INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PIC TURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11/N5349
December 2002, Awaji, JP

Title:	Text of ISO/IEC FCD 21000-5 Rights Expression Language
Source:	Multimedia Description Schemes Group
Status:	Approved
Editors:	Thomas DeMartini (ContentGuard, US), Xin Wang (ContentGuard, US), Barney Wragg (UMG, UK)

ISO/IEC JTC 1/SC 29/WG 11 N 5349

Date: 2002-12-13

ISO/IEC FCD 21000-5

ISO/IEC JTC 1/SC 29/WG 11

Secretariat: XXXX

Information technology — Multimedia framework (MPEG-21) — Part 5: Rights Expression Language

Document type:
Document subtype:
Document stage:
Document language: E

Contents

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 21000 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 21000-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *JTC*, Subcommittee SC 29.

This second/third/... edition cancels and replaces the first/second/... edition (), [clause(s) / subclause(s) / table(s) / figure(s) / annex(es)] of which [has / have] been technically revised.

ISO/IEC 21000 consists of the following parts, under the general title *Information Technology — Multimedia Framework*:

Part 1: Vision, Technologies and Strategy

Part 2: Digital Item Declaration

Part 3: Digital Item Identification & Description

Part 4: IPMP

Part 5: Rights Expression Language

Part 6: Rights Data Dictionary

Part 7: Digital Item Adaptation

Part 8: Reference Software

Part 9: File Format

Part 10: Digital Item Processing

Part 11: Persistent Association

Part 12: Test Bed for MPEG-21 Resource Delivery

Introduction

Today, many elements exist to build an infrastructure for the delivery and consumption of multimedia content. There is, however, no “big picture” to describe how these elements, either in existence or under development, relate to each other. The aim for MPEG-21 is to describe how these various elements fit together. Where gaps exist, MPEG-21 will recommend which new standards are required. ISO/IEC JTC 1/SC 29/WG 11 (MPEG) will then develop new standards as appropriate while other relevant standards may be developed by other bodies. These specifications will be integrated into the multimedia framework through collaboration between MPEG and these bodies.

The result is an open framework for multimedia delivery and consumption, with both the content creator and content consumer as focal points. This open framework provides content creators and service providers with equal opportunities in the MPEG-21 enabled open market. This will also be to the benefit of the content consumer providing them access to a large variety of content in an interoperable manner.

The vision for MPEG-21 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.

This fifth part of MPEG-21 (ISO/IEC 21000-5) specifies the expression language for issuing rights for Users to act on Digital Items, their Components, Fragments, and Containers.

Information technology — Multimedia framework (MPEG-21) — Part 5: Rights Expression Language

1 Scope

1.1 Organization of the document

This document explains the basic concepts of a machine-interpretable language for issuing rights to Users to act upon Digital Items, Components, Fragments, and Containers. It does not provide specifications for security in trusted systems, propose specific applications, or describe the details of the accounting systems required. This document does not address the agreements, coordination, or institutional challenges in building an implementation of this standard. The standard describes the syntax and semantics of the language.

Clause 1 introduces this part of ISO/IEC 21000. Clause 2 gives the normative references. Clause 3 specifies conformance. Clause 4 gives pertinent terms and definitions. Clause 5 specifies the REL Core, composed of the REL Architecture, supporting types and elements, and REL Authorization Algorithm. Clause 6 specifies the types, elements, codes, and functions used to specify rights, resources, conditions, payment terms, service descriptions, countries, regions, currencies, and regular expressions that are useful not only in the multimedia domain but other domains as well. Clause 7 specifies the types and elements used to specify rights, resources, and conditions particular to multimedia. Annex A specifies the W3C XML Schema definition of the types and elements defined throughout this part of ISO/IEC 21000. Annex B gives some example rights expressions. Annex C demonstrates how to introduce new rights as an extension to this part of ISO/IEC 21000. Annex D describes the relationship between ISO/IEC 21000-6 and this part of ISO/IEC 21000.

1.2 Conventions

1.2.1 Typographical Conventions

Sequences of characters in all capital letters are key words (as described in Clause 1.2.2) or abbreviations.

Sequences of characters in italics are the names of variables (in the mathematical sense) used to formally describe syntax and semantics.

Sequences of characters in fixed-width font are literal machine-readable character sequences. Conventions regarding machine-readable character sequences for schemas are described in more detail in Clause 1.2.3.

1.2.2 Keyword Conventions

The keyword "REL" in this document is to be interpreted as referring to this part of ISO/IEC 21000.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.2.3 Schema Conventions

The syntax of REL is described and defined using the XML Schema technology defined by the Worldwide Web Consortium (W3C). Significantly more powerful and expressive than DTD technology, the extensive use of XML Schema in REL allows for significant richness and flexibility in its expressiveness and extensibility.

To that end, a principal design goal for REL is to allow for and support a significant amount of extensibility and customizability without the need to make actual changes to the REL core itself. Indeed, the core itself makes use of this extensibility internally. Others parties may, if they wish, define their own extensions to REL. This is accomplished using existing, standard XML Schema and XML Namespace mechanisms.

Readers of these schemas should notice that a certain editorial style has, for ease of comprehension, been uniformly adopted. The XML Schema artifacts found within the REL core schema fall into three categories: attributes, elements, and types. The names of each have a different stylistic treatment: the names of types are in mixed case, with an initial capital letter, while the names of elements and attributes are in mixed case but with an initial lower case letter. For example, `Grant` is the name of a type, while `grant` is the name of an element and `licensePartId` is the name of an attribute.

