupPattern`. Let x be an XML document. Let m be the root element contained in x. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if all of the following are true:

— m is an `r:GrantGroup`,

— if a/`r:principal` is present, m/`r:principal` is Equal to a/`r:principal`,

— for each a/`r:principalPattern`/`r:anXmlExpression` or a/`r:principalPattern`/`r:principalPatternAbstract` α that is present, a new XML document containing m/`r:principal` as the root element Matches α with respect to q and e,

— if a/`r:condition` is present, m/`r:condition` is Equal to a/`r:condition`,

— for each a/`r:conditionPattern`/`r:anXmlExpression` or a/`r:conditionPattern`/`r:conditionPatternAbstract` α that is present, a new XML document containing m/`r:condition` as the root element Matches α with respect to q and e,

— letting n be the number of a/`r:grant`, a/`r:grantPattern`, a/`r:grantGroup`, and a/`r:grantGroupPattern` children of a, letting α_k be the k^{th} a/`r:grant`, a/`r:grantPattern`,

a/r:grantGroup, or a/r:grantGroupPattern child of a, and letting μ_k be the k^{th} m/r:grant or m/r:grantGroup child of m, both of the following are true:

— n is also the number of m/r:grant and m/r:grantGroup children of m, and

— for each k from 1 to n, both of the following are true:

— if α_k is an r:Grant or an r:GrantGroup then μ_k is Equal to α_k, and

— if α_k is an r:GrantPattern or an r:GrantGroupPattern then a new document containing μ_k as the root element Matches α_k, and

— for each a/r:wholeGrantGroupExpression α that is present, x Matches α with respect to q and e.

EXAMPLE
```
<r:grantGroupPattern licensePartId="adaptAndPlayVideo">
   <r:grant>
      <mx:adapt/>
      <r:digitalResource licensePartIdRef="video"/>
   </r:grant>
   <r:grant>
      <mx:play/>
      <r:digitalResource licensePartIdRef="video"/>
   </r:grant>
</r:grantGroupPattern>
```

7.9.4.2 GrantPattern

Let a be an r:GrantPattern. Let x be an XML document. Let m be the root element contained in x. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if all of the following are true:

— m is an r:Grant,

— if a/r:principal is present, m/r:principal is Equal to a/r:principal,

— for each a/r:principalPattern/r:anXmlExpression or a/r:principalPattern/r:principalPatternAbstract α that is present, a new XML document containing m/r:principal as the root element Matches α with respect to q and e,

— if a/r:right is present, m/r:right is Equal to a/r:right,

— for each a/r:rightPattern/r:anXmlExpression or a/r:rightPattern/r:rightPatternAbstract α that is present, a new XML document containing m/r:right as the root element Matches α with respect to q and e,

— if a/r:resource is present, m/r:resource is Equal to a/r:resource,

— for each a/r:resourcePattern/r:anXmlExpression or a/r:resourcePattern/r:resourcePatternAbstract α that is present, a new XML document containing m/r:resource as the root element Matches α with respect to q and e,

— if a/r:condition is present, m/r:condition is Equal to a/r:condition,

— for each a/r:conditionPattern/r:anXmlExpression or a/r:conditionPattern/r:conditionPatternAbstract α that is present, a new XML document containing m/r:condition as the root element Matches α with respect to q and e, and

— for each a/r:wholeGrantExpression α that is present, x Matches α with respect to q and e.

NOTE An a/r:rightPattern with no children can be used to cause an r:Grant g to Match a irrespective of which g/r:right child it has.

EXAMPLE
```
<r:grantPattern licensePartId="playVideo">
   <mx:play/>
   <r:digitalResource licensePartIdRef="video"/>
</r:grantPattern>
```

7.9.5 Condition patterns

No other condition patterns are defined in the Core Namespace.

7.10 Core delegation constraints

7.10.1 ConditionIncremental

Let d be an `r:ConditionIncremental`. Let q be an authorization request. Let e be an authorizer. Let g be an `r:Grant` or `r:GrantGroup`. Let p be an `r:Principal`. Let c be an `r:Condition`. Let δ be an `r:delegationControl`.

Then p is an Allowable Destination Principal of d with respect to q, e, and g.

Further c is an Allowable Destination Condition of d with respect to q, e, and g if and only if either of the following are true:

— c is Equal to g/`r:condition`, or

— c is Equal to an `r:allConditions` element containing g/`r:condition` as its first child along with any number of other children.

Further δ is an Allowable Destination Delegation Control of d with respect to q, e, and g if and only if there is a δ/`r:dcConstraint` that is either Equal to d or Equal to an `r:ConditionUnchanged` element.

7.10.2 ConditionUnchanged

Let d be an `r:ConditionUnchanged`. Let q be an authorization request. Let e be an authorizer. Let g be an `r:Grant` or `r:GrantGroup`. Let p be an `r:Principal`. Let c be an `r:Condition`. Let δ be an `r:delegationControl`.

Then p is an Allowable Destination Principal of d with respect to q, e, and g.

Further c is an Allowable Destination Condition of d with respect to q, e, and g if and only if c is Equal to g/`r:condition`.

Further δ is an Allowable Destination Delegation Control of d with respect to q, e, and g if and only if there is a δ/`r:dcConstraint` that is Equal to d.

7.10.3 DepthConstraint

Let d be an `r:DepthConstraint`. Let q be an authorization request. Let e be an authorizer. Let g be an `r:Grant` or `r:GrantGroup`. Let p be an `r:Principal`. Let c be an `r:Condition`. Let δ be an `r:delegationControl`.

Then p is an Allowable Destination Principal of d with respect to q, e, and g.

Further c is an Allowable Destination Condition of d with respect to q, e, and g.

Further δ is an Allowable Destination Delegation Control of d with respect to q, e, and g if and only if there is a δ/`r:dcConstraint` z that is Equal to d, except that z/`r:count` is any nonnegative integer strictly less than d/`r:count`.

7.10.4 ToConstraint

Let d be an `r:ToConstraint`. Let q be an authorization request. Let e be an authorizer. Let g be an `r:Grant` or `r:GrantGroup`. Let p be an `r:Principal`. Let c be an `r:Condition`. Let δ be an `r:delegationControl`.

Then p is an Allowable Destination Principal of d with respect to q, e, and g if and only if both of the following are true:

— if d does not have at least one d/`r:forAll` child element, then there is a d/`r:principal` that is Equal to p, and

— if d has at least one d/`r:forAll` child element, then, letting f be the first d/`r:forAll` child of d, there is an `r:ToConstraint` t such that all of the following are true:

  — t is Equal to d, except that

    — f is not present in t and

    — there exists an x in the eligible bindings of the Variable declared by f with respect to q and e such that, throughout the scope of that Variable in d, all elements having references to that Variable are replaced in t by x.

  — p is an Allowable Destination Principal of t with respect to q, e, and g.

Further c is an Allowable Destination Condition of d with respect to q, e, and g.

Further δ is an Allowable Destination Delegation Control of d with respect to q, e, and g if and only if there is a δ/`r:dcConstraint` z that is Equal to d except for the following variations:

— z may make zero, one, or more additions of an `r:forAll` child preceding any that may be present in d.

— z may make zero, one, or more omissions of some d/`r:principal` that may be present in d.

— z may make zero, one, or more replacements of some d/`r:principal` π that may be present in d by replacing it with an `r:allPrincipals` element containing π as its first child along with any number of other children.

EXAMPLE

```
<r:delegationControl>
   <r:conditionIncremental/>
   <r:depthConstraint>
      <r:count>3</r:count>
   </r:depthConstraint>
   <r:toConstraint>
      <r:forAll varName="delegatedTo">
         <r:propertyPossessor licensePartIdRef="acmeSubscriber"/>
      </r:forAll>
      <r:keyHolder varRef="delegatedTo"/>
   </r:toConstraint>
</r:delegationControl>
```

7.11 Core trust roots

7.11.1 TrustedRootGrants

Let z be an `r:TrustedRootGrants`. Let q be an authorization request. Let e be an authorizer. Then z identifies with respect to q and e the set R of all z/`r:grant` children of z.

7.11.2 TrustedRootIssuers

Let z be an `r:TrustedRootIssuers`. Let q be an authorization request. Let e be an authorizer. Then z identifies with respect to q and e the set R of `r:Grant` elements such that for each z/`r:principal` child p of z there is exactly one `r:Grant` g in R such that all of the following are true:

— exactly one g/`r:forAll` is present with an `r:varName` attribute whose value is `x` and no element content,

— g/`r:principal` is Equal to p,

— g/`r:right` is Equal to I,

— g/`r:resource` is Equal to an `r:resource` element with an `r:varRef` attribute whose value is `x` and no element content, and

— g has no other children.

EXAMPLE

```
<r:trustedRootIssuers licensePartId="acmeCA">
   <r:keyHolder licensePartIdRef="acme"/>
</r:trustedRootIssuers>
```

7.12 Core service descriptions

No other service descriptions are defined in the Core Namespace.

7.13 Core properties

No other properties are defined in the Core Namespace.

8 Standard extension

8.1 General

This Clause specifies syntax and semantics peripheral to the architecture but still useful in many domains beyond multimedia.

8.2 Standard extension authorization context properties

Table 4 specifies the authorization context properties relating to Clause 8 and the statements they represent. If a property has the name given in the first column of Table 4 and the value given in the second column of Table 4, then the statement represented by that property is the statement given in the third column of Table 4.

Table 4 — Standard extension authorization context properties

Property name	Property value	Statement represented
sx:cF(d)	true	d is an `r:ServiceDescription`, and there is a state of communication failure with the service described by d.
sx:cFC(d, ρ)	c	d is an `r:ServiceDescription`, ρ is an ordered tuple, c is an `r:Condition`, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance.
sx:eL(d, ρ)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance. NOTE It is generally expected that the value of the property named sx:eL(d, ρ) will be true when the number of performances in some class of which the requested performance is a member has not yet reached its limit.
sx:eLC(d, ρ)	n	d is an `r:ServiceDescription`, ρ is an ordered tuple, n is an integer, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance. NOTE It is generally expected that the value of the property named sx:eLC(d, ρ) will correlate to the number of performances in some class of which the requested performance is a member.
sx:fA()	a	a is an `sx:AccountPayable`, and a is the default `sx:AccountPayable` to use for the requested performance.
sx:fF(d, ρ, a, τ)	ϕ	d is an `r:ServiceDescription`, ρ is an ordered tuple, a is an `sx:AccountPayable`, τ is a time instant, ϕ is an `sx:Rate`, and the service described by d claims that, at τ, the amount of money indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for flat fees with respect to ρ. NOTE Services will vary in their policies for what they consider paid. Some services might have policies that only consider money paid after a payment is actually made. Others might consider money paid as soon as a bill is generated for it. Others might consider money paid when a considers it paid.
sx:fG(a)	ϕ	a is an `sx:AccountPayable`, ϕ is an `sx:Rate`, and the amount of money indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for the requested performance. The meaning of *considered paid* is payment mechanism-specific.
sx:fPI(d, ρ, v, a)	ϕ	d is an `r:ServiceDescription`, ρ is an ordered tuple, v is a time interval, a is an `sx:AccountPayable`, ϕ is an `sx:Rate`, and the service described by d claims that, at the start of v, the amount of money indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for v with respect to ρ. NOTE Services will vary in their policies for what they consider paid. Some services might have policies that only consider money paid after a payment is actually made. Others might consider money paid as soon as a bill is generated for it. Others might consider money paid when a considers it paid.

Table 4 *(continued)*

sx:fPUPP(d, ρ, ϕ, n, a)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, ϕ is an `sx:Rate`, n is an positive integer, a is an `sx:AccountPayable`, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance. NOTE It is generally expected that the value of the property named sx:fPUPP(d, ρ, ϕ, n, a) will be true when the number of performances in some class of which the requested performance is a member has not yet reached the number of performances considered paid for using the mechanism for making a payment indicated by a to the account indicated by a at the rate given by the amount of money indicated by ϕ per each package of n performances.
sx:sA(d, ρ)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance.
sx:sRVP(d, ρ, v)	a	d is an `r:ServiceDescription`, ρ is an ordered tuple, v is a time interval, a is an `sx:StateReferenceValuePattern`, and the service described by d claims that the children of a represent the state of the service throughout v with respect to ρ.
sx:tC(d, ρ, v)	p	d is an `r:ServiceDescription`, ρ is an ordered tuple, v is a time interval, p is an `r:Principal`, and the service described by d claims that, throughout v, p identifies the owner of some virtual token with respect to ρ. NOTE 1 Typically the virtual token would be identified entirely by ρ in a service-specific fashion but, for some services, might also be known to the service by other means. NOTE 2 Though no normative requirement is placed on how the service deals with the change of ownership of a virtual token, it is generally expected that many services will provide features to allow the ordered exchange of ownership from one Principal to another.
sx:tD(a)	true	a is a URI identifying a digital location, and the requested performance occurs in the digital location identified by a.
sx:tL(a, b, w, x, y, z)	true	a is a Qualified Name identifying a country, b is a Qualified Name identifying a country subdivision, w is a string identifying a state, x is a string identifying a city, y is a postal code, z is a street address, and the requested performance occurs at z in y and in the city identified by x and in the state identified by w and in the country subdivision identified by b and in the country identified by a. NOTE Some Qualified Names identifying countries and country subdivisions are defined in Annex B. EXAMPLE 1 An example string identifying a state is a two-letter code for US states. EXAMPLE 2 An example postal code is a zip code.
sx:tQ(d, ρ, τ)	n	d is an `r:ServiceDescription`, ρ is an ordered tuple, τ is a time instant, n is an integer, and the service described by d claims that, at τ, n is the track query integral value managed with respect to ρ.

Table 4 *(continued)*

sx:tR(d, ρ)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, and the service described by d claims that it considers the requested performance tracked with respect to ρ. NOTE Services will vary in their policies for what they consider tracked. Some services might have policies that only consider a performance tracked if they are contacted about it in a timely manner, such as before the performance begins or after it ends. Others might require that they be contacted at the end of the day in which the performance begins.
sx:vIFD(d, ρ)	τ	d is an `r:ServiceDescription`, ρ is an ordered tuple, τ is a time instant, and the service described by d claims that τ is the determined floating start time with respect to ρ. NOTE It is generally expected that the determined floating start time will correspond with the start of the first performance where the authorization context used to established permission for that performance has a property named sx:vIFD(d, ρ).
sx:vIF(d, ρ)	v	d is an `r:ServiceDescription`, ρ is an ordered tuple, v is a time interval, and the service described by d claims that v is the determined floating interval with respect to ρ. NOTE It is generally expected that the start of the determined floating interval will correspond with the start of the first performance where the authorization context used to established permission for that performance has a property named sx:vIF(d, ρ).
sx:vTM(d, ρ, v)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, v is a time interval, and the service described by d claims that all of v is valid metered time with respect to ρ.
sx:vTMD(d, ρ, τ)	y	d is an `r:ServiceDescription`, ρ is an ordered tuple, τ is a time instant, y is a time duration, and the service described by d claims that the total duration of time before τ that it has claimed, claims, or will claim as valid metered time with respect to ρ is y.