This stylistic convention has also been used in this specification when referring to these elements and types:

- A passage herein which mentions an element as in "the `grant`" is using the word in a technical sense to refer to the notion of `grant` as an XML Schema element.

- A passage which mentions a type and prefixes the type with the word "type" as in "the type `Grant`" is using the word in a technical sense to refer to the notion of `Grant` as an XML Schema type.

- A passage which mentions a type and prefixes the type with an article such as "a" or "the" as in "a `Grant`" is using the word in a technical sense to refer to any element whose type is the type `Grant` or any derivation thereof. (Semantics assigned to a type in this way MUST 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.)

1.3 Namespace

The namespace (XML Namespaces) for the REL will be `urn:mpeg:mpeg21:2002:01-REL-NS`. The "01" represents a serial number that is expected to change as the REL schema evolves along with this part of ISO/IEC 21000.

2 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 2100. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 2100 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

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

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

Multimedia Framework ISO/IEC 21000 (all parts).

RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC 2396, August 1998.

RFC 2141, *Uniform Resource Names (URN)*, IETF RFC 2141, May 1997.

RFC 1738, *Uniform Resource Locators (URL)*, IETF RFC 1738, December 1994.

RFC 2119, *Key words for use in RFCs to Indicate Requirements Levels*, IETF RFC 2119, March 1997.

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

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

Schema Centric XML Canonicalization, *Schema Centric XML Canonicalization Version 1.0*, UDDI Version 3, 10 July 2002, http://uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm.

UDDI, *Universal Description, Discovery, and Integration (UDDI)*, http://www.uddi.org/.

WSDL, *Web Services Definition Language (WSDL) 1.1*, W3C Note, 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

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

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

Codes for the representation of names of countries and their subdivisions, ISO 3166 (all parts).

Codes for the representation of currencies and funds, ISO 4217 (all parts).

3 Conformance

3.1 Types of Conformance

Applications claiming conformance to this part of ISO/IEC 21000 must satisfy several types of conformance. All applications must satisfy Basic Conformance. In addition to Basic Conformance, a particular application must also satisfy other conformance types as are appropriate to that application. Additional conformance types beyond those presently defined within Clause 3 may be defined in amendments to this part of ISO/IEC 21000.

3.2 Basic Conformance

An application that satisfies Basic Conformance must possess all of the following properties:

- `License`s created by it must be schema-valid according to the schemas in Annex A.
- `License`s created by it must not conflict with any of the conformance statements throughout this part of ISO/IEC 21000. For example, clause 5.1.2 requires that "For a given `LicensePartId` value *v*, there may be at most one `LicensePart` in a given `License` that contains a `licensePartId` attribute with the value *v*." Conformant applications must not create any `License`s violating that statement.
- `License`s interpreted by it must be interpreted consistent with the semantics defined throughout this part of ISO/IEC 21000 and exemplified by the REL Authorization Algorithm defined in clause 5.7.

4 Terms and definitions

4.1 Terminology

For the purposes of this International Standard, the terms and definitions given in ISO/IEC 21000-1 and the following apply.

4.1.1
condition

something that must exist or be fulfilled in order for a **right** to be exercised. Examples include temporal constraints, payment, territorial location, exercise limit, and possession of some credentials and other **rights**.

4.1.2
digital resource

a **resource** that exists in the digital domain.

4.1.3
principal

an encapsulation of the identification of an entity involved in the granting or exercise of **rights**.

4.1.4
repository

a system that can hold **digital resources**, such as personal systems, on-line storefront systems, library systems, and archive systems.

4.1.5
resource

the object to which a **principal** may be granted **rights**. A **resource** can be a digital work (such as an e-book, an audio or video file, or an image), a service (such as an email service, or B2B transaction service), or even a piece of information that can be owned by a **principal** (such as a name or an email address).

4.1.6
resource attribute
a property that a **resource** possesses. Attributes include authorship, ownership, formats, measures, categories, location, and creation time. Attributes may also be security related, such as ones about encryption, watermark and **repository** for protecting the resource.

4.1.7
right

a privilege that someone may claim or that is due to them, which makes them entitled to make copies of, distribute, or perform all or part of a published or recorded work for a certain extended period of time. In REL, it is the "verb" that a **principal** can be granted to exercise against some **resource** under some **condition**. Typically, a **right** specifies an action (or activity) or a class of actions that a **principal** may perform on or using the associated **resource**.

4.2 Acronyms

For the purposes of this International Standard, the following acronyms apply.

4.2.1
IETF

Internet Engineering Task Force

4.2.2
IETF RFC

Internet Engineering Task Force Request For Comments

4.2.3
RDD

Rights Data Dictionary

4.2.4
REL

Rights Expression Language

4.2.5
UDDI

Universal Description, Discovery, and Integration

4.2.6
URI

Uniform Resource Identifier

4.2.7
URL

Uniform Resource Locator

4.2.8
URN

Uniform Resource Name

4.2.9
WSDL

Web Services Description Language

4.2.10
W3C

World Wide Web Consortium

4.2.11
XML

Extensible Markup Language

5 REL Core

5.1 Architectural Details of the REL Core

At the heart of REL is the REL Core Schema. The elements and types defined therein define the core structural and validation semantics that comprise the essence of the specification. It is expected that every REL validation processor will be aware of the semantics embodied in this core. That is not to say that each and every such processor need to implement and fully support all of the functionality herein described; rather, it indicates that such processors must be conscious of all the semantics defined therein that logically affect those core features they indeed do choose to support. This is also true for REL extensions that these processors intend to process.

5.1.1 License

The single most important concept in REL is that of the **`License`**. A `License` is conceptually a container of **`Grant`s**, each one of which conveys to a particular **`Principal`** the sanction to exercise some identified **`Right`** against some identified **`Resource`**, possibly subject to the need for some **`Condition`** to be first fulfilled. A `License` is also a container of **`GrantGroup`s**, each of which is in turn an eventual container of `Grant`s. To avoid confusion, it should be noted that, while a `License` *is* a conceptual container, it is not *only* just a container: it is also the means by which `License` issuers convey authorization.

A `License` may be issued by a party, signifying that the party **authorizes** certain `Grant`s and `GrantGroup`s. This semantic notion of whether or not a `Grant` or `GrantGroup` has been authorized is an important one. A `Grant` or `GrantGroup` which has not been authorized conveys no authorization, it merely exists as an XML element. Unless otherwise indicated by this specification, `Grant`s or `GrantGroup`s which may physically appear in a `License` are not to be considered authorized.

Syntactically, multiple `Issuer`s may be present on a given `License`; however no additional semantic is associated with their collective issuance. The semantics are, rather, as if they had each independently issued their own copy of the `License`. Therefore, one can unambiguously speak of *the* `Issuer` of a given `License`.

5.1.1.1 License/title

Each of the zero or more `title` elements in a `License` provides a descriptive phrase about the `License` that is intended for human consumption in user interfaces and the like. Automated processors MUST NOT interpret semantically the contents of such `title` elements.

5.1.1.2 License/grant and License/grantGroup

The `Grant`s and `GrantGroup`s contained in a `License` are the means by which authorization policies are conveyed in the REL architecture.

Each `Grant` or `GrantGroup` that is an immediate child of a `License` exists independently within that `License`: no collective semantic (having to do with their particular ordering or otherwise) is intrinsically associated with the presence of two or more of them within a certain one `License` (though there may be syntactic issues; see License Parts).

See below in this specification for an elaboration of the semantics of `Grant` and `GrantGroup`.

5.1.1.3 License/issuer

Each `Issuer` in a `License` may contain two pieces of information:

- a set of `Issuer`-specific `details` about the circumstances under which he issues the `License`, and
- an identification of the issuer, possibly coupled with a digital signature for the `License`.

The optional `Issuer`-specific `details` are found in the `Issuer/details` element, which is of type `IssuerDetails`. These `details` optionally include any of the following information:

1. the specific date and time at which this `Issuer` claims to have effected his issuance of the `License`.
2. an indication of the mechanism or mechanisms by which the `Issuer` of the `License` will, if he later `Revoke`s it, post notice of such revocation. When checking for revocation, REL processing systems may choose to use *any* one of the identified mechanisms: that is, they are all considered equally authoritative as to the revocation status of the issuance of the `License`.

Let *g* be any `Grant` or `GrantGroup` which is an immediate child of a `License` *l*, and let *i* be the `Issuer` element of *l*. If *i*/`dsig:Signature` is present and Core Validation (XML Digital Signature) of *i*/`dsig:Signature` succeeds, then *g* is defined to be *directly authorized* by the issuing `Principal` whose signature over *l* appears in *i*. On the other hand, if *i*/`principal` is present and it can be verified out of band that *i*/`principal` is the issuer of *l*, then *g* is defined to be *directly authorized* by *i*/`principal`. Lastly, if neither *i*/`dsig:Signature` nor *i*/`principal` is present, yet the issuing `Principal` of *l* can still be determined out-of-band, then *g* is defined to be *directly authorized* by that issuing `Principal` (determined out-of-band). Otherwise, if none of these are the case, the presence of *i* in *l* does not imply the authorization of *g*.
When `dsig:Signature` is used within an `Issuer`, some of the general freedoms and flexibilities permitted by XML-Signature Syntax and Processing (XML Digital Signature) are profiled and constrained. Specifically, with the aim of simplifying the determination of exactly which pieces of the `License` have and have not been actually signed by a given `Issuer`, the `dsig:Signature/dsig:SignedInfo/dsig:Reference` elements are restricted in how they may refer to pieces of the `License`. In concept, the restriction is that, of the information in a `License`, a signature may only reference

a. the whole `License` less its `Issuer` children, together with

b. the issuance `details` corresponding to the `dsig:Signature`

but not any other piecemeal subparts of the `License` (the `dsig:Signature` may still, if it wishes, reference items external to the `License` though such use is beyond the scope of this specification). Concretely, when an `Issuer` wishes to reference pieces of the `License`, to do so it MUST use a `dsig:Signature/dsig:SignedInfo/dsig:Reference` element *r* such that the following is true:

1. the attribute *r*/@`dsig:URI` MUST be omitted
2. the element *r*/`dsig:Transforms` MUST contain exactly one child `dsig:Transform` element *t*, where

   a. *t* MUST be empty

   b. the attribute *t*/@`dsig:Algorithm` MUST contain the value `http://www.xrml.org/schema/2002/05/xrml2core#license`

The transform algorithm so indicated is known as the *REL License Transform Algorithm* .