8.3 General standard extension elements and types

8.3.1 AccountPayable

Let a be an `sx:AccountPayable`. The child of a shall indicate a mechanism for making a payment and an account to pay. a/`sx:paymentService`, if present, indicates that the service reference identified by a/`sx:paymentService`/`r:serviceReference` is the mechanism for making a payment and the account to pay is determined in a service-specific fashion. a/`sx:aba`, if present, indicates that the American Bankers Association routing number (ROUTINGABA) given by the value of a/`sx:aba`/`sx:institution` is the mechanism for making a payment and the account to pay is indicated by the routing number along with the account given by the value of a/`sx:aba`/`sx:account`.

8.3.2 ProfileCompliance

For an `r:License` l, each list member in the value of the l/@`sx:profileCompliance` attribute, if the attribute is present, shall provide a Qualified Name representing one profile with which the License is compliant. If present, this attribute need not provide a complete list of the profiles with which the License is compliant. Applications may ignore this attribute. If they do process it, they may ignore any members of the list.

8.3.3 Rate

Let ϕ be an `sx:Rate`. Then ϕ indicates an amount of money whose amount (as a float number) is the value of ϕ/`sx:amount` and whose currency is identified by the value of ϕ/`sx:currency`, if ϕ/`sx:currency` is present, or is USD, if ϕ/`sx:currency` is absent.

NOTE Some Qualified Names identifying currencies are defined in Annex B.

8.3.4 StateDistinguisher

There is no normative semantics for `sx:stateDistinguisher`.

NOTE This type exists merely due to the observation that it is often convenient to be able to use it within an `r:Datum` to hold simple content to distinguish which state pertains to an `sx:StatefulCondition`.

EXAMPLE
```
<sx:stateDistinguisher>5001EB7E-80BF-43f7-9065-7E98CE52279E</sx:stateDistinguisher>
```

8.3.5 UddiKey

Let k be an `sx:UddiKey`. If k/`sx:uuid` is present, then the primary key represented by k is the value of k/`sx:uuid`. If k/`sx:uri` is present, then the primary key represented by k is the value of k/`sx:uri`.

8.3.6 Uuid

Let u be an `sx:Uuid`. Then the value of u is a Universally Unique Identifier as defined in UDDIV2DSR.

8.4 Standard extension conceptually abstract elements and types

8.4.1 Name

The element `sx:name` is conceptually abstract. The type `sx:Name` is conceptually abstract. An `sx:Name` shall identify a name that can be possessed by a Principal.

NOTE An `sx:Name` can be used along with an `sx:PossessProperty` to associate a name with a Principal. Such associations allow other `r:Grant` elements to refer to Principals described by their names.

8.4.2 StatefulCondition

The type `sx:StatefulCondition` is conceptually abstract. Let c be an `sx:StatefulCondition`. Let e be an authorizer.

If c/`r:serviceReference` is present, the state reference of c with respect to e is c/`r:serviceReference`.

If c/`r:serviceReference` is absent and e is absent, the state reference of c with respect to e is undefined.

If c/`r:serviceReference` is absent and e is present, then, letting (l, π, i, σ, s) be the ordered five-tuple representation of e, the state reference of c with respect to e is an `r:ServiceReference` m such that m/`r:serviceDescription` is Equal to an `sx:anonymousStateService` element, there is exactly one m/`r:serviceParameters`/`r:datum`, and the child of that `r:datum` element is Equal to l.

8.5 Standard extension principals

No other principals are defined in the Standard Extension Namespace.

8.6 Standard extension rights – RightUri

Let r be an `sx:RightUri`. Then r identifies the act identified by r/@`sx:definition`.

EXAMPLE
```
<sx:rightUri definition="urn:acme:copy"/>
```

8.7 Standard extension resources

No other resources are defined in the Standard Extension Namespace.

8.8 Standard extension conditions

8.8.1 CallForCondition

Let c be an `sx:CallForCondition`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if there is a c/`r:serviceReference` m such that, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:cFC(m/`r:serviceDescription`, ρ) is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e).

EXAMPLE
```
<sx:callForCondition>
   <r:serviceReference licensePartIdRef="uddiService"/>
</sx:callForCondition>
```

8.8.2 ExerciseLimit

Let c be an `sx:ExerciseLimit`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, both of the following are true:

— if c/`sx:count` is present, Σ.sx:eLC(m/`r:serviceDescription`, ρ) is less than or equal to the value of c/`sx:count`, and

— if c/`sx:count` is absent, Σ.sx:eL(m/`r:serviceDescription`, ρ) is true.

EXAMPLE
```
<sx:exerciseLimit>
   <r:serviceReference licensePartIdRef="anonymousService"/>
   <sx:count>3</sx:count>
</sx:exerciseLimit>
```

8.8.3 FeeFlat

Let c be an `sx:FeeFlat`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let τ be the time instant at the start of v. Let a be c/`sx:to`, if c/`sx:to` is present, or Σ.sx:fA(), if c/`sx:to` is absent. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:fF(m/`r:serviceDescription`, ρ, a, τ) indicates an amount of money that is greater or equal in amount to and of the same currency as the amount of money indicated by c/`sx:rate`.

EXAMPLE
```
<sx:feeFlat xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency">
   <r:serviceReference licensePartIdRef="uddiService"/>
   <sx:rate>
```

```
      <sx:amount>1.00</sx:amount>
      <sx:currency>iso:JPY</sx:currency>
   </sx:rate>
   <sx:to>
      <sx:aba>
         <sx:institution>123456789</sx:institution>
         <sx:account>987654321</sx:account>
      </sx:aba>
   </sx:to>
</sx:feeFlat>
```

8.8.4 FeeMetered

Let c be an `sx:FeeMetered`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be c/`sx:to`, if c/`sx:to` is present, or Σ.sx:fA(), if c/`sx:to` is absent. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.sx:fG(a) indicates an amount of money that is greater or equal in amount to and of the same currency as

$$x \times (y/z) \times (\text{floor}(w/y) + \theta)$$

where

— x is the amount of money indicated by c/`sx:rate`,

— y is the numerical value of c/`sx:by` in seconds,

— z is the numerical value of c/`sx:per` in seconds,

— w is the numerical value of the duration of v in seconds, and

— θ is 1 if w mod y is greater than the numerical value of c/`sx:phase` in seconds and 0 otherwise.

EXAMPLE

```
<sx:feeMetered xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency">
   <sx:rate>
      <sx:amount>24.00</sx:amount>
      <sx:currency>iso:USD</sx:currency>
   </sx:rate>
   <sx:per>P1D</sx:per>
   <sx:by>PT1H</sx:by>
   <sx:phase>PT30M</sx:phase>
</sx:feeMetered>
```

8.8.5 FeePerInterval

Let c be an `sx:FeePerInterval`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be c/`sx:to`, if c/`sx:to` is present, or Σ.sx:fA(), if c/`sx:to` is absent. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, both of the following are true:

— for every time instant τ in v there exists some time interval δ such that both δ contains τ and Σ.sx:fPI(m/`r:serviceDescription`, ρ, δ, a) indicates an amount of money that is greater or equal in amount to and of the same currency as the amount of money indicated by c/`sx:rate`, and

— there does not exist some time interval δ such that Σ has a property named sx:fPI(m/`r:serviceDescription`, ρ, δ, a) and δ is not equal in duration to the value of c/`sx:per`.

EXAMPLE
```
<sx:feePerInterval xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency">
   <r:serviceReference licensePartIdRef="uddiService"/>
   <sx:rate>
      <sx:amount>5.00</sx:amount>
      <sx:currency>iso:USD</sx:currency>
   </sx:rate>
   <sx:per>P1D</sx:per>
</sx:feePerInterval>
```

8.8.6 FeePerUse

Let c be an `sx:FeePerUse`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be c/`sx:to`, if c/`sx:to` is present, or Σ.sx:fA(), if c/`sx:to` is absent. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.sx:fG(a) indicates an amount of money that is greater or equal in amount to and of the same currency as the amount of money indicated by c/`sx:rate`.

EXAMPLE
```
<sx:feePerUse xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency">
   <sx:rate>
      <sx:amount>2.00</sx:amount>
      <sx:currency>iso:USD</sx:currency>
   </sx:rate>
</sx:feePerUse>
```

8.8.7 FeePerUsePrePay

Let c be an `sx:FeePerUsePrePay`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let ϕ be c/`sx:rate`. Let n be the value of c/`sx:initialNumberOfUses`, if c/`sx:initialNumberOfUses` is present, or 1 if c/`sx:initialNumberOfUses` is absent. Let a be c/`sx:to`, if c/`sx:to` is present, or Σ.sx:fA(), if c/`sx:to` is absent. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:fPUPP(m/`r:serviceDescription`, ρ, ϕ, n, a) is true.

EXAMPLE
```
<sx:feePerUsePrePay xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency">
   <r:serviceReference licensePartIdRef="uddiService"/>
   <sx:rate>
      <sx:amount>3.00</sx:amount>
      <sx:currency>iso:USD</sx:currency>
   </sx:rate>
   <sx:initialNumberOfUses>7</sx:initialNumberOfUses>
</sx:feePerUsePrePay>
```

8.8.8 SeekApproval

Let c be an `sx:SeekApproval`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:sA(m/`r:serviceDescription`, ρ) is true.

EXAMPLE
```
<sx:seekApproval>
   <r:serviceReference licensePartIdRef="uddiService"/>
</sx:seekApproval>
```

8.8.9 Territory

Let c be an `sx:Territory`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if there is either at least one c/`sx:domain` child γ of c such that Σ.sx:tD(γ/`sx:uri`) is true or at least one c/`sx:location` child γ of c such that there exists Qualified Names a and b and strings w, x, y, and z such that all of the following are true:

— Σ.sx:tL(a, b, w, x, y, z) is true,

— if γ/`sx:country` is present, its value is a,

— if γ/`sx:region` is present, its value is b,

— if γ/`sx:state` is present, its value is w,

— if γ/`sx:city` is present, its value is x,

— if γ/`sx:postalCode` is present, its value is y, and

— if γ/`sx:street` is present, its value is z.

EXAMPLE

```
<sx:territory xmlns:iso="urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:country">
   <sx:location>
      <sx:country>iso:US</sx:country>
   </sx:location>
   <sx:domain>
      <sx:uri>urn:acme:domains:123</sx:uri>
   </sx:domain>
</sx:territory>
```

8.8.10 TrackQuery

Let c be an `sx:TrackQuery`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let τ be the time instant at the start of v. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:tQ(m/`r:serviceDescription`, ρ, τ) is both greater than or equal to the value of c/`sx:notLessThan`, if c/`sx:notLessThan` is present, and less than or equal to the value of c/`sx:notMoreThan`, if c/`sx:notMoreThan` is present.

EXAMPLE

```
<sx:trackQuery>
   <r:serviceReference licensePartIdRef="uddiService"/>
   <sx:notLessThan>5</sx:notLessThan>
   <sx:notMoreThan>10</sx:notMoreThan>
</sx:trackQuery>
```

8.8.11 TrackReport

Let c be an `sx:TrackReport`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, either Σ.sx:tR(m/`r:serviceDescription`, ρ) is true or both the value of c/`sx:communicationFailurePolicy` is `lax` and Σ.sx:cF(m/`r:serviceDescription`) is true.

EXAMPLE
```
<sx:trackReport>
   <r:serviceReference licensePartIdRef="uddiService"/>
   <sx:communicationFailurePolicy>required</sx:communicationFailurePolicy>
</sx:trackReport>
```

8.8.12 TransferControl

Let c be an `sx:TransferControl`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:tC(m/`r:serviceDescription`, ρ, v) is Equal to p.

EXAMPLE
```
<r:grant>
   <r:delegationControl>
      <r:conditionUnchanged/>
      <r:depthConstraint>
         <r:count>1</r:count>
      </r:depthConstraint>
   </r:delegationControl>
   <r:keyHolder licensePartIdRef="Alice"/>
   <mx:play/>
   <mx:diReference licensePartIdRef="video"/>
   <sx:transferControl>
      <r:serviceReference>
         <sx:anonymousStateService/>
         <r:serviceParameters>
            <r:datum>
               <sx:stateDistinguisher>5001EB7E-80BF-43f7-9065-
7E98CE52279E</sx:stateDistinguisher>
            </r:datum>
         </r:serviceParameters>
      </r:serviceReference>
   </sx:transferControl>
</r:grant>
```

8.8.13 ValidityIntervalFloating

Let c be an `sx:ValidityIntervalFloating`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, both of the following are true:

— if c/`sx:duration` is present, v is wholly contained within the interval starting at Σ.sx:vIFD(m/`r:serviceDescription`, ρ) and lasting the value of c/`sx:duration`, and

— if c/`sx:duration` is absent, v is wholly contained within Σ.sx:vIF(m/`r:serviceDescription`, ρ).

EXAMPLE
```
<sx:validityIntervalFloating>
   <sx:duration>P2D</sx:duration>
</sx:validityIntervalFloating>
```

8.8.14 ValidityIntervalStartsNow

Let c be an `sx:ValidityIntervalStartsNow`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if both of the following hold:

— if *c*/sx:backwardTolerance is present, then both *c*/r:validityInterval/r:notBefore is present and its value is greater than or equal to the result of the start of v set backward by the value *c*/sx:backwardTolerance, and

— if *c*/sx:forwardTolerance is present, then *c*/r:validityInterval/r:notBefore is either

  — absent or

  — present and less than or equal to the result of the start of v set forward by the value of *c*/sx:forwardTolerance.

NOTE This condition is especially useful when used in conjunction with an sx:ValidityIntervalDurationPattern.

EXAMPLE
```
<sx:validityIntervalStartsNow>
   <r:validityInterval>
      <r:notBefore>2003-06-25T00:00:00</r:notBefore>
      <r:notAfter>2003-07-25T23:59:59</r:notAfter>
   </r:validityInterval>
   <sx:backwardTolerance>P1D</sx:backwardTolerance>
   <sx:forwardTolerance>P2D</sx:forwardTolerance>
</sx:validityIntervalStartsNow>
```

8.8.15 ValidityTimeMetered

Let c be an sx:ValidityTimeMetered. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be the state reference of c as defined by 8.4.2 with respect to e. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if either m is undefined or, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, all of the following are true:

— for every time instant τ in v, there exists some time interval δ such that both Σ.sx:vTM(m/r:serviceDescription, ρ, δ) is true and δ contains τ,

— if *c*/sx:duration is present, then, letting τ be the time instant at the end of v, Σ.sx:vTMD(m/r:serviceDescription, ρ, τ) is less than or equal to the value of *c*/sx:duration, and

— if *c*/sx:quantum is present, then there does not exist some time interval δ such that both Σ.sx:vTM(m/r:serviceDescription, ρ, δ) is true and δ is shorter in duration than the value of *c*/sx:quantum.

NOTE When *c*/sx:duration is absent, it is generally expected that the state reference used will refer to a service that has its own limit as to how much time it claims as valid metered time. On the other hand, if *c*/sx:duration is present, the service will typically just keep track of how much time it claims as valid metered time but not impose any of its own limits.

EXAMPLE
```
<sx:validityTimeMetered>
   <sx:duration>PT1H</sx:duration>
   <sx:quantum>PT15S</sx:quantum>
</sx:validityTimeMetered>
```

8.8.16 ValidityTimePeriodic

Let c be an sx:ValidityTimePeriodic. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be the value of *c*/sx:start, b be the value of *c*/sx:period, x be the value of *c*/sx:phase if it is present or 0 otherwise, y be the value of *c*/sx:duration, and z be the value of *c*/sx:periodCount if it is present. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if for every time instant τ in v there exists an integer n such that all of the following are true:

— the inequality a + n x b + x <= τ <= a + n x b + x + y holds,

— if x >= 0 then n >= 0,

— if x < 0 then n >= 1,

— if x >= 0 and c/`sx:periodCount` is present then n <= z - 1, and

— if x < 0 and c/`sx:periodCount` is present then n <= z.

NOTE This condition is useful to express time windows such as every weekend or the second week of every month.

EXAMPLE
```
<sx:validityTimePeriodic>
   <sx:start>2003-01-01T00:00:00</sx:start>
   <sx:period>P1M</sx:period>
   <sx:phase>P7D</sx:phase>
   <sx:duration>P2D</sx:duration>
   <sx:periodCount>12</sx:periodCount>
</sx:validityTimePeriodic>
```

8.9 Standard extension patterns

8.9.1 General patterns

8.9.1.1 LicenseIdPattern

Let a be an `sx:LicenseIdPattern`. Let x be an XML document. Let m be the root element contained in x. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (l, π, i, σ, s) be an authorizer. Then x Matches a with respect to (p, r, t, v, Σ, L, R) and (l, π, i, σ, s) if and only if both m contains only simple content and that simple content is the value of l/@`r:licenseId`.

8.9.1.2 StateReferenceValuePattern

Let a be an `sx:StateReferenceValuePattern`. Let x be an XML document. Let m be the root element contained in x. Let (p, r, t, v, Σ, L, R) be an authorization request. Let e be an authorizer. Then x Matches a with respect to (p, r, t, v, Σ, L, R) and e if and only if both m is an `r:ServiceReference` and, letting ρ be the ordered tuple containing the values of the reference-specific parameters determined by m, Σ.sx:sRVP(m/`r:serviceDescription`, ρ, v) is a.

NOTE An `r:StateReferenceValuePattern` is typically used with an `r:Obtain` to give the user of the `r:Grant` with the `r:Obtain` an idea about the initial state of an `r:ServiceReference` that would appear in the `r:Grant` he would obtain.

EXAMPLE
```
<sx:stateReferenceValuePattern>
   <acme:value>6</acme:value>
</sx:stateReferenceValuePattern>
```

8.9.2 Principal patterns

No other principal patterns are defined in the Standard Extension Namespace.

8.9.3 Right patterns

No other right patterns are defined in the Standard Extension Namespace.

8.9.4 Resource patterns – X509SubjectNamePattern

Let a be an `sx:X509SubjectNamePattern`. Let x be an XML document. Let m be the root element contained in x. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if both m is an `sx:X509SubjectName` and the content of a is the root of the subject name tree as specified in ISO/IEC 9594-8 for the subject name identified by m.

EXAMPLE
```
<r:forAll varName="acmeDistributorCertificates">
   <sx:x509SubjectNamePattern>CN=Acme Distributor</sx:x509SubjectNamePattern>
</r:forAll>
```

8.9.5 Condition patterns – ValidityIntervalDurationPattern

Let a be an `sx:ValidityIntervalDurationPattern`. Let x be an XML document. Let m be the root element contained in x. Let q be an authorization request. Let e be an authorizer. Then x Matches a with respect to q and e if and only if both m is an `r:ValidityInterval` and the duration of the interval represented by m is equal to the value of a/`sx:duration`.

NOTE This pattern is useful to express intervals that are fixed at some License issuing step, either for business model reasons or to minimize the amount of state keeping done by a compact device.

EXAMPLE
```
<sx:validityIntervalDurationPattern>
   <sx:duration>P1M</sx:duration>
</sx:validityIntervalDurationPattern>
```

8.10 Standard extension delegation constraints

No other delegation constraints are defined in the Standard Extension Namespace.

8.11 Standard extension trust roots

No other trust roots are defined in the Standard Extension Namespace.

8.12 Standard extension service descriptions

8.12.1 AnonymousStateService

Let d be an `sx:AnonymousStateService`. Then the service described by d is implementation dependent and provides the following functionality.

— With respect to any requested performance, for any ordered tuple ρ and any integer n, it shall not claim that the authorization context property with name sx:eLC(d, ρ) and value n may be used in an authorization context to establish permission for the requested performance unless there are exactly n performances, including the requested performance, for which both of the following are true:

  — the authorization context used to establish permission for that performance has a property named sx:eLC(d, ρ), and

  — the interval of time in which that performance occurs starts before or at the start of the interval of time in which the requested performance would occur.

— For any ordered tuple ρ, any time interval v, and any `r:Principal` p, it shall not claim that, throughout v, p identifies the owner of a virtual token it identifies by ρ if it has already claimed that during some time in v some other Principal is the owner of that same virtual token identified by ρ.

— For any ordered tuple ρ, any time instant τ, and any integer n, it shall not claim that, at τ, n is the track query integral value managed with respect to ρ unless there are exactly n performances, including the requested performance if both of the following are true for it, for which both of the following are true:

  — the authorization context used to establish permission for that performance has a property named sx:tR(d, ρ), and

  — the interval of time in which that performance occurs starts before or at τ.

— For any ordered tuple ρ and any time instant τ, it shall not claim that τ is the determined floating start time with respect to ρ if it has already claimed that some other time instant is the determined floating start time with respect to ρ.

— For any ordered tuple ρ and any time interval v, it shall not claim that v is valid metered time with respect to ρ if making this claim would contradict any claims it has already made about total duration of valid metered time.

— For any ordered tuple ρ, any time instant τ, and any time duration y, it shall not claim that the total duration of time before τ that it has claimed, claims, or will claim as valid metered time with respect to ρ is y if it has already made claims about valid metered time that would result in more total duration than y.

— For any ordered tuple ρ, any `sx:AccountPayable` a, any time instant τ, and any `sx:Rate` ϕ, it shall not claim that, at τ, the amount of money indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for flat fees with respect to ρ if it has already claimed that, at τ, a greater amount of money than that indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for flat fees with respect to ρ.

— For any ordered tuple ρ, any time interval v, any `sx:AccountPayable` a, and any `sx:Rate` ϕ, it shall not claim that, at the start of v, the amount of money indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for v with respect to ρ if it has already claimed that, at the start of v, a greater amount of money than that indicated by ϕ is considered paid using the mechanism for making a payment indicated by a to the account indicated by a for v with respect to ρ.

8.12.2 Uddi

Let d be an `sx:Uddi`. Let r be the Registry named by d/`sx:registry`, if it is present, or the Universal Business Registry, if d/`sx:registry` is not present. Then the description of the service described by d is given by the Business Service indicated by the primary key represented by d/`sx:serviceKey` (according to the semantics of `sx:UddiKey`) within the Registry r.

EXAMPLE
```
<sx:uddi licensePartId="uddiDescription">
   <sx:serviceKey>
      <sx:uuid>D04951E4-332C-4693-B7DB-D3D1D1C21111</sx:uuid>
   </sx:serviceKey>
</sx:uddi>
```

8.12.3 WsdlAddress

Let d be an `sx:WsdlAddress`. Then d/`sx:kind`/`sx:wsdl` shall identify a `wsdl:definitions` element, and the description of the service described by d is given in that `wsdl:definitions` element for the WSDL binding whose name is the value of d/`sx:kind`/`sx:binding`. The endpoint of the service is given by the contents of d/`sx:address`.

EXAMPLE
```
<sx:wsdlAddress xmlns:ts="urn:TrackingService">
   <sx:kind>
```

```
      <sx:wsdl>
         <r:nonSecureIndirect URI="http://services.acme.org/wsdl/trackingService.wsdl"/>
      </sx:wsdl>
      <sx:binding>ts:TrackPrintSoapBinding</sx:binding>
   </sx:kind>
   <sx:address>
      <soap:address location="http://services.acme.org/trackingService"/>
   </sx:address>
</sx:wsdlAddress>
```

8.12.4 WsdlComplete

Let *d* be an `sx:WsdlComplete`. Then *d*/`sx:wsdl` shall identify a `wsdl:definitions` element, and the description of the service described by *d* is given in that `wsdl:definitions` element for the WSDL service whose name is the value of *d*/`sx:service`. If *d*/`sx:portType` is present, it limits the description only to those WSDL ports whose WSDL port type is the one whose name is the value of *d*/`sx:portType`.

EXAMPLE
```
<sx:wsdlComplete xmlns:ts="urn:TrackingService">
   <sx:wsdl>
      <r:nonSecureIndirect URI="http://services.acme.org/wsdl/trackingService.wsdl"/>
   </sx:wsdl>
   <sx:service>ts:TrackPrint</sx:service>
</sx:wsdlComplete>
```

8.13 Standard extension properties – PropertyUri

Let *t* be an `sx:PropertyUri`. Then *t* identifies the property identified by *t*/@`sx:definition`.

EXAMPLE
```
<sx:propertyUri definition="urn:acme:distributor"/>
```

8.14 Standard extension name properties

8.14.1 CommonName

Let *t* be an `sx:CommonName`. Then *t* identifies the common-name indicated by the contents of *t* as specified in ISO/IEC 10021-2.

8.14.2 DnsName

Let *t* be an `sx:DnsName`. Then *t* identifies the domain name with trailing period omitted indicated by the contents of *t* as specified in RFC 1034.

EXAMPLE
```
<sx:dnsName>professor.acme.org</sx:dnsName>
```

8.14.3 EmailName

Let *t* be an `sx:EmailName`. Then *t* identifies the Internet email address indicated by the contents of *t* as specified in RFC 2822.

EXAMPLE
```
<sx:emailName>alice@acme.org</sx:emailName>
```

8.14.4 X509SubjectName

Let *t* be an `sx:X509SubjectName`. Then *t* identifies the subject name that is the contents of *t* as specified in ISO/IEC 9594-8.

EXAMPLE
<sx:x509SubjectName>CN=Alice</sx:x509SubjectName>

9 Multimedia extension

9.1 General

This Clause specifies syntax and semantics specific to multimedia.

9.2 Multimedia extension authorization context properties

Table 5 specifies the authorization context properties relating to Clause 9 and the statements they represent. If a property has the name given in the first column of Table 5 and the value given in the second column of Table 5, then the statement represented by that property is the statement given in the third column of Table 5.

Table 5 — Multimedia extension authorization context properties

Property name	Property value	Statement represented
mx:destination()	p	p is an `r:Principal`, and p identifies the destination repository (according to the semantics of the Right Member of the authorization request) for the requested performance.
mx:didl(t, τ)	d	t is an `mx:DiReference`, d is a `didl:Container`, `didl:Descriptor`, `didl:Item`, `didl:Component`, or `didl:Anchor`; τ is a time instant; and, at τ, d declares the Container, Descriptor, Digital Item, Component, or Fragment identified by t.
mx:helpers()	P	P is a set of `r:Principal` elements, and P is the exact set of elements where both each element identifies one piece of software helping in the requested performance and every piece of software helping in the requested performance is identified by some element in P.
mx:mark(a, b)	true	a is an `r:Resource`, b is an element that describes a mark, and, during the course of the requested performance, the Resource identified by a becomes marked as described by b.
mx:marked(a, b, τ)	true	a is an `r:Resource`, b is an element that describes a mark, τ is a time instant, and, at τ, the Resource identified by a is already marked as described by b.
mx:partOf(a, b, τ)	true	a is an `mx:DiReference`, b is an `mx:DiReference`, τ is a time instant, and, at τ, the Container, Descriptor, Digital Item, Component, or Fragment identified by a is (through any number of levels) a part of the Container, Descriptor, Digital Item, Component, or Fragment identified by b.
mx:rAC()	A	A is a set of resource attributes, and A is the exact set of resource attributes that are changed (according to the semantics of the Right Member of the authorization request) during the course of the requested performance on the Resource identified by the Resource Member of the authorization request.

Table 5 *(continued)*

mx:renderers()	P	P is a set of `r:Principal` elements, and P is the exact set of elements where both each element identifies one Principal that renders some component of the Resource identified by the Resource Member of the authorization request into its directly perceivable representation for the requested performance and every Principal that renders some component of the Resource identified by the Resource Member of the authorization request into its directly perceivable representation for the requested performance is identified by some element in P. EXAMPLE Audio cards and video cards are examples of renderers. NOTE This property is used with render rights.
mx:sigFound(s, τ)	true	s is a `dsig:Signature`, τ is a time instant, and s exists at τ.
mx:sigRefValid(s, f, t, τ)	true	s is a `dsig:Signature`, f is a `dsig:Reference`, t is an `r:Resource`, τ is a time instant, and, at τ, Digital Signature Reference Validation of s succeeds when the data object to be digested for f is obtained (as is called for in XMLDSIG 3.2.1.2.1) using, instead, t.
mx:sigSigValid(s, p, τ)	true	s is a `dsig:Signature`, p is an `r:Principal`, τ is a time instant, and, at τ, Digital Signature Signature Validation of s succeeds when the keying information is obtained (as is called for in XMLDSIG 3.2.2.1) using, instead, p.
mx:source()	p	p is an `r:Principal`, and p identifies the source repository (according to the semantics of the Right Member of the authorization request) for the requested performance.
mx:tXA(d, ρ)	true	d is an `r:ServiceDescription`, ρ is an ordered tuple, and the service described by d claims that this property may be used in an authorization context to establish permission for the requested performance. NOTE It is generally expected that the value of the property named mx:tXA(d, ρ) will be true if the service is utilized to perform one transaction for the purposes of the requested performance.

9.3 General multimedia extension elements and types – resource attribute set definitions

9.3.1 Complement

Let s be an `mx:complement`. Then the set identified by s is the complement (under the universe of all resource attributes) of the set identified by the resource attribute set definition element child of s.