A `dsig:Transform` element *t* indicating the use of the REL License Transform Algorithm emits as output the most immediate ancestor of *t* that is of type `License` or a derivation thereof but with any element descendants of that `License` which occupy (perhaps through type derivation) the particle defined by the `issuer` child of the `License` wholly removed, except for that `Issuer` that contains *t*, which is kept, removing its `dsig:Signature` child instead.

It is RECOMMENDED that `dsig:Signature`s created by issuers of REL `License`s indicate the use of the Schema Centric Canonicalization algorithm (Schema Centric XML Canonicalization).

Moreover, as a general note of good digital signature hygiene, it is RECOMMENDED that REL `License`s explicitly (re)declare no higher up the XML element tree than at the `License` level any XML Namespaces that are used anywhere throughout the `License`. That is, a `License` SHOULD be a self-contained unit with respect to XML Namespace declarations, not relying on any such declarations to be imported from their surrounding XML context. This hygienic practice greatly facilitates the ability to manipulate `License`s as a self-contained XML unit within REL processing systems.

5.1.1.4 License/inventory

REL provides a syntactic mechanism for reducing redundancy and verbosity in `License`s. This syntactic macro-like mechanism can be used throughout a `License`, so long as there is in a given `License` only one definition to each `LicensePartId`. Such definitions can lie, for example, inside of `grant`s or other semantically important structures. However, it is sometimes useful and convenient to be able to provide a definition of a part of a `License` without at the definition site necessarily associating any particular semantic with the part. The `inventory` element provides a means for doing this.

The `inventory` element of a `License` is a simple container of `LicensePart`s. The presence of such parts in the `inventory` container does not provide any semantic at all. The parts simply exist as syntactic structures within the `inventory`. Usefully and usually, parts in the `inventory` will have `LicensePart`/@`licensePartId` attributes so that they can be referenced from elsewhere in the `License`.

5.1.1.5 License/otherInfo

Using the wildcard construct from XML Schema, a `License` provides an extensibility hook within which `License` issuers may place additional content as they find appropriate and convenient. This can be useful for conveying information which is peripherally related to, for example, authentication and authorization, but is not part of the REL core infrastructure. Such content will of necessity be referenced by the `dsig:Signature` of the `Issuer` of the `License`, and so can be considered as being attested to by the `License`'s `Issuer`; indeed, it is the inclusion of this data in the signature which is likely the most important reason for contemplating the use of this facility.

It should, however, be carefully understood that not all processors of REL `License`s will understand the semantics intended by any particular use of this extensibility hook. Processors of the `License` MAY choose wholly at their own discretion to completely ignore any such content that might be present therein.

5.1.1.6 License/encryptedLicense

A mechanism is provided by which the contents of a License may be encrypted and so hidden from view from inappropriate parties. This mechanism makes straightforward use of XML Encryption Syntax and Processing (XML Encryption).
Specifically, the XML content model of a License is a choice between a sequence containing the elements previously described in this section and an encryptedLicense element. encryptedLicense represents the encryption of the contents (but not the attributes) of the License element. See the type EncryptedContent for a more detailed discussion of the decryption process.

5.1.1.7 License Attributes

A License may have a licenseId attribute which indicates the URI that may be used to identify the License. Additionally, using the wildcard construct from XML Schema, a License provides an extensibility hook within which License issuers may place additional attributes as they find appropriate and convenient. This can be useful for conveying information which is peripherally related to, for example, authentication and authorization, but is not part of the REL core infrastructure. Such content will of necessity be referenced by the dsig:Signature of the Issuer of the License, and so can be considered as being attested to by the License's Issuer. It should, however, be carefully understood that not all processors of REL Licenses will understand the semantics intended by any particular use of this extensibility hook. Processors of the License MAY choose wholly at their own discretion to completely ignore any such content that might be present therein.

5.1.2 License Parts

Many of the types defined in REL are, in the XML Schema sense, derivations of the type LicensePart, including Grants, Resources, and Rights, just to name a few.
The role of LicensePart is twofold:

1. LicensePart, through its licensePartId and licensePartIdRef attributes, which are both of type LicensePartId, defines a macro-like purely syntactic mechanism by which fragments of XML which must logically be present in several places within a License may avoid being literally written out multiple times.

2. In contrast, LicensePart, through its varRef attribute, defines a *semantically* important mechanism. As is later described herein, REL defines a pattern-matching mechanism which may be used, for example, to denote sets of Principals that a grant might apply to or sets of grants that might be validly issued by an authorized authority. Such patterns logically describe sets of entities. When a pattern is applied to a concrete situation, a matching process occurs, resulting in a single entity that matches that pattern. It is useful to be able to, elsewhere in a License, talk about the entity that might match a given pattern when such matching process later occurs.

The matching process and its relationship to variables is somewhat involved, and a detailed discussion is provided later in this specification.
The macro-like facility of licensePartId and licensePartIdRef, on the other hand, is quite straightforward. Use of the licensePartId and licensePartIdRef attributes MUST adhere to the following constraints:

1. On any given LicensePart at most one of the attributes licensePartId and licensePartIdRef may appear. That is, it is illegal for both attributes to be present on one LicensePart.

2. For a given `LicensePartId` value *v*, there may be at most one `LicensePart` in a given `License` that contains a `licensePartId` attribute with the value *v*.

3. If a `LicensePart` *p* contains a `licensePartIdRef` attribute, then it MUST have empty content. As a corollary, therefore, it is required that all types which are derivations of `LicensePart` SHOULD allow their content to be empty (for otherwise they cannot usefully be used within the `LicensePart` infrastructure).

4. If a `LicensePart` *p* contains a `licensePartIdRef` attribute with a certain value *v*, then there must exist some (other) `LicensePart` *q* in the same `License` as *p* which has a `licensePartId` attribute with value *v* (and, per (2), there cannot be two such *q*s). It is further required that the expanded element name of *p* exactly match that of *q*. Moreover, it is required that *q* not be an ancestor of *p* (or, per (3), a descendant of *p*).

If a `LicensePart` *p* contains a `licensePartIdRef` attribute with a certain value *v*, and *q* is the `LicensePart` in the same `License` as *p* which has a `licensePartId` attribute with value *v*, then the semantics of the `License` containing *p* and *q* are as if:

a. *p* were removed from the `License` and replaced with a copy *q'* of the element *q*,

b. the `licensePartId` attribute were removed from *q'* and all of its descendants,

c. any "preserved" attributes that may be present on *q'*were removed therefrom, and

d. any "preserved" attributes that may be present on *p* were copied and added to *q'*.

where here a "preserved" attribute is any of the following:

1. any attribute of type `xsd:ID`

2. any attribute for which 'id' is the LocalPart of its qualified name

(It is the intent of the last of these points to allow for the useful definition of other identification systems on license parts beyond the document-global `xsd:ID`-typed identifiers.)
If a `License` contains no `LicensePart`s with a `licensePartIdRef` attribute, then the semantics of that `License` are as if the `licensePartId` attribute were removed from all `LicensePart`s with such a `licensePartId`.
With the exception of signature verification, both `licensePartIdRef` macro expansion and `licensePartId` removal MUST be carried out before the other `License` processing steps defined by this specification. In particular, it is carried out before such processing as the evaluation of variable references or the testing of equality.

5.1.3 Equality of XML Elements

REL defines a formal notion by which two arbitrary XML elements can be compared and said to be " *equal* " or not. This notion is used extensively and heavily in the design in such places, for example, as determining whether a `Grant` in a particular `License` actually contains a particular `Right` which is attempting to be exercised. In order to determine this, the `Right` being exercised must be compared in a precise and technical manner against the `Right` in the `Grant`. Perhaps surprisingly, no existing notion of equality appears defined on XML elements. Accordingly, we define one here as follows.

5.1.3.1 Background

In order to address the question of equality, one must first consider the question of whether the notion of the XML information conveyed by a piece of XML is in fact well-defined. Fortunately, this is in fact the case: the XML Information Set specification

(XML Infoset) normatively defines the abstract information contained in any the possible physical representations of a piece of XML. This information is, however, altered by XML Schema (XML Schema), in that the assessment of validation of an infoset by XML Schema augments that infoset with information contained in the schema(s) in question (for example, default values are inserted, the character content of elements of simple type is normalized, and so on). Thus, in order to understand the full set of information conveyed by a piece of XML, one must, generally speaking, validate the data according to its schemas.

If all the schemas in question are relatively fixed, and so 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. 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 infosets are considered to actually convey the same information. This is precisely the sort of situation which is likely to arise in many REL applications, especially those that act as utility layers for solutions that exploit the extensibility and customizability of the REL architecture.

What is needed, therefore, is an efficiently implementable, generic algorithm that evaluates whether two XML information items are equal according to the representational liberties permitted by the schemas of the items in question. It is the intent of this specification to define such an algorithm.

5.1.3.2 Overview of Equality Comparison

As was mentioned, the information conveyed by a piece of XML can generally speaking only be understood by considering the content of the information set for that XML together with the content of the schemas with which it is associated. Fortunately, it was one of the central design goals of the Schema Centric Canonicalization algorithm (Schema Centric XML Canonicalization) to exactly capture this information. That is, the result of processing some XML through Schema Centric Canonicalization captures in its output *all* of the information content of the XML that was latent in the schemas with which it is associated; all the contributions such as default values, data type lexical canonicalization, and so on, are extracted and made explicitly manifest in the canonicalized form. Therefore, one can succinctly compare two XML information items for equality by comparing the bit strings of their respective processing by Schema Centric Canonicalization: the items are equal if and only if the bit strings are bit-for-bit identical.

Were that algorithm easy to efficiently implement, then little more need be said about the matter. Unfortunately, this is not the case: Schema Centric Canonicalization is to an approximation at least as complicated to implement as full-blown XML Schema validity assessment, which is, unfortunately, in many situations, more expensive than is reasonable. In order to address this, we therefore seek an additional, efficiently implementable algorithm that can, in certain identifiable common cases, evaluate whether two XML items are equal or not in the same sense as processing through Schema Centric Canonicalization would do, but without the expense involved (specifically, without the expense of retrieving and processing the associated schemas). When such an algorithm identifies that the common case is in use, it can quickly give a definitive answer; in other cases, the full treatment through the Schema Centric Canonicalization algorithm is necessary.

Of course, many such auxiliary algorithms are possible, differing (likely) in exactly which set of common cases they cover. We present one of these possible algorithms here (embodied in the *equalQuickItem* function defined below), one that we believe will be of broad general utility. Note, however, that implementations are free to alter or augment this algorithm in order to appropriately tailor and tune it for their specific needs.

5.1.3.3 Specification of Equality Comparison

Two XML information items, *left* and *right*, are to be considered *equal* or *not equal* according to the application of the function *equalItem(left, right)*.