EXAMPLE
```
<mx:complement>
   <mx:set definition="urn:osmaker:fileProperties"/>
</mx:complement>
```

9.3.2 Intersection

Let s be an `mx:intersection`. Then the set identified by s is the intersection of the sets identified by the resource attribute set definition element children of s. s may have no resource attribute set definition element children even if it also has no `r:licensePartIdRef` or `r:varRef` attribute; in this case the set identified by s is the universe of all resource attributes.

EXAMPLE
```
<mx:intersection>
   <mx:set definition="urn:acmepub1:bookProperties"/>
   <mx:set definition="urn:acmepub2:bookProperties"/>
</mx:intersection>
```

9.3.3 Set

Let s be an `mx:set`. Then the set identified by s is the set that is defined by the URI s/@`mx:definition`. If that definition requires any parameters, they shall be provided as the content of s.

EXAMPLE
```
<mx:set definition="urn:osmaker:fileProperties"/>
```

9.3.4 Union

Let s be an `mx:union`. Then the set identified by s is the union of the sets identified by the resource attribute set definition element children of s. s may have no resource attribute set definition element children even if it also has no `r:licensePartIdRef` or `r:varRef` attribute; in this case the set identified by s is the empty set.

EXAMPLE
```
<mx:union>
   <mx:set definition="urn:acmepub1:bookProperties"/>
   <mx:set definition="urn:acmepub2:bookProperties"/>
</mx:union>
```

9.4 Multimedia extension conceptually abstract elements and types

No other conceptually abstract elements and types are defined in the Multimedia Extension Namespace.

9.5 Multimedia extension principals

No other principals are defined in the Multimedia Extension Namespace.

9.6 Multimedia extension rights

9.6.1 Adapt

Let r be an `mx:Adapt`. Then r identifies the act of Adapt as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the SourceOfAdaptation as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, both Σ.mx:source() shall identify the repository that is the PlaceOfAdaptingFrom as defined in ISO/IEC 21000-6 and Σ.mx:destination() shall identify the repository that is the PlaceOfAdaptingTo as defined in ISO/IEC 21000-6.

NOTE 1 An `mx:Adapt` identifies the act of changing transiently an existing Resource to derive a new Resource. With an `mx:Adapt`, two distinct Resources will exist as a result of the process, one of which is the original Resource in unchanged form, and one of which is newly made. Changes can include the addition to and removal of elements of the original Resource, including the embedding of other Resources. Changes can be made temporarily to the original Resource in the course of the adapt process, but such changes are not saved in the original Resource at the end of the process.

NOTE 2 An `mx:Adapt` can be used with Conditions requiring specific resource attributes of the Resource to be preserved or changed. The specific resource attributes can be on a list or can be called out by using a list. Lists can be inclusive (for example, "Attributes α and β must be changed") or exclusive (for example, "Everything except attributes γ and δ must be changed"). Resource attributes that are not otherwise constrained by Conditions can be changed.

9.6.2 Delete

Let *r* be an `mx:Delete`. Then *r* identifies the act of Delete as defined in ISO/IEC 21000-6.

If *r* is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the DeletedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfDeleting as defined in ISO/IEC 21000-6.

NOTE An `mx:Delete` identifies the act of destroying a DigitalResource as defined in ISO/IEC 21000-6. Delete is not capable of reversal. After a delete process, an "undelete" action is impossible.

9.6.3 Diminish

Let *r* be an `mx:Diminish`. Then *r* identifies the act of Diminish as defined in ISO/IEC 21000-6.

If *r* is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the SourceOfDiminution as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, both Σ.mx:source() shall identify the repository that is the PlaceOfDiminishingFrom as defined in ISO/IEC 21000-6 and Σ.mx:destination() shall identify the repository that is the PlaceOfDiminishingTo as defined in ISO/IEC 21000-6.

NOTE An `mx:Diminish` identifies the act of deriving a new Resource which is smaller than its Source as defined in ISO/IEC 21000-6. With an `mx:Diminish`, two distinct Resources will exist at the end of the process, one of which is the original Resource in unchanged form, and one of which is newly made, whose content is adapted from the original Resource, and a measure of which is smaller than that of the original. Changes can include the removal of elements of the original Resource. Changes can be made temporarily to the original Resource in the course of the diminish process, but such changes are not saved in the original Resource at the end of the process.

9.6.4 Embed

Let *r* be an `mx:Embed`. Then *r* identifies the act of Embed as defined in ISO/IEC 21000-6.

If *r* is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the EmbeddedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, both Σ.mx:source() shall identify the repository that is the PlaceOfEmbeddingFrom as defined in ISO/IEC 21000-6 and Σ.mx:destination() shall identify the repository that is the PlaceOfEmbeddingTo as defined in ISO/IEC 21000-6.

NOTE 1 An `mx:Embed` identifies the act of putting a Resource into another Resource. The Resource into which a Resource is embedded can be pre-existing or can be created by the act of combining the embedded Resource with one or more others.

NOTE 2 An `mx:Embed` refers only to the embedding of an existing Resource in another. If a "copy" of an existing Resource is to be created and embedded in another, then both `mx:Adapt` and `mx:Embed` would be used.

9.6.5 Enhance

Let *r* be an `mx:Enhance`. Then *r* identifies the act of Enhance as defined in ISO/IEC 21000-6.

If *r* is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the SourceOfEnhancement as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, both Σ.mx:source() shall identify the repository that is the PlaceOfEnhancingFrom as defined in ISO/IEC 21000-6 and Σ.mx:destination() shall identify the repository that is the PlaceOfEnhancingTo as defined in ISO/IEC 21000-6.

NOTE An `mx:Enhance` identifies the act of deriving a new Resource which is larger than its Source as defined in ISO/IEC 21000-6. With an `mx:Enhance`, two distinct Resources will exist at the end of the process, one of which is the original Resource in unchanged form, and one of which is newly made, whose content is adapted from the original Resource, and a measure of which is smaller than that of the original. Changes can include the addition to elements of the original Resource, including the embedding of other Resources. Changes can be made temporarily to the original Resource in the course of the enhance process, but such changes are not saved in the original Resource at the end of the process.

9.6.6 Enlarge

Let r be an `mx:Enlarge`. Then r identifies the act of Enlarge as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the EnlargedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfEnlarging as defined in ISO/IEC 21000-6.

NOTE An `mx:Enlarge` identifies the act of modifying a Resource by adding to it. With an `mx:Enlarge`, a single Resource is preserved at the end of the process. Changes can include the addition of new material, including the embedding of other Resources, but not the changing or removal of existing elements of the original Resource.

9.6.7 Execute

Let r be an `mx:Execute`. Then r identifies the act of Execute as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the ExecutedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfExecuting as defined in ISO/IEC 21000-6.

NOTE An `mx:Execute` identifies the act of executing a DigitalResource as defined in ISO/IEC 21000-6. An `mx:Execute` refers to the primitive computing process of executing.

9.6.8 Install

Let r be an `mx:Install`. Then r identifies the act of Install as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the InstallingResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request Σ.mx:source() shall identify the repository that is the PlaceOfInstalling as defined in ISO/IEC 21000-6.

NOTE An `mx:Install` identifies the act of following the instructions provided by an InstallingResource as defined in ISO/IEC 21000-6.

9.6.9 Modify

Let r be an `mx:Modify`. Then r identifies the act of Modify as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the ModifiedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfModifying as defined in ISO/IEC 21000-6.

NOTE 1 An `mx:Modify` identifies the act of changing a Resource, preserving the alterations made. With an `mx:Modify`, a single Resource is preserved at the end of the process. Changes can include the addition to and removal of elements of the original Resource, including the embedding of other Resources.

NOTE 2 An `mx:Modify` can be used with Conditions requiring specific resource attributes of the Resource to be preserved or changed. The specific resource attributes can be on a list or can be called out by using a list. Lists can be

inclusive (for example, "Attributes α and β must be changed") or exclusive (for example, "Everything except attributes γ and δ must be changed"). Resource attributes that are not otherwise constrained by Conditions can be changed.

9.6.10 Move

Let r be an `mx:Move`. Then r identifies the act of Move as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the MovedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, both Σ.mx:source() shall identify the repository that is the Origin as defined in ISO/IEC 21000-6 and Σ.mx:destination() shall identify the repository that is the Destination as defined in ISO/IEC 21000-6.

NOTE An `mx:Move` identifies the act of relocating a Resource from one place to another. With an `mx:Move`, at least the location of the Resource is changed.

9.6.11 Play

Let r be an `mx:Play`. Then r identifies the act of Play as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the SourceOfPlaying as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfPlayingFrom as defined in ISO/IEC 21000-6.

NOTE 1 An `mx:Play` identifies the act of deriving a transient and directly perceivable representation of a Resource. An `mx:Play` covers the making of any forms of transient representation that can be perceived directly (that is, without any intermediary process) with at least one of the five human senses. An `mx:Play` includes playing a video or audio clip, displaying an image or text document, or creating transient representations that can be touched, or perceived to be touched. When an `mx:Play` is applied to a Digital Resource, content can be rendered in any order or sequence according to the technical constraints of the Digital Resource and renderer.

NOTE 2 An `mx:Play` is a render right.

9.6.12 Print

Let r be an `mx:Print`. Then r identifies the act of Print as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the SourceOfPrintedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfPrintingFrom as defined in ISO/IEC 21000-6.

NOTE 1 An `mx:Print` identifies the act of deriving a fixed and directly perceivable representation of a Resource. An `mx:Print` refers to the making of a fixed physical representation, such as a hard-copy print of an image or text, that can be perceived directly (that is, without any intermediary process) with one or more of the five human senses.

NOTE 2 An `mx:Print` is a render right.

9.6.13 Reduce

Let r be an `mx:Reduce`. Then r identifies the act of Reduce as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the ReducedResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request, Σ.mx:source() shall identify the repository that is the PlaceOfReducing as defined in ISO/IEC 21000-6.

NOTE An `mx:Reduce` identifies the act of modifying a Resource by taking away from it. With an `mx:Reduce`, a single Resource is preserved at the end of the process. Changes can include only the removal of existing elements of the original Resource.

9.6.14 Uninstall

Let r be an `mx:Uninstall`. Then r identifies the act of Uninstall as defined in ISO/IEC 21000-6.

If r is used as the Right Member of an authorization request, then both the Resource Member of that authorization request shall be present and shall identify the UninstallingResource as defined in ISO/IEC 21000-6 and, letting Σ be the Authorization Context Member of that authorization request Σ.mx:source() shall identify the repository that is the PlaceOfUninstalling as defined in ISO/IEC 21000-6.

NOTE An `mx:Uninstall` identifies the act of following the instructions provided by an UninstallingResource as defined in ISO/IEC 21000-6.

9.7 Multimedia extension resources

9.7.1 DiItemReference

Let t be an `mx:DiItemReference` and i be the value of t/`mx:identifier`. Then t identifies the Digital Item identified by i using ISO/IEC 21000-3 less any sub-Digital Items of that Digital Item.

EXAMPLE
```
<mx:diItemReference>
   <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
</mx:diItemReference>
```

9.7.2 DiReference

Let t be an `mx:DiReference` and i be the value of t/`mx:identifier`. Then t identifies the Container, Descriptor, Digital Item, Component, or Fragment identified by i using ISO/IEC 21000-3.

EXAMPLE
```
<mx:diReference>
   <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
</mx:diReference>
```

9.8 Multimedia extension conditions

9.8.1 Digital Item conditions

9.8.1.1 DiCriteria

Let c be an `mx:DiCriteria`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let τ be the time instant at the start of v. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if a new XML document containing Σ.mx:didl(c/`mx:diReference`, τ) as the root element Matches each of the c/`r:anXmlPatternAbstract` children of c with respect to (p, r, t, v, Σ, L, R) and e.

EXAMPLE
```
<mx:diCriteria>
   <mx:diReference>
      <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
   </mx:diReference>
   <r:anXmlExpression>//didl:Resource/@mimeType[contains(.,"video/mpeg")]</r:anXmlExpressi
on>
</mx:diCriteria>
```

9.8.1.2 DiPartOf

Let c be an `mx:DiPartOf`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be the first child of c. Let b be the second child of c. Let τ be the time instant at the start of v. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.mx:partOf(a, b, τ) is true.

EXAMPLE
```
<mx:diPartOf>
   <mx:diReference>
      <mx:identifier>urn:acme:photographs:logo</mx:identifier>
   </mx:diReference>
   <mx:diReference>
      <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
   </mx:diReference>
</mx:diPartOf>
```

9.8.2 Marking conditions

9.8.2.1 IsMarked

Let c be an `mx:IsMarked`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be the c/`r:resource` child of c. Let b be the other child of c. Let τ be the time instant at the start of v. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.mx:marked(a, b, τ) is true.

EXAMPLE
```
<mx:isMarked>
   <mx:diReference>
      <mx:identifier>urn:acme:photographs:superbowl25</mx:identifier>
   </mx:diReference>
   <acme:acmeWatermark value="photographer=Acme Man"/>
</mx:isMarked>
```

9.8.2.2 Mark

Let c be an `mx:Mark`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let a be the c/`r:resource` child of c. Let b be the other child of c. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.mx:mark(a, b) is true.

EXAMPLE
```
<mx:mark>
   <mx:diReference>
      <mx:identifier>urn:acme:photographs:superbowl25</mx:identifier>
   </mx:diReference>
   <acme:acmeWatermark value="photographer=Acme Man"/>
</mx:mark>
```

9.8.3 Security conditions

9.8.3.1 Destination

Let c be an `mx:Destination`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if c/`r:principal` is Equal to Σ.mx:destination().

EXAMPLE
```
<mx:destination>
   <r:keyHolder licensePartIdRef="destinationRepository"/>
</mx:destination>
```

9.8.3.2 Helper

Let c be an `mx:Helper`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if, for each `r:Principal` π in Σ.mx:helpers(), either π is Equal to one of the c/`r:principal` elements or there is a c/`mx:wildcard` element w such that a new XML document containing π as the root element Matches each of the w/`r:anXmlPatternAbstract` children of w, if any are present.

EXAMPLE
```
<mx:helper>
   <r:keyHolder licensePartIdRef="acmeMediaPlayer"/>
</mx:helper>
```

9.8.3.3 Renderer

Let c be an `mx:Renderer`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if, for each `r:Principal` π in Σ.mx:renderers(), either π is Equal to one of the c/`r:principal` elements or there is a c/`mx:wildcard` element w such that a new XML document containing π as the root element Matches each of the w/`r:anXmlPatternAbstract` children of w, if any are present.

EXAMPLE
```
<mx:renderer>
   <r:keyHolder licensePartIdRef="trustedDevice"/>
</mx:renderer>
```

9.8.3.4 ResourceSignedBy

Let c be an `mx:ResourceSignedBy`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let τ be the time instant at the start of v. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if there exists some `dsig:Signature` s such that all of the following are true:

— Σ.mx:sigFound(s, τ) is true,

— c/`dsig:CanonicalizationMethod` is Equal to s/`dsig:SignedInfo`/`dsig:CanonicalizationMethod`,

— c/`dsig:SignatureMethod` is Equal to s/`dsig:SignedInfo`/`dsig:SignatureMethod`,

— there exists some s/`dsig:SignedInfo`/`dsig:Reference` f such that all of the following are true:

  — c/`dsig:Transforms` is Equal to f/`dsig:Transforms` (or both are absent),

  — c/`dsig:DigestMethod` is Equal to f/`dsig:DigestMethod`, and

  — Σ.mx:sigRefValid(s, f, c/`r:resource`, τ) is true, and

— Σ.mx:sigSigValid(s, c/`r:principal`, τ) is true.

EXAMPLE
```
<mx:resourceSignedBy>
   <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
   <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
   <mx:diReference>
      <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
   </mx:diReference>
   <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
   <r:keyHolder licensePartIdRef="acme"/>
</mx:resourceSignedBy>
```

9.8.3.5 Source

Let c be an `mx:Source`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if c/`r:principal` is Equal to Σ.mx:source().

EXAMPLE
```
<mx:source>
   <r:keyHolder licensePartIdRef="sourceRepository"/>
</mx:source>
```

9.8.4 Transaction

Let c be an `mx:Transaction`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Let m be c/`r:serviceReference`. Let ρ be the ordered tuple containing the values of the reference-specific parameters determined by m. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if Σ.mx:tXA(m/`r:serviceDescription`, ρ) is true.

EXAMPLE
```
<mx:transaction>
   <r:serviceReference>
      <sx:uddi licensePartIdRef="uddiDescription"/>
      <r:serviceParameters>
         <r:datum>
            <mx:diReference>
               <mx:identifier>urn:acme:mpeg-rel-unleashed</mx:identifier>
            </mx:diReference>
         </r:datum>
      </r:serviceParameters>
   </r:serviceReference>
</mx:transaction>
```

9.8.5 Resource attribute conditions

9.8.5.1 ProhibitedAttributeChanges

Let c be an `mx:ProhibitedAttributeChanges`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if for each resource attribute set definition element child s of c there is no resource attribute that is both a member of Σ.mx:rAC() and a member of the set identified by s. c may have no resource attribute set definition element children even if it also has no `r:licensePartIdRef` or `r:varRef` attribute; in this case c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e).

EXAMPLE
```
<mx:prohibitedAttributeChanges>
   <mx:set definition="urn:osmaker:fileProperties"/>
</mx:prohibitedAttributeChanges>
```

9.8.5.2 RequiredAttributeChanges

Let c be an `mx:RequiredAttributeChanges`. Let (p, r, t, v, Σ, L, R) be an authorization request. Let (g, h, e) be an authorization story. Then c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e) if and only if for each resource attribute set definition element child s of c there is no resource attribute that is both a member of the set identified by s and not a member of Σ.mx:rAC(). c may have no resource attribute set definition element children even if it also has no `r:licensePartIdRef` or `r:varRef` attribute; in this case c is Satisfied with respect to (p, r, t, v, Σ, L, R) and (g, h, e).

EXAMPLE

```
<mx:requiredAttributeChanges>
   <mx:set definition="urn:osmaker:fileProperties"/>
</mx:requiredAttributeChanges>
```

9.9 Multimedia extension patterns

No other patterns are defined in the Multimedia Extension Namespace.

9.10 Multimedia extension delegation constraints

No other delegation constraints are defined in the Multimedia Extension Namespace.

9.11 Multimedia extension trust roots

No other trust roots are defined in the Multimedia Extension Namespace.

9.12 Multimedia extension service descriptions

No other service descriptions are defined in the Multimedia Extension Namespace.

9.13 Multimedia extension properties

No other properties are defined in the Multimedia Extension Namespace.

9.14 Multimedia extension name properties

No other name properties are defined in the Multimedia Extension Namespace.

Annex A
(normative)

W3C XML Schemas

A.1 General

This Annex contains a listing of the three schemas (XMLSCHEMA) that define the XML syntax of the types and elements defined throughout this part of ISO/IEC 21000.

A.2 Schema for the Core Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sccns="urn:uddi-org:schemaCentricC14N:2002-07-10"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
   <xsd:import namespace="http://www.w3.org/2001/04/xmlenc#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-
schema.xsd"/>
   <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>
   <xsd:import namespace="urn:uddi-org:schemaCentricC14N:2002-07-10"
schemaLocation="http://www.uddi.org/schema/SchemaCentricCanonicalization.xsd"/>
   <!-- Elements -->
   <xsd:element name="allConditions" type="r:AllConditions"
substitutionGroup="r:condition"/>
   <xsd:element name="allPrincipals" type="r:AllPrincipals"
substitutionGroup="r:principal"/>
   <xsd:element name="anXmlExpression" type="r:AnXmlExpression"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="anXmlPatternAbstract" type="r:AnXmlPatternAbstract"
substitutionGroup="r:resource"/>
   <xsd:element name="condition" type="r:Condition"
substitutionGroup="r:licensePart"/>
   <xsd:element name="conditionIncremental" type="r:ConditionIncremental"
substitutionGroup="r:dcConstraint"/>
   <xsd:element name="conditionPattern" type="r:ConditionPattern"/>
   <xsd:element name="conditionPatternAbstract" type="r:ConditionPatternAbstract"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="datum" type="r:Datum"/>
   <xsd:element name="conditionUnchanged" type="r:ConditionUnchanged"
substitutionGroup="r:dcConstraint"/>
   <xsd:element name="dcConstraint" type="r:DcConstraint"
substitutionGroup="r:licensePart"/>
   <xsd:element name="delegationControl" substitutionGroup="r:licensePart">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart">
               <xsd:sequence>
```

```
                  <xsd:element ref="r:dcConstraint" minOccurs="0"
maxOccurs="unbounded"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="depthConstraint" type="r:DepthConstraint"
substitutionGroup="r:dcConstraint"/>
   <xsd:element name="digitalResource" type="r:DigitalResource"
substitutionGroup="r:resource"/>
   <xsd:element name="exerciseMechanism" type="r:ExerciseMechanism"
substitutionGroup="r:condition"/>
   <xsd:element name="existsRight" type="r:ExistsRight"
substitutionGroup="r:condition"/>
   <xsd:element name="forAll" block="#all" substitutionGroup="r:licensePart"
final="#all">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart">
               <xsd:sequence>
                  <xsd:element ref="r:anXmlPatternAbstract" minOccurs="0"
maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:attribute name="varName" type="r:VariableName"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="fulfiller" type="r:Fulfiller"
substitutionGroup="r:condition"/>
   <xsd:element name="grant" type="r:Grant" substitutionGroup="r:resource"/>
   <xsd:element name="grantGroup" type="r:GrantGroup"
substitutionGroup="r:resource"/>
   <xsd:element name="grantGroupPattern" type="r:GrantGroupPattern"
substitutionGroup="r:resourcePatternAbstract"/>
   <xsd:element name="grantPattern" type="r:GrantPattern"
substitutionGroup="r:resourcePatternAbstract"/>
   <xsd:element name="issue" block="#all" substitutionGroup="r:right"
final="#all">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="issuer" type="r:Issuer"/>
   <xsd:element name="keyHolder" type="r:KeyHolder"
substitutionGroup="r:principal"/>
   <xsd:element name="license" type="r:License"/>
   <xsd:element name="licenseGroup" type="r:LicenseGroup"/>
   <xsd:element name="licensePart" type="r:LicensePart"/>
   <xsd:element name="obtain" type="r:Obtain" substitutionGroup="r:right"/>
   <xsd:element name="patternFromLicensePart" type="r:PatternFromLicensePart"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="possessProperty" type="r:PossessProperty"
substitutionGroup="r:right"/>
   <xsd:element name="prerequisiteRight" type="r:PrerequisiteRight"
substitutionGroup="r:condition"/>
```

```
   <xsd:element name="principal" type="r:Principal"
substitutionGroup="r:resource"/>
   <xsd:element name="principalPattern" type="r:PrincipalPattern"/>
   <xsd:element name="principalPatternAbstract" type="r:PrincipalPatternAbstract"
substitutionGroup="r:resourcePatternAbstract"/>
   <xsd:element name="propertyAbstract" type="r:PropertyAbstract"
substitutionGroup="r:resource"/>
   <xsd:element name="propertyPossessor" type="r:PropertyPossessor"
substitutionGroup="r:principalPatternAbstract"/>
   <xsd:element name="resource" type="r:Resource"
substitutionGroup="r:licensePart"/>
   <xsd:element name="resourcePattern" type="r:ResourcePattern"/>
   <xsd:element name="resourcePatternAbstract" type="r:ResourcePatternAbstract"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="revocable" type="r:Revocable"
substitutionGroup="r:resource"/>
   <xsd:element name="revocationFreshness" type="r:RevocationFreshness"
substitutionGroup="r:condition"/>
   <xsd:element name="revoke" type="r:Revoke" substitutionGroup="r:right"/>
   <xsd:element name="right" type="r:Right" substitutionGroup="r:licensePart"/>
   <xsd:element name="rightPattern" type="r:RightPattern"/>
   <xsd:element name="rightPatternAbstract" type="r:RightPatternAbstract"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="serviceDescription" type="r:ServiceDescription"
substitutionGroup="r:licensePart"/>
   <xsd:element name="serviceReference" type="r:ServiceReference"
substitutionGroup="r:resource"/>
   <xsd:element name="toConstraint" type="r:ToConstraint"
substitutionGroup="r:dcConstraint"/>
   <xsd:element name="trustedRootGrants" type="r:TrustedRootGrants"
substitutionGroup="r:trustRoot"/>
   <xsd:element name="trustedRootIssuers" type="r:TrustedRootIssuers"
substitutionGroup="r:trustRoot"/>
   <xsd:element name="trustRoot" type="r:TrustRoot"
substitutionGroup="r:licensePart"/>
   <xsd:element name="validityInterval" type="r:ValidityInterval"
substitutionGroup="r:condition"/>
   <!--Complex Types-->
   <xsd:complexType name="AllConditions">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence>
               <xsd:element ref="r:condition" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="AllPrincipals">
      <xsd:complexContent>
         <xsd:extension base="r:Principal">
            <xsd:sequence>
               <xsd:element ref="r:principal" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="AnXmlExpression" mixed="true"
sccns:embeddedLangAttribute="r:lang">
```

```
      <xsd:complexContent mixed="true">
         <xsd:extension base="r:AnXmlPatternAbstract">
            <xsd:attribute name="lang" type="xsd:anyURI"
default="http://www.w3.org/TR/1999/REC-xpath-19991116"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="AnXmlPatternAbstract">
      <xsd:complexContent>
         <xsd:extension base="r:Resource"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Condition">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ConditionIncremental">
      <xsd:complexContent>
         <xsd:extension base="r:DcConstraint"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ConditionPattern">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="r:anXmlExpression"/>
         <xsd:element ref="r:conditionPatternAbstract"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="ConditionPatternAbstract">
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Datum">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart">
            <xsd:sequence minOccurs="0">
               <xsd:any namespace="##any" processContents="lax"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ConditionUnchanged">
      <xsd:complexContent>
         <xsd:extension base="r:DcConstraint"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="DcConstraint">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="DepthConstraint">
      <xsd:complexContent>
         <xsd:extension base="r:DcConstraint">
            <xsd:sequence minOccurs="0">
               <xsd:element name="count" type="xsd:int"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
```

```
   </xsd:complexType>
   <xsd:complexType name="DigitalResource">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:choice minOccurs="0">
               <xsd:element name="nonSecureIndirect"
type="r:NonSecureReference"/>
               <xsd:element name="secureIndirect" type="dsig:ReferenceType"/>
               <xsd:element name="binary" type="xsd:base64Binary"/>
               <xsd:element name="anXml">
                  <xsd:complexType mixed="true">
                     <xsd:sequence>
                        <xsd:any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
               <xsd:any namespace="##other" processContents="lax"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="EncryptedContent">
      <xsd:complexContent>
         <xsd:extension base="enc:EncryptedDataType"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ExerciseMechanism">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:choice minOccurs="0">
               <xsd:element name="exerciseService">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element ref="r:serviceReference"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
               <xsd:any namespace="##other" processContents="lax"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ExistsRight">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:choice>
                  <xsd:element ref="r:grant"/>
                  <xsd:element ref="r:grantGroup"/>
               </xsd:choice>
               <xsd:element ref="r:trustRoot" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Fulfiller">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
```

```
               <xsd:element ref="r:principal"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Grant">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:choice minOccurs="0">
               <xsd:sequence>
                  <xsd:element ref="r:forAll" minOccurs="0"
maxOccurs="unbounded"/>
                  <xsd:element ref="r:delegationControl" minOccurs="0"/>
                  <xsd:element ref="r:principal" minOccurs="0"/>
                  <xsd:element ref="r:right"/>
                  <xsd:element ref="r:resource" minOccurs="0"/>
                  <xsd:element ref="r:condition" minOccurs="0"/>
               </xsd:sequence>
               <xsd:element name="encryptedGrant" type="r:EncryptedContent"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="GrantGroup">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:choice minOccurs="0">
               <xsd:sequence>
                  <xsd:element ref="r:forAll" minOccurs="0"
maxOccurs="unbounded"/>
                  <xsd:element ref="r:delegationControl" minOccurs="0"/>
                  <xsd:element ref="r:principal" minOccurs="0"/>
                  <xsd:element ref="r:condition" minOccurs="0"/>
                  <xsd:choice maxOccurs="unbounded">
                     <xsd:element ref="r:grant"/>
                     <xsd:element ref="r:grantGroup"/>
                  </xsd:choice>
               </xsd:sequence>
               <xsd:element name="encryptedGrantGroup"
type="r:EncryptedContent"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="GrantGroupPattern">
      <xsd:complexContent>
         <xsd:extension base="r:ResourcePatternAbstract">
            <xsd:sequence minOccurs="0">
               <xsd:choice minOccurs="0">
                  <xsd:element ref="r:principal"/>
                  <xsd:element ref="r:principalPattern"/>
               </xsd:choice>
               <xsd:choice minOccurs="0">
                  <xsd:element ref="r:condition"/>
                  <xsd:element ref="r:conditionPattern"/>
               </xsd:choice>
               <xsd:choice maxOccurs="unbounded">
                  <xsd:choice>
                     <xsd:element ref="r:grant"/>
                     <xsd:element ref="r:grantPattern"/>
```