5.1.3.3.1 The equalItem function

The *equalItem* function takes two information items, *left* and *right*, as inputs and yields either the result *equal* or the result *not equal* as follows:

1. If *equalQuickItem(left, right)* is *equal* or *not equal*, then *equalItem(left, right)* is that value.
2. Otherwise, let *leftBits* and *rightBits* respectively be result of the execution of the Schema Centric Canonicalization algorithm on an infoset whose document information item contains in its [children] property the item *left* or *right* (respectively). Then if *leftBits* is the identical bit string to *rightBits*, then *equalItem(left, right)* is *equal*; otherwise, *equalItem(left, right)* is *not equal*.

5.1.3.3.2 The equalQuickItem function

The *equalQuickItem* function takes two information items, *left* and *right*, 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 *left* and *right* are different kinds of information item, then *not equal* is returned.
If *left* and *right* are both **element** information items, then the following steps are considered in order:

1. If *left[namespace name]* is not identical to *right[namespace name]* then *not equal* is returned.
2. If *left[local name]* is not identical to *right[local name]*, then *not equal* is returned.
3. The sets *left[attributes]* and *right[attributes]* are examined to define the value *attributesIdentical(left, right)*:
    a. If a permutation *r'* of *right[attributes]* exists such that *equalQuickList(left[attributes], r')* is *equal*, then *attributesIdentical(left, right)* is *equal.*
    b. Otherwise, if *left[attributes]* contains a member *ll* and *right[attributes]* contains a member *rr* where both
        i. *ll[namespace name]* is identical to *rr[namespace name]* and
        ii. *ll[local name]* is identical to *rr[local name]*

        then if *equalQuickItem(ll, rr)* is *not equal* or *indeterminate*, then *attributesIdentical(left,right)* is *not equal* or *indeterminate*, respectively.
    c. Otherwise, *attributesIdentical(left,right)* is *indeterminate*, due to the potential existence of default attributes in the DTD or schema.
4. The ordered lists *left[children]* and *right[children]* are examined to define the value *childrenIdentical(left, right)*. Let *lec* be the subsequence of *left[children]* and *rec* be the subsequence of *right[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).

a. If *equalQuickList(lec, rec)* is *equal*, then *childrenIdentical(left, right)* is *equal*. That is, an exact match guarantees equality.

b. Otherwise, let *le* and *re* be respectively the subsequences of *lec* and *rec* containing only element information items. If there does not exist a permutation *rec'* of *rec* such that *equalQuickList(lec, rec')* is *equal* or *indeterminate*, then *childrenIdentical(left, right)* 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.

c. Otherwise, if one of the lists *lec* and *rec* is empty and the other contains only character information items, then *childrenIdentical(left, right)* is *indeterminate*, since the schema might indicate a default content value which is equal to the non-empty list.

d. Otherwise, if both of the lists *lec* and *rec* contain only character information items, then *childrenIdentical(left, right)* is the value returned by *equalQuickSimple(lec, rec, 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.

e. Otherwise, if at least one of *lec* and *rec* contains any element information items and at least one of *lec* or *rec* contains any non-whitespace character information items, then (the content type must be *mixed*, and so) let the character information items in *lec* and *rec* be divided respectively into sequences of sub-lists l_1 through l_k and r_1 through r_k such that *k-1* is the number of element information items in each of *lec* and *rec* (necessarily the same due to 4(b) above) and any given l_i or r_i consists of all those character items in order in *lec* or *rec* 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 l_i and corresponding r_i such that *equalQuickList*(l_i, r_i) is *not equal*, then *childrenIdentical(left, right)* is *not equal*. That is, the characters used in mixed content must match exactly.

f. Otherwise, *childrenIdentical(left, right)* is *indeterminate.*

5. If either *attributesIdentical(left, right)* is *not equal* or *childrenIdentical(left, right)* is *not equal*, then *not equal* is *returned.*

6. Otherwise, if either *attributesIdentical(left, right)* is *indeterminate* or *childrenIdentical(left, right)* is *indeterminate*, then *indeterminate* is *returned.*

7. Otherwise, *equal* is returned.

If *left* and *right* are **attribute** information items, the the following steps are considered in order:

1. If *left[namespace name]* is not identical to *right[namespace name]* then *not equal* is returned.

2. If *left[local name]* is not identical to *right[local name]*, then *not equal* is returned.

3. Otherwise, *equalQuickSimple(left[normalized value], right[normalized value], true)* is returned.

If *left* and *right* are **character** information items:

1. If *left[character code]* is the same as *right[character code]* then *equal* is returned

2. Otherwise, *not equal* is returned.

Otherwise, *indeterminate* is returned.

5.1.3.3.3 The equalQuickList function

The *equalQuickList* function takes as input two ordered lists of information items *left* and *right* and returns *equal*, *not equal*, or *indeterminate* as follows.

1. If the size of *left* differs from the size of *right*, then *not equal* is returned.
2. If there exists any member *ll* of *left* and corresponding member *rr* of *right* such that *equalQuickItem(ll, rr)* is *not equal*, then *not equal* is returned.
3. If there exists any member *ll* of *left* and corresponding member *rr* of *right* such that *equalQuickItem(ll, rr)* is *indeterminate*, then *indeterminate* is returned.
4. Otherwise, *equal* is returned.