```
                  </xsd:choice>
                  <xsd:choice>
                     <xsd:element ref="r:grantGroup"/>
                     <xsd:element ref="r:grantGroupPattern"/>
                  </xsd:choice>
               </xsd:choice>
               <xsd:element name="wholeGrantGroupExpression"
type="r:AnXmlExpression" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="GrantPattern">
      <xsd:complexContent>
         <xsd:extension base="r:ResourcePatternAbstract">
            <xsd:sequence minOccurs="0">
               <xsd:choice minOccurs="0">
                  <xsd:element ref="r:principal"/>
                  <xsd:element ref="r:principalPattern"/>
               </xsd:choice>
               <xsd:choice>
                  <xsd:element ref="r:right"/>
                  <xsd:element ref="r:rightPattern"/>
               </xsd:choice>
               <xsd:choice minOccurs="0">
                  <xsd:element ref="r:resource"/>
                  <xsd:element ref="r:resourcePattern"/>
               </xsd:choice>
               <xsd:choice minOccurs="0">
                  <xsd:element ref="r:condition"/>
                  <xsd:element ref="r:conditionPattern"/>
               </xsd:choice>
               <xsd:element name="wholeGrantExpression" type="r:AnXmlExpression"
minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Inventory">
      <xsd:sequence>
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="r:licensePart"/>
         </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Issuer">
      <xsd:sequence>
         <xsd:choice minOccurs="0">
            <xsd:element ref="dsig:Signature"/>
            <xsd:element ref="r:principal"/>
         </xsd:choice>
         <xsd:element name="details" type="r:IssuerDetails" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="IssuerDetails">
      <xsd:sequence>
         <xsd:element name="timeOfIssue" type="xsd:dateTime" minOccurs="0"/>
         <xsd:element name="revocationMechanism" type="r:RevocationMechanism"
minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
```

```
   </xsd:complexType>
   <xsd:complexType name="KeyHolder">
      <xsd:complexContent>
         <xsd:extension base="r:Principal">
            <xsd:sequence minOccurs="0">
               <xsd:element name="info" type="dsig:KeyInfoType"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="License">
      <xsd:choice>
         <xsd:sequence>
            <xsd:element name="title" type="r:LinguisticString" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="inventory" type="r:Inventory" minOccurs="0"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element ref="r:grant"/>
               <xsd:element ref="r:grantGroup"/>
            </xsd:choice>
            <xsd:element ref="r:issuer" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="otherInfo" minOccurs="0">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:any namespace="##any" processContents="lax"
maxOccurs="unbounded"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
         <xsd:element name="encryptedLicense" type="r:EncryptedContent"/>
      </xsd:choice>
      <xsd:attribute name="licenseId" type="xsd:anyURI" use="optional"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
   </xsd:complexType>
   <xsd:complexType name="LicenseGroup">
      <xsd:sequence>
         <xsd:element ref="r:license" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="LicensePart">
      <xsd:attribute name="licensePartId" type="r:LicensePartId" use="optional"/>
      <xsd:attribute name="licensePartIdRef" type="r:LicensePartId"
use="optional"/>
      <xsd:attribute name="varRef" type="r:VariableName" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="LinguisticString" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:restriction base="xsd:anyType">
            <xsd:sequence>
               <xsd:any processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute ref="xml:lang"/>
         </xsd:restriction>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="NonSecureReference">
      <xsd:sequence>
         <xsd:element ref="dsig:Transforms" minOccurs="0"/>
```

```
   </xsd:sequence>
   <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
   <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
   <xsd:attribute name="Type" type="xsd:anyURI" use="optional"/>
</xsd:complexType>
<xsd:complexType name="Obtain">
   <xsd:complexContent>
      <xsd:extension base="r:Right"/>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PatternFromLicensePart">
   <xsd:complexContent>
      <xsd:extension base="r:AnXmlPatternAbstract">
         <xsd:sequence minOccurs="0">
            <xsd:element ref="r:licensePart"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PossessProperty">
   <xsd:complexContent>
      <xsd:extension base="r:Right"/>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PrerequisiteRight">
   <xsd:complexContent>
      <xsd:extension base="r:Condition">
         <xsd:sequence minOccurs="0">
            <xsd:element ref="r:principal" minOccurs="0"/>
            <xsd:element ref="r:right"/>
            <xsd:element ref="r:resource" minOccurs="0"/>
            <xsd:element ref="r:trustRoot" minOccurs="0"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Principal">
   <xsd:complexContent>
      <xsd:extension base="r:Resource"/>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PrincipalPattern">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="r:anXmlExpression"/>
      <xsd:element ref="r:principalPatternAbstract"/>
   </xsd:choice>
</xsd:complexType>
<xsd:complexType name="PrincipalPatternAbstract">
   <xsd:complexContent>
      <xsd:extension base="r:ResourcePatternAbstract"/>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PropertyAbstract">
   <xsd:complexContent>
      <xsd:extension base="r:Resource"/>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PropertyPossessor">
   <xsd:complexContent>
      <xsd:extension base="r:PrincipalPatternAbstract">
```

```
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:propertyAbstract"/>
               <xsd:element ref="r:trustRoot" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Resource">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ResourcePattern">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="r:anXmlExpression"/>
         <xsd:element ref="r:resourcePatternAbstract"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="ResourcePatternAbstract">
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Revocable">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:choice minOccurs="0">
               <xsd:element ref="dsig:SignatureValue"/>
               <xsd:element ref="dsig:Reference"/>
               <xsd:sequence>
                  <xsd:element name="licenseId" type="xsd:anyURI"/>
                  <xsd:element ref="r:principal"/>
               </xsd:sequence>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RevocationFreshness">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element name="priorToStart" type="xsd:duration"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RevocationMechanism">
      <xsd:choice>
         <xsd:element name="revocationService">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="r:serviceReference"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="Revoke">
      <xsd:complexContent>
```

```
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Right">
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="RightPattern">
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="r:anXmlExpression"/>
            <xsd:element ref="r:rightPatternAbstract"/>
         </xsd:choice>
      </xsd:complexType>
      <xsd:complexType name="RightPatternAbstract">
         <xsd:complexContent>
            <xsd:extension base="r:AnXmlPatternAbstract"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="ServiceDescription">
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="ServiceReference">
         <xsd:complexContent>
            <xsd:extension base="r:Resource">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="r:serviceDescription"/>
                  <xsd:element name="serviceParameters" minOccurs="0">
                     <xsd:complexType>
                        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                           <xsd:element ref="r:datum"/>
                           <xsd:element name="transforms" type="dsig:TransformsType"
minOccurs="0"/>
                        </xsd:sequence>
                     </xsd:complexType>
                  </xsd:element>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="ToConstraint">
         <xsd:complexContent>
            <xsd:extension base="r:DcConstraint">
               <xsd:sequence>
                  <xsd:element ref="r:forAll" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element ref="r:principal" minOccurs="0"
maxOccurs="unbounded"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="TrustedRootGrants">
         <xsd:complexContent>
            <xsd:extension base="r:TrustRoot">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="r:grant" maxOccurs="unbounded"/>
               </xsd:sequence>
            </xsd:extension>
```

```
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="TrustedRootIssuers">
      <xsd:complexContent>
         <xsd:extension base="r:TrustRoot">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:principal" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="TrustRoot">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ValidityInterval">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence>
               <xsd:element name="notBefore" type="xsd:dateTime" minOccurs="0"/>
               <xsd:element name="notAfter" type="xsd:dateTime" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <!-- Simple Types-->
   <xsd:simpleType name="LicensePartId">
      <xsd:restriction base="xsd:NCName"/>
   </xsd:simpleType>
   <xsd:simpleType name="VariableName">
      <xsd:restriction base="xsd:NCName"/>
   </xsd:simpleType>
</xsd:schema>
```

A.3 Schema for the Standard Extension Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:import namespace="urn:mpeg:mpeg21:2003:01-REL-R-NS" schemaLocation="rel-
r.xsd"/>
   <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>
   <!-- Elements -->
   <xsd:element name="anonymousStateService" type="sx:AnonymousStateService"
substitutionGroup="r:serviceDescription"/>
   <xsd:element name="callForCondition" type="sx:CallForCondition"
substitutionGroup="r:condition"/>
   <xsd:element name="commonName" type="sx:CommonName"
substitutionGroup="sx:name"/>
   <xsd:element name="dnsName" type="sx:DnsName" substitutionGroup="sx:name"/>
   <xsd:element name="emailName" type="sx:EmailName"
substitutionGroup="sx:name"/>
```

```
   <xsd:element name="exerciseLimit" type="sx:ExerciseLimit"
substitutionGroup="r:condition"/>
   <xsd:element name="feeFlat" type="sx:FeeFlat"
substitutionGroup="r:condition"/>
   <xsd:element name="feeMetered" type="sx:FeeMetered"
substitutionGroup="r:condition"/>
   <xsd:element name="feePerInterval" type="sx:FeePerInterval"
substitutionGroup="r:condition"/>
   <xsd:element name="feePerUse" type="sx:FeePerUse"
substitutionGroup="r:condition"/>
   <xsd:element name="feePerUsePrePay" type="sx:FeePerUsePrePay"
substitutionGroup="r:condition"/>
   <xsd:element name="licenseIdPattern" type="sx:LicenseIdPattern"
substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="name" type="sx:Name"
substitutionGroup="r:propertyAbstract"/>
   <xsd:element name="propertyUri" type="sx:PropertyUri"
substitutionGroup="r:propertyAbstract"/>
   <xsd:element name="rate" type="sx:Rate" substitutionGroup="r:licensePart"/>
   <xsd:element name="rightUri" type="sx:RightUri" substitutionGroup="r:right"/>
   <xsd:element name="seekApproval" type="sx:SeekApproval"
substitutionGroup="r:condition"/>
   <xsd:element name="stateDistinguisher" block="#all"
substitutionGroup="r:licensePart" final="#all">
      <xsd:complexType mixed="true">
         <xsd:complexContent mixed="true">
            <xsd:extension base="r:LicensePart"/>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="stateReferenceValuePattern"
type="sx:StateReferenceValuePattern" substitutionGroup="r:anXmlPatternAbstract"/>
   <xsd:element name="territory" type="sx:Territory"
substitutionGroup="r:condition"/>
   <xsd:element name="trackQuery" type="sx:TrackQuery"
substitutionGroup="r:condition"/>
   <xsd:element name="trackReport" type="sx:TrackReport"
substitutionGroup="r:condition"/>
   <xsd:element name="transferControl" type="sx:TransferControl"
substitutionGroup="r:condition"/>
   <xsd:element name="uddi" type="sx:Uddi"
substitutionGroup="r:serviceDescription"/>
   <xsd:element name="validityIntervalDurationPattern"
type="sx:ValidityIntervalDurationPattern"
substitutionGroup="r:conditionPatternAbstract"/>
   <xsd:element name="validityIntervalFloating"
type="sx:ValidityIntervalFloating" substitutionGroup="r:condition"/>
   <xsd:element name="validityIntervalStartsNow"
type="sx:ValidityIntervalStartsNow" substitutionGroup="r:condition"/>
   <xsd:element name="validityTimeMetered" type="sx:ValidityTimeMetered"
substitutionGroup="r:condition"/>
   <xsd:element name="validityTimePeriodic" type="sx:ValidityTimePeriodic"
substitutionGroup="r:condition"/>
   <xsd:element name="wsdlAddress" type="sx:WsdlAddress"
substitutionGroup="r:serviceDescription"/>
   <xsd:element name="wsdlComplete" type="sx:WsdlComplete"
substitutionGroup="r:serviceDescription"/>
   <xsd:element name="x509SubjectName" type="sx:X509SubjectName"
substitutionGroup="sx:name"/>
```

```
   <xsd:element name="x509SubjectNamePattern" type="sx:X509SubjectNamePattern"
substitutionGroup="r:resourcePatternAbstract"/>
   <!-- Complex Types -->
   <xsd:complexType name="AccountPayable">
      <xsd:choice>
         <xsd:element name="paymentService">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="r:serviceReference"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="aba">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="institution">
                     <xsd:simpleType>
                        <xsd:restriction base="xsd:integer">
                           <xsd:totalDigits value="9"/>
                        </xsd:restriction>
                     </xsd:simpleType>
                  </xsd:element>
                  <xsd:element name="account" type="xsd:integer"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="AnonymousStateService">
      <xsd:complexContent>
         <xsd:extension base="r:ServiceDescription"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="CallForCondition">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:serviceReference" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="CommonName" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:extension base="sx:Name"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="DnsName" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:extension base="sx:Name"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="EmailName" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:extension base="sx:Name"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ExerciseLimit">
      <xsd:complexContent>
```

```
            <xsd:extension base="sx:StatefulCondition">
               <xsd:sequence>
                  <xsd:element name="count" type="xsd:integer" minOccurs="0"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="FeeFlat">
         <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="sx:rate"/>
                  <xsd:element name="to" type="sx:AccountPayable" minOccurs="0"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="FeeMetered">
         <xsd:complexContent>
            <xsd:extension base="r:Condition">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="sx:rate"/>
                  <xsd:element name="per" type="xsd:duration"/>
                  <xsd:element name="by" type="xsd:duration"/>
                  <xsd:element name="phase" type="xsd:duration"/>
                  <xsd:element name="to" type="sx:AccountPayable" minOccurs="0"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="FeePerInterval">
         <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="sx:rate"/>
                  <xsd:element name="per" type="xsd:duration"/>
                  <xsd:element name="to" type="sx:AccountPayable" minOccurs="0"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="FeePerUse">
         <xsd:complexContent>
            <xsd:extension base="r:Condition">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="sx:rate"/>
                  <xsd:element name="to" type="sx:AccountPayable" minOccurs="0"/>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="FeePerUsePrePay">
         <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition">
               <xsd:sequence minOccurs="0">
                  <xsd:element ref="sx:rate"/>
                  <xsd:element name="initialNumberOfUses" type="xsd:integer"
minOccurs="0"/>
                  <xsd:element name="to" type="sx:AccountPayable" minOccurs="0"/>
               </xsd:sequence>
```

```
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="LicenseIdPattern">
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Name">
      <xsd:complexContent>
         <xsd:extension base="r:PropertyAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PropertyUri">
      <xsd:complexContent>
         <xsd:extension base="r:PropertyAbstract">
            <xsd:attribute name="definition" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Rate">
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart">
            <xsd:sequence minOccurs="0">
               <xsd:element name="amount" type="xsd:float"/>
               <xsd:element name="currency" type="xsd:QName" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RightUri">
      <xsd:complexContent>
         <xsd:extension base="r:Right">
            <xsd:attribute name="definition" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="SeekApproval">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="StatefulCondition">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence>
               <xsd:element ref="r:serviceReference" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="StateReferenceValuePattern">
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract">
            <xsd:sequence minOccurs="0">
               <xsd:any namespace="##any" processContents="lax"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
```

```
   </xsd:complexType>
   <xsd:complexType name="Territory">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="location">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="country" type="xsd:QName"
minOccurs="0"/>
                        <xsd:element name="region" type="xsd:QName"
minOccurs="0"/>
                        <xsd:element name="state" type="xsd:string"
minOccurs="0"/>
                        <xsd:element name="city" type="xsd:string"
minOccurs="0"/>
                        <xsd:element name="postalCode" type="xsd:string"
minOccurs="0"/>
                        <xsd:element name="street" type="xsd:string"
minOccurs="0"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
               <xsd:element name="domain">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="uri" type="xsd:anyURI"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="TrackQuery">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition">
            <xsd:sequence>
               <xsd:element name="notLessThan" type="xsd:integer" minOccurs="0"/>
               <xsd:element name="notMoreThan" type="xsd:integer" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="TrackReport">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition">
            <xsd:sequence>
               <xsd:element name="communicationFailurePolicy" default="required"
minOccurs="0">
                  <xsd:simpleType>
                     <xsd:restriction base="xsd:NMTOKEN">
                        <xsd:pattern value="lax"/>
                        <xsd:pattern value="required"/>
                     </xsd:restriction>
                  </xsd:simpleType>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
```

```
   </xsd:complexType>
   <xsd:complexType name="TransferControl">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Uddi">
      <xsd:complexContent>
         <xsd:extension base="r:ServiceDescription">
            <xsd:sequence minOccurs="0">
               <xsd:element name="serviceKey" type="sx:UddiKey"/>
               <xsd:element name="registry" type="xsd:anyURI" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="UddiKey">
      <xsd:choice>
         <xsd:element name="uuid" type="sx:Uuid"/>
         <xsd:element name="uri" type="xsd:anyURI"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="ValidityIntervalDurationPattern">
      <xsd:complexContent>
         <xsd:extension base="r:ConditionPatternAbstract">
            <xsd:sequence minOccurs="0">
               <xsd:element name="duration" type="xsd:duration"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ValidityIntervalFloating">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition">
            <xsd:sequence>
               <xsd:element name="duration" type="xsd:duration" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ValidityIntervalStartsNow">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:validityInterval"/>
               <xsd:element name="backwardTolerance" type="xsd:duration"
minOccurs="0"/>
               <xsd:element name="forwardTolerance" type="xsd:duration"
minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ValidityTimeMetered">
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition">
            <xsd:sequence>
               <xsd:element name="duration" type="xsd:duration" minOccurs="0"/>
               <xsd:element name="quantum" type="xsd:duration" minOccurs="0"/>
            </xsd:sequence>
```

```
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ValidityTimePeriodic">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element name="start" type="xsd:dateTime"/>
               <xsd:element name="period" type="xsd:duration"/>
               <xsd:element name="phase" type="xsd:duration" minOccurs="0"/>
               <xsd:element name="duration" type="xsd:duration"/>
               <xsd:element name="periodCount" type="xsd:nonNegativeInteger"
minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="WsdlAddress">
      <xsd:complexContent>
         <xsd:extension base="r:ServiceDescription">
            <xsd:sequence minOccurs="0">
               <xsd:element name="kind">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="wsdl" type="r:DigitalResource"/>
                        <xsd:element name="binding" type="xsd:QName"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
               <xsd:element name="address">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:any namespace="##any" processContents="lax"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="WsdlComplete">
      <xsd:complexContent>
         <xsd:extension base="r:ServiceDescription">
            <xsd:sequence minOccurs="0">
               <xsd:element name="wsdl" type="r:DigitalResource"/>
               <xsd:element name="service" type="xsd:QName"/>
               <xsd:element name="portType" type="xsd:QName" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="X509SubjectName" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:extension base="sx:Name"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="X509SubjectNamePattern" mixed="true">
      <xsd:complexContent mixed="true">
         <xsd:extension base="r:ResourcePatternAbstract"/>
      </xsd:complexContent>
```

```
   </xsd:complexType>
   <!-- Simple Types -->
   <xsd:simpleType name="ProfileCompliance">
      <xsd:list itemType="xsd:QName"/>
   </xsd:simpleType>
   <xsd:simpleType name="Uuid">
      <xsd:restriction base="xsd:string">
         <xsd:length value="36"/>
      </xsd:restriction>
   </xsd:simpleType>
   <!-- Attributes -->
   <xsd:attribute name="profileCompliance" type="sx:ProfileCompliance"/>
</xsd:schema>
```