5.1.3.3.4 The equalQuickSimple function

It is intended that *equalQuickSimple* embody the appropriate comparison tests for a sequence of characters which are either known to be or may potentially be the data consisting of a simple type. The *equalQuickSimple* function takes as input two sequences of character information items *left* and *right* and a boolean *isAlreadyNormalized* and returns *equal*, *not equal*, or *indeterminate* as follows:

1. *If equalQuickList(left, right)* is *equal*, then *equal* is returned. That is, an exact match guarantees equality.
2. Otherwise, if *alreadyNormalized* is *true*, then *indeterminate* is returned. If *left* and *right* 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.
3. Otherwise, if *isAlreadyNormalized* is *false*, then *indeterminate* is returned.

5.1.4 Patterns

Within REL, it is quite useful and important at times to be able to write in XML formal expressions that semantically denote particular sets of XML instance elements. To give but one example, a `License` that provides to a `Principal` the authorization that is analogous to that held by a "Certificate Authority" in X509 parlance needs to be able to precisely specify and carefully indicate exactly which set of `Grant`s the `Principal` is authorized to `issue`. REL has a rich architecture of "patterns" designed to address this and similar needs.

5.1.4.1 AnXmlPatternAbstract

All formal patterns in REL have types which derive from the type `AnXmlPatternAbstract`. As such, this type forms the root of a type hierarchy of various flavors of patterns suitable for different pattern matching requirements. The corresponding element `anXmlPatternAbstract`, which is of this type, usefully forms the head of a substitution group of all possible patterns.

5.1.4.2 AnXmlExpression

`AnXmlExpression` provides a means by which patterns written in formal expression languages defined outside of REL can be straightforwardly incorporated herein. The particular expression language used is indicated by the `lang` attribute, which is a URI (RFC 2396).

The default value for the `lang` attribute is `http://www.w3.org/TR/1999/REC-xpath-19991116`, which indicates that the contents of the `AnXmlExpression` contains a string which is an XPath (XPath) expression. If the expression contained in that string is not of XPath type boolean, then it is to be automatically converted to such as if the function boolean were applied. An element is said to match an `AnXmlExpression` pattern if the enclosed expression evaluates to true over that element.
All REL processing systems which choose to support the use of any form of REL patterns at all MUST support the use of the `http://www.w3.org/TR/1999/REC-xpath-19991116` expression language in `AnXmlExpression` elements.

5.1.4.3 PrincipalPatternAbstract / RightPatternAbstract / ResourcePatternAbstract / ConditionPatternAbstract

As an alternative to using patterns written in externally-defined expression languages, it is often useful to define new XML types and elements that, in their intrinsic semantic, define some pattern matching algorithm. This can, of course, be done by simply deriving from `AnXmlPatternAbstract`; but, if appropriate to a given situation, deriving one of the four types here might be more useful.
Patterns which are of types which derive from `PrincipalPatternAbstract`, `RightPatternAbstract`, `ResourcePatternAbstract`, and `ConditionPatternAbstract` are always evaluated in a context of an entire XML document which (respectively) contains exactly just one `Principal`, `Right`, `Resource`, or `Condition`. Such known contextual setting may make it possible to more succinctly express and define the semantics of the intended pattern.

5.1.4.4 Everyone

`Everyone` is a type which is derived from `PrincipalPatternAbstract`.
As such, it matches documents which are elements of some subset of the universe of `Principal`s. That subset is defined as those `Principal`s who posses a certain property described within the `Everyone` element.
More precisely, let *e* be an instance of `Everyone`, and let *P* be the set of `Principal`s denoted by *e*. If *e*/`propertyAbstract` does not exist, then *P* is defined to be the entire universe of `Principal`s. Otherwise, *P* is defined to be the set of those `Principal`s *p* for which the following `PrerequisiteRight` condition *q* can be shown to be fulfilled with respect to the same tuple of Authorization Algorithm inputs within which *e* is being processed:

1. *q*/`principal` is equal to *p*
2. *q*/`right` is equal to the `possessProperty` element
3. *q*/`resource` is equal to *e*/`propertyAbstract`
4. *q*/`trustedIssuer` is a copy of *e*/`trustedIssuer` (if such is present) or is absent (otherwise).

5.1.4.5 PatternFromLicensePart

`PatternFromLicensePart` is a semantically simple pattern. Each element of this type contains exactly one `LicensePart`. The pattern is defined to match exactly those elements which are equal to this contained part.

5.1.4.6 GrantPattern

A `GrantPattern` is a relatively complex pattern which matches XML elements of type `Grant`. Let *G* be a `GrantPattern`, and let *g* be a target `Grant` against which one wishes to attempt to match *G*.
The `GrantPattern` *G* can contain four separate pieces, each of which provide sub-patterns which are matched (respectively) in the context of the `Principal`, `Right`, `Resource`, and `Condition` of the target `Grant` *g*, along with an optional fifth piece which is matched in the context of *g* as a whole. The overall `GrantPattern` *G* is considered to successfully match against the target `Grant` *g* if and only if each of the five pieces which may be present in *G* successfully match against their respective context.
The first piece of a `GrantPattern`, which is optional, contains either a literal `Principal`, or several patterns for a `Principal`. If a literal `Principal` *p* is provided, then the target `Grant` *g* must contain as its `principal` an element that is equal to *p*. If patterns for a `Principal` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Principal` from the target Grant *g*, must successfully match.
The second piece of a `GrantPattern`, which for technical reasons is not optional, contains either a literal `Right`, or several patterns for a `Right`. If a literal `Right` *r* is provided, then the target `Grant` *g* must contain as its `right` an element that is equal to *r*. If patterns for a `Right` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Right` from the target `Grant` *g*, must successfully match. Note that although this second piece of a `GrantPattern` is required, a pattern of the form

```
<rightPattern/>
```