A.4 Schema for the Multimedia Extension Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:import namespace="urn:mpeg:mpeg21:2003:01-REL-SX-NS" schemaLocation="rel-
sx.xsd"/>
   <xsd:import namespace="urn:mpeg:mpeg21:2003:01-REL-R-NS" schemaLocation="rel-
r.xsd"/>
   <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>
   <!-- -->
   <!-- === Rights === -->
   <!-- -->
   <xsd:complexType name="Modify">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Enlarge">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Reduce">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Move">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Adapt">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Diminish">
      <xsd:complexContent>
```

```
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Enhance">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Embed">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Play">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Print">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Execute">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Install">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Uninstall">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Delete">
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:element name="modify" type="mx:Modify" substitutionGroup="r:right"/>
      <xsd:element name="enlarge" type="mx:Enlarge" substitutionGroup="r:right"/>
      <xsd:element name="reduce" type="mx:Reduce" substitutionGroup="r:right"/>
      <xsd:element name="move" type="mx:Move" substitutionGroup="r:right"/>
      <xsd:element name="adapt" type="mx:Adapt" substitutionGroup="r:right"/>
      <xsd:element name="diminish" type="mx:Diminish" substitutionGroup="r:right"/>
      <xsd:element name="enhance" type="mx:Enhance" substitutionGroup="r:right"/>
      <xsd:element name="embed" type="mx:Embed" substitutionGroup="r:right"/>
      <xsd:element name="play" type="mx:Play" substitutionGroup="r:right"/>
      <xsd:element name="print" type="mx:Print" substitutionGroup="r:right"/>
      <xsd:element name="execute" type="mx:Execute" substitutionGroup="r:right"/>
      <xsd:element name="install" type="mx:Install" substitutionGroup="r:right"/>
      <xsd:element name="uninstall" type="mx:Uninstall"
substitutionGroup="r:right"/>
      <xsd:element name="delete" type="mx:Delete" substitutionGroup="r:right"/>
      <!-- -->
      <!-- === Resources === -->
```

```
   <!-- -->
   <!-- Digital Item Resources -->
   <xsd:complexType name="DiItemReference">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:sequence minOccurs="0">
               <xsd:element name="identifier" type="xsd:anyURI"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="DiReference">
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:sequence minOccurs="0">
               <xsd:element name="identifier" type="xsd:anyURI"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="diItemReference" type="mx:DiItemReference"
substitutionGroup="r:resource"/>
   <xsd:element name="diReference" type="mx:DiReference"
substitutionGroup="r:resource"/>
   <!-- -->
   <!-- === Conditions === -->
   <!-- -->
   <!-- Digital Item Conditions -->
   <xsd:complexType name="DiCriteria">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="mx:diReference"/>
               <xsd:element ref="r:anXmlPatternAbstract" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="DiPartOf">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="mx:diReference"/>
               <xsd:element ref="mx:diReference"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="diCriteria" type="mx:DiCriteria"
substitutionGroup="r:condition"/>
   <xsd:element name="diPartOf" type="mx:DiPartOf"
substitutionGroup="r:condition"/>
   <!-- Marking Conditions -->
   <xsd:complexType name="IsMarked">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:resource"/>
               <xsd:any namespace="##any" processContents="lax"/>
            </xsd:sequence>
```

```
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Mark">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:resource"/>
               <xsd:any namespace="##any" processContents="lax"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="isMarked" type="mx:IsMarked"
substitutionGroup="r:condition"/>
   <xsd:element name="mark" type="mx:Mark" substitutionGroup="r:condition"/>
   <!-- Security Conditions -->
   <xsd:complexType name="Source">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:principal"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Destination">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:principal"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Helper">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence>
               <xsd:element ref="r:principal" minOccurs="0"
maxOccurs="unbounded"/>
               <xsd:element name="wildcard" minOccurs="0" maxOccurs="unbounded">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element ref="r:anXmlPatternAbstract" minOccurs="0"
maxOccurs="unbounded"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Renderer">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence>
               <xsd:element ref="r:principal" minOccurs="0"
maxOccurs="unbounded"/>
               <xsd:element name="wildcard" minOccurs="0" maxOccurs="unbounded">
```

```
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element ref="r:anXmlPatternAbstract" minOccurs="0"
maxOccurs="unbounded"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ResourceSignedBy">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="dsig:CanonicalizationMethod"/>
               <xsd:element ref="dsig:SignatureMethod"/>
               <xsd:element ref="r:resource"/>
               <xsd:element ref="dsig:Transforms" minOccurs="0"/>
               <xsd:element ref="dsig:DigestMethod"/>
               <xsd:element ref="r:principal"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="source" type="mx:Source" substitutionGroup="r:condition"/>
   <xsd:element name="destination" type="mx:Destination"
substitutionGroup="r:condition"/>
   <xsd:element name="helper" type="mx:Helper" substitutionGroup="r:condition"/>
   <xsd:element name="renderer" type="mx:Renderer"
substitutionGroup="r:condition"/>
   <xsd:element name="resourceSignedBy" type="mx:ResourceSignedBy"
substitutionGroup="r:condition"/>
   <!-- Transactional Conditions -->
   <xsd:complexType name="Transaction">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:serviceReference"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="transaction" type="mx:Transaction"
substitutionGroup="r:condition"/>
   <!-- Resource Attribute Conditions -->
   <xsd:complexType name="RequiredAttributeChanges">
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element ref="mx:complement"/>
               <xsd:element ref="mx:intersection"/>
               <xsd:element ref="mx:set"/>
               <xsd:element ref="mx:union"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ProhibitedAttributeChanges">
      <xsd:complexContent>
```

```
         <xsd:extension base="r:Condition">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element ref="mx:complement"/>
               <xsd:element ref="mx:intersection"/>
               <xsd:element ref="mx:set"/>
               <xsd:element ref="mx:union"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="requiredAttributeChanges"
type="mx:RequiredAttributeChanges" substitutionGroup="r:condition"/>
   <xsd:element name="prohibitedAttributeChanges"
type="mx:ProhibitedAttributeChanges" substitutionGroup="r:condition"/>
   <!-- Resource Attribute Set Definitions -->
   <xsd:element name="complement" substitutionGroup="r:licensePart">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart">
               <xsd:choice minOccurs="0">
                  <xsd:element ref="mx:complement"/>
                  <xsd:element ref="mx:intersection"/>
                  <xsd:element ref="mx:set"/>
                  <xsd:element ref="mx:union"/>
               </xsd:choice>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="intersection" substitutionGroup="r:licensePart">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart">
               <xsd:choice minOccurs="0" maxOccurs="unbounded">
                  <xsd:element ref="mx:complement"/>
                  <xsd:element ref="mx:intersection"/>
                  <xsd:element ref="mx:set"/>
                  <xsd:element ref="mx:union"/>
               </xsd:choice>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="set" substitutionGroup="r:licensePart">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="r:LicensePart">
               <xsd:sequence>
                  <xsd:any namespace="##any" minOccurs="0"
maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:attribute name="definition" type="xsd:anyURI"
use="optional"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="union" substitutionGroup="r:licensePart">
      <xsd:complexType>
         <xsd:complexContent>
```

```
            <xsd:extension base="r:LicensePart">
               <xsd:choice minOccurs="0" maxOccurs="unbounded">
                  <xsd:element ref="mx:complement"/>
                  <xsd:element ref="mx:intersection"/>
                  <xsd:element ref="mx:set"/>
                  <xsd:element ref="mx:union"/>
               </xsd:choice>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Annex B
(normative)

Country, region, and currency Qualified Names

B.1 Namespace URI structure

The namespaces of the Qualified Names defined in this Annex shall be identified by a URI of the form `urn:mpeg:mpeg21:2003:01-REL-SX-NS:`*`YYYY`*`:`*`TYPE`*, where *`YYYY`* indicates any year after or including 2003 and *`TYPE`*, which is either `country`, `region`, or `currency`, indicates the type of Qualified Names the namespace contains.

NOTE In order to support the consistent meaning of Licenses over time, countries, regions, and currencies are indicated by Qualified Names. Utilizing the facilities provided by ISO 3166 and ISO 4217, this Annex defines an algorithm for generating a number of such Qualified Names, grouped into a number of namespaces.

B.2 Country Qualified Names

The local part of the country Qualified Names defined in this Annex is the ISO 3166-1 country code.

The country namespace `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:country` contains one Qualified Name for each ISO 3166-1 country code in use during 2003. The other country namespaces defined in this Annex contain one Qualified Name for each ISO 3166-1 country code that appears in an ISO 3166-1 publication during the respective year.

The meaning of any country Qualified Name q defined in this Annex is the same as the meaning that the ISO 3166-1 country code corresponding to the local part of q had during the year corresponding to the namespace of q.

Qualified names from the `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:country` namespace should be used to refer to a country whenever possible. In the event it is not possible to use a Qualified Name from this namespace, the Qualified Name from the namespace with the earliest *`YYYY`* value and still referring to the desired country should be used instead. If neither of these is possible, any Qualified Name referring to the desired country may be used.

B.3 Region Qualified Names

The local part of the region Qualified Names defined in this Annex is the ISO 3166-2 region code.

The region namespace `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:region` contains one Qualified Name for each ISO 3166-2 region code in use during 2003. The other region namespaces defined in this Annex contain one Qualified Name for each ISO 3166-2 region code that appears in an ISO 3166-2 publication during the respective year.

The meaning of any region Qualified Name q defined in this Annex is the same as the meaning that the ISO 3166-2 region code corresponding to the local part of q had during the year corresponding to the namespace of q.

Qualified names from the `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:region` namespace should be used to refer to a region whenever possible. In the event it is not possible to use a Qualified Name from this namespace, the Qualified Name from the namespace with the earliest *`YYYY`* value and still referring to the desired region should be used instead. If neither of these is possible, any Qualified Name referring to the desired region may be used.

B.4 Currency Qualified Names

The local part of the currency Qualified Names defined in this Annex is the ISO 4217 currency code.

The currency namespace `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency` contains one Qualified Name for each ISO 4217 currency code in use during 2003. The other currency namespaces defined in this Annex contain one Qualified Name for each ISO 4217 currency code that appears in an ISO 4217 publication during the respective year.

The meaning of any currency Qualified Name q defined in this Annex is the same as the meaning that the ISO 4217 currency code corresponding to the local part of q had during the year corresponding to the namespace of q.

Qualified names from the `urn:mpeg:mpeg21:2003:01-REL-SX-NS:2003:currency` namespace should be used to refer to a currency whenever possible. In the event it is not possible to use a Qualified Name from this namespace, the Qualified Name from the namespace with the earliest *YYYY* value and still referring to the desired currency should be used instead. If neither of these is possible, any Qualified Name referring to the desired currency may be used.

Annex C
(informative)

Simplified equality algorithm

C.1 General

There are a number of techniques that can be used to efficiently test for equality without having to actually invoke the full Schema Centric Canonicalization algorithm. If the schemas in question are relatively fixed so that their structure can be compiled into or otherwise cached by an application, then the assessment of whether two pieces of XML are Equal or not is straightforward to implement efficiently and might very naturally be carried out at the application object model layer. However, if the schemas involved are not so intimately known, then the task of assessing equality is much more complicated and subtle: considerable flexibility and latitude exists in XML Schema wherein possibly quite different XML Information Sets are considered to actually convey the same information. Fortunately, in many of the common cases, it is possible to determine whether two pieces of XML are Equal or not without retrieving and processing the associated schemas. This leads to the possibility to develop a number of auxiliary algorithms, likely differing in exactly which set of common cases they cover, that result in Equal or not Equal for the cases they cover and indeterminate for the cases they do not cover. One of these possible auxiliary algorithms is presented here (embodied in the equalQuickItem function defined below) in the belief that it will be of broad general utility. Note, however, that the presentation of this algorithm is not intended to have any normative impact. Applications are not required to support the same set of common cases supported by this algorithm. If this algorithm differs in its determination of Equal from the normative definition of Equal in 6.1, the normative definition prevails.

C.2 The equalQuickItem function

The equalQuickItem function takes two information items, α and β, as inputs and yields either the result Equal, not Equal, or indeterminate according to whether it determines that the information items can be determined to be Equal or not or that an evaluation by a more comprehensive algorithm is necessary. Let the notation $x[y]$ be understood to represent the value of the property whose name is y of the information item x. Then the equalQuickItem function is defined as follows.

If α and β are different kinds of information items, then not Equal is returned.

If α and β are both **element** information items, then the following steps are considered in order:

a) If α[namespace name] is not identical to β[namespace name], then not Equal is returned.

b) If α[local name] is not identical to β[local name], then not Equal is returned.

c) The sets α[attributes] and β[attributes] are examined to define the value attributesIdentical(α, β):

   1) If a permutation p of β[attributes] exists such that equalQuickList(α[attributes], p) is Equal, then attributesIdentical(α, β) is Equal.

   2) Otherwise, if α[attributes] contains a member a and β[attributes] contains a member b where both a[namespace name] is identical to b[namespace name] and a[local name] is identical to b[local name] then, if equalQuickItem(a, b) is not Equal or indeterminate, then attributesIdentical(α, β) is not Equal or indeterminate, respectively.

   3) Otherwise, attributesIdentical(α, β) is indeterminate, due to the potential existence of default attributes in the DTD or schema.

d) The ordered lists α[children] and β[children] are examined to define the value childrenIdentical(α, β). Let a be the subsequence of α[children] and b be the subsequence of β[children] consisting of only the element and character information items therein (thus, comment, processing instruction, and unexpanded entity reference items are ignored, just as they are by XML Schema).

   1) If equalQuickList(a, b) is Equal, then childrenIdentical(α, β) is Equal. That is, an exact match guarantees equality.

   2) Otherwise, let A and B be, respectively, the subsequences of a and b containing only element information items. If there does not exist a permutation p of b such that equalQuickList(a, p) is Equal or indeterminate, then childrenIdentical(α, β) is not Equal. That is, because the potential existence in the schema of a model group with a {compositor} of *all*, possibly even in a content model with content type *mixed*, we must allow for potential reordering of the elements in comparing for equality. But, if no such reordering can be made to work, then we can know for certain that no equality is possible.

   3) Otherwise, if one of the lists a and b is empty and the other contains only character information items, then childrenIdentical(α, β) is indeterminate, since the schema might indicate a default content value which is Equal to the non-empty list.

   4) Otherwise, if both of the lists a and b contain only character information items, then childrenIdentical(α, β) is the value returned by equalQuickSimple(a, b, false). Element content consisting entirely of characters might be an occurrence of the use of simple types and so must be conservatively evaluated as such.

   5) Otherwise, if at least one of a and b contains any element information items and at least one of a or b contains any non-whitespace character information items, then (the content type must be *mixed*, and so) let the character information items in a and b be divided respectively into sequences of sub-lists A_1 through A_k and B_1 through B_k such that k-1 is the number of element information items in each of a and b (necessarily the same due to d)2) above) and any given A_i or B_i consists of all those character items in order in a or b that are separated therein by two consecutive element items or an element item and the start or end of the list as the case may be. If there exists any A_i and corresponding B_i such that equalQuickList(A_i, B_i) is not Equal, then childrenIdentical(α, β) is not Equal. That is, the characters used in mixed content must match exactly.

   6) Otherwise, childrenIdentical(α, β) is indeterminate.

e) If either attributesIdentical(α, β) is not Equal or childrenIdentical(α, β) is not Equal, then not Equal is returned.

f) Otherwise, if either attributesIdentical(α, β) is indeterminate or childrenIdentical(α, β) is indeterminate, then indeterminate is returned.

g) Otherwise, Equal is returned.

If α and β are both **attribute** information items, then the following steps are considered in order:

a) If α[namespace name] is not identical to β[namespace name], then not Equal is returned.

b) If α[local name] is not identical to β[local name], then not Equal is returned.

c) Otherwise, equalQuickSimple(α[normalized value], β[normalized value], true) is returned.

If α and β are both **character** information items:

a) If α[character code] is the same as β[character code], then Equal is returned.

b) Otherwise, not Equal is returned.

Otherwise, indeterminate is returned.

C.3 The equalQuickList function

The equalQuickList function takes as input two ordered lists of information items α and β and returns Equal, not Equal, or indeterminate as follows.

a) If the size of α differs from the size of β, then not Equal is returned.

b) If there exists any member a of α and corresponding member b of β such that equalQuickItem(a, b) is not Equal, then not Equal is returned.

c) If there exists any member a of α and corresponding member b of β such that equalQuickItem(a, b) is indeterminate, then indeterminate is returned.

d) Otherwise, Equal is returned.

C.4 The equalQuickSimple function

It is intended that equalQuickSimple embody the appropriate comparison tests for a sequence of characters that are either known to be or might potentially be the data consisting of a simple type. The equalQuickSimple function takes as input two sequences of character information items α and β and a boolean γ and returns Equal, not Equal, or indeterminate as follows.

a) If equalQuickList(α, β) is Equal, then Equal is returned. That is, an exact match guarantees equality.

b) Otherwise, if γ is true, then indeterminate is returned. If α and β are not identical, then their canonicalized lexical representations still might be. A more elaborate implementation might perhaps consider each of the various data types and their possible canonicalized lexical representations in order to in some situations eke out a not Equal instead of indeterminate, but such is not elaborated here.

c) Otherwise, if γ is false, then indeterminate is returned.

Annex D
(informative)

Example Rights Expressions

D.1 Overview of examples

This annex presents both a simple end-user license example and a richer, distribution license example. The two examples are related in that the simple end-user license is issued pursuant to the distribution license.

The two examples, when taken together, demonstrate some of the most typical features of the language and illustrate how those features can enable the business model that is presented throughout the course of this Annex.

It is worth noting that the business model presented here does not come at the cost of such Rights Expression complexity as to make it impossible for an otherwise-capable resource-constrained device to participate in the scenario as an end user device. On the contrary, the end user License lends itself very well to a number of optimizations (for instance, compression of the License or targeted interpretation) that would make such participation not only possible, but exceedingly practical as well.

Interpretation of the distribution License is somewhat more involved than interpretation of the end user License, but this is the trade-off for increased functionality. While some of the most highly resource-constrained devices may have difficulty supporting the functionality of the distribution License, a good system designer will typically direct such functionality toward those devices that have sufficient resources for the desired level of functionality.

There are four actors in the following examples: Acme, Alice, John, and Xin. To gain some perspective on the resource limitations related to each of the actors, it may help to consider what devices each of them may have. Acme maintains a music club. It may be sponsored by a consortium of authors and run a web site on a single old personal computer with a 100 MHz processor. Alice is a digital item author. She may have a personal computer that she uses to check her e-mail and run her favourite authoring applications. John is an end user who wishes to experience Alice's digital item. He might be operating a cell phone with the appropriate display or audio capabilities. Xin is a distributor operating a License server providing Licenses on demand to many cell phone users like John. Xin's license server might be mirrored for load balancing and likely has significant bandwidth, storage, and processor power available to it.

D.2 Simple end-user License example

Alice's Digital Item has identifier `urn:grid:a1-abcde-1234567890-f`. John wishes to experience Alice's Digital Item. Xin provides the following License to John allowing him to play the Digital Item during 2003:

```
<?xml version="1.0" encoding="UTF-8"?>
<r:license
xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-REL-MX-NS rel-mx.xsd">
   <r:grant>
      <r:keyHolder licensePartId="John">
         <r:info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>KtdToQQyzA==</dsig:Modulus>
```

```
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </r:info>
      </r:keyHolder>
      <mx:play/>
      <mx:diReference>
         <mx:identifier>urn:grid:a1-abcde-1234567890-f</mx:identifier>
      </mx:diReference>
      <r:validityInterval>
         <r:notBefore>2003-01-01T00:00:00</r:notBefore>
         <r:notAfter>2004-01-01T00:00:00</r:notAfter>
      </r:validityInterval>
   </r:grant>
   <!--The license is issued by Xin, the distributor.-->
   <r:issuer>
      <r:keyHolder licensePartId="Xin">
         <r:info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>X0j9q99yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </r:info>
      </r:keyHolder>
   </r:issuer>
</r:license>
```

Looking closely at the License above, we observe that it has two parts: an `r:grant` and an `r:issuer`, as shown in the skeletal License outline below.

```
<r:license ...>
   <r:grant>
      <r:keyHolder licensePartId="John">...</r:keyHolder>
      <mx:play/>
      <mx:diReference>...</mx:diReference>
      <r:validityInterval>...</r:validityInterval>
   </r:grant>
   <r:issuer>
      <r:keyHolder licensePartId="Xin">...</r:keyHolder>
   </r:issuer>
</r:license>
```

The issuer is Xin. (Note: It is assumed for the purposes of this License that the authenticity and integrity of the License are determined by other means. For instance, if Xin owns the cell phone network that John is on, it is very possible that he has some other method to insure the authenticity and integrity of his Licenses. Optionally, the License could have been signed inside the `r:issuer` to guarantee the authenticity and integrity inline.) The `r:grant` has four parts:

a) The `r:keyHolder` identifies that John is being granted the rights.

b) The `mx:play` identifies the right that John is being granted.

c) The `mx:diReference` identifies the Digital Item, `urn:grid:a1-abcde-1234567890-f`, to which John is being granted rights.

d) The `r:validityInterval` identifies the time interval, 2003, during which John is allowed to play the Digital Item.

D.3 Distribution License example

In order to issue the end user License shown in D.2, Xin needs to have a License from Alice to do so. Alice, however, does not want to give Xin a separate License for each possible person to whom he may want to issue. Instead, she prefers to give Xin one License that allows him to issue to any number of people. In doing so, she also wants to constrain her License in the following ways:

— Each time Xin issues a License including an `r:grant` pursuant to her License, Xin must pay her $1.

— The `r:grant` elements that Xin includes in the Licenses he issues pursuant to her License must have the exact Right, play, for an exact Resource (her Digital Item with identification `urn:grid:a1-abcde-1234567890-f`) under an exact Condition (that playing occurs during 2003).

— The `r:grant` elements that Xin includes in the Licenses he issues pursuant to her License must be grants to Principals who are members of Acme music club at the time he issues the Licenses.

Alice gives Xin the following License:

```
<?xml version="1.0" encoding="UTF-8"?>
<r:license
xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-REL-MX-NS rel-mx.xsd">
   <r:grant>
      <r:forAll varName="AcmeMusicClubMember">
         <r:propertyPossessor>
            <sx:propertyUri definition="urn:acme:musicClubMember"/>
            <r:trustedRootIssuers>
               <r:keyHolder licensePartId="Acme">
                  <r:info>
                     <dsig:KeyValue>
                        <dsig:RSAKeyValue>
                           <dsig:Modulus>aaaM4ccyzA==</dsig:Modulus>
                           <dsig:Exponent>AQABAA==</dsig:Exponent>
                        </dsig:RSAKeyValue>
                     </dsig:KeyValue>
                  </r:info>
               </r:keyHolder>
            </r:trustedRootIssuers>
         </r:propertyPossessor>
      </r:forAll>
      <r:keyHolder licensePartId="Xin">
         <r:info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>X0j9q99yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </r:info>
      </r:keyHolder>
      <r:issue/>
      <r:grant>
         <r:principal varRef="AcmeMusicClubMember"/>
         <mx:play/>
         <mx:diReference>
```

```
            <mx:identifier>urn:grid:a1-abcde-1234567890-f</mx:identifier>
         </mx:diReference>
         <r:validityInterval>
            <r:notBefore>2003-01-01T00:00:00</r:notBefore>
            <r:notAfter>2004-01-01T00:00:00</r:notAfter>
         </r:validityInterval>
      </r:grant>
      <sx:feePerUse>
         <sx:rate>
            <sx:amount>1.00</sx:amount>
         </sx:rate>
         <sx:to>
            <!--This is Alice's bank account information.-->
            <sx:aba>
               <sx:institution>139371581</sx:institution>
               <sx:account>111111</sx:account>
            </sx:aba>
         </sx:to>
      </sx:feePerUse>
   </r:grant>
   <!--The license is issued and signed by Alice, the author.-->
   <r:issuer>
      <dsig:Signature>
         <dsig:SignedInfo>
            <dsig:CanonicalizationMethod
               Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <dsig:SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <dsig:Reference>
               <dsig:Transforms>
                  <dsig:Transform Algorithm=
                     "urn:mpeg:mpeg21:2003:01-REL-R-NS:licenseTransform"/>
               </dsig:Transforms>
               <dsig:DigestMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <dsig:DigestValue>Jk9QbKOQCo941tTExbj1/Q==</dsig:DigestValue>
            </dsig:Reference>
         </dsig:SignedInfo>
         <dsig:SignatureValue>DFgqOhh5QQ==</dsig:SignatureValue>
         <dsig:KeyInfo>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>gOyM4ccyzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </dsig:KeyInfo>
      </dsig:Signature>
   </r:issuer>
</r:license>
```

Looking closely at the License above, we observe that it has the same two high-level parts as the end user License: an `r:grant` and an `r:issuer`, as shown in the following skeletal License outline.

```
<r:license ...>
   <r:grant>
      <r:forAll varName="AcmeMusicClubMember">...</r:forAll>
      <r:keyHolder licensePartId="Xin">...</r:keyHolder>
      <r:issue/>
      <r:grant>
```