can be used to match any `Right`.
The third piece of a `GrantPattern`, which is optional, contains either a literal `Resource` *R*, or several patterns for a `Resource`. If a literal `Resource` is provided, then the target `Grant` *g* must contain as its `resource` an element that is equal to *R*. If patterns for a `Resource` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Resource` from the target `Grant` *g*, must successfully match.
The fourth piece of a `GrantPattern`, which is optional, contains either a literal `Condition` *c*, or several patterns for a `Condition`. If a literal `Condition` is provided, then the target `Grant` *g* must contain as its `Condition` an element which is equal to *c*. If patterns for a `Condition` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Condition` from the target `Grant` *g*, must successfully match.
The fifth piece of a `GrantPattern` is also optional. If present, then it is an `AnXmlExpression` that, when evaluated in a target context of a new XML document containing the whole target `Grant` *g*, must successfully match.

5.1.4.7 GrantGroupPattern

Much as `GrantPattern`s provide a structured way to match against `Grant`s, `GrantGroupPattern`s provide a structured way to match against `GrantGroup`s. Let *G* be a `GrantGroupPattern`, and let *g* be a target `GrantGroup` against which one wishes to attempt to match *G*. *G* consists of possibly several pieces. The overall `GrantGroupPattern` *G* is considered to successfully match against the target

`GrantGroup` *g* only if each of the pieces which may be present in *G* successfully match against their respective context.

The first piece of a `GrantGroupPattern`, which is optional, contains either a literal `Principal`, or several patterns for a `Principal`. If a literal `Principal` *p* is provided, then the target `GrantGroup` *g* must contain as its `principal` an element that is equal to *p*. If patterns for a `Principal` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Principal` from the target `GrantGroup` *g*, must successfully match.

The second piece of a `GrantGroupPattern`, which is optional, contains either a literal `Condition` *c*, or several patterns for a `Condition`. If a literal `Condition` is provided, then the target `GrantGroup` *g* must contain as its `condition` an element that is equal to *c*. If patterns for a `Condition` are provided, then each such pattern, when evaluated in a target context of a new XML document containing only the `Condition` from the target `GrantGroup` *g*, must successfully match.

The third piece of a `GrantGroupPattern` consists of a sequence of sub-patterns, each of which is either a literal `Grant` or pattern for a `Grant`, or a literal `GrantGroup` or a pattern for a `GrantGroup`. Each literal or pattern in this sequence, when evaluated in the context of a new XML document containing only the corresponding `Grant` or `GrantGroup` from the sequence thereof at the end of the target `GrantGroup` *g*, must successfully match. In doing so, sub-patterns which are `Grant`s or `GrantGroup`s are, as one would by now expect, to match elements which are equal to themselves. Further, the sequence of `Grant`s and `GrantGroup`s at the end of *g* can be no longer than that sequence in *G*.

The fourth piece of a `GrantGroupPattern` is also optional. If present, then it is an `AnXmlExpression` that, when evaluated in a target context of a new XML document containing just the whole target `GrantGroup` *g*, must successfully match.

5.1.5 Variable Definition and Referencing

A particularly powerful and useful construct in `Grant`s and `GrantGroup`s is the definition and use of variables therein. With variables, a single `Grant` or `GrantGroup` can be written (and thus can be issued or otherwise authorized) that allows some carefully controlled variation and flexibility in the rights actually conveyed.

5.1.5.1 Variable Definition

Variables are defined using universal quantification as embodied in the `forAll` element.

Let *f* be a `forAll` element. The `varName` attribute of *f* indicates the name of the variable being defined. The elemental contents of *f* are zero or more patterns which determine what the variable *f/@*`varName` binds to.

If *x* is any XML element, let *d(x)* be a new XML document containing the element *x* as the root. Define *m(x)* to be the boolean function which is true if and only if all of the patterns in *f*, when evaluated in a context of *d(x)*, successfully matches. Let *B(f)* be that subset of the universe *X* of XML elements such that *m(b)* is true for every *b* in *B(f)* and is false for every *b'* in *X*-*B(f)*(note that this implies that if *f* contains no patterns that *B(f)* is the entire universe *X*). The set of *bindings* of the variable *f/@*`varName` is then defined to be the set *B(f)*.

The element *f* has a *scope* within which the variable it defines may be referenced. Colloquially, that scope is the rest of the parent element in which *f* is contained, less the scope of any other `forAll` element therein which happens to (re)declare the same variable. More precisely, let *N(y)* be that set of XPath nodes selected by the XPath (XPath) location path:

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

when evaluated with *y* as the contextual XPath node. For a `forAll` element *z*, let *O(z)* be that set of XPath nodes selected by location path:

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

(where the XML Namespace prefix `r` is bound to the REL core namespace) when evaluated with *z* as the contextual XPath node and `$fVarName` as the value of *z*/@`varName`.
Let *P(f)* be the union over all *w* in *O(f)* of *N(w).* Then the scope of *f* is defined to be *N(f)* less *P(f)*.
The set *S(f)* of the *eligible bindings* of the variable *f*/@`varName`, then, is defined to be that subset of *B(f)* such that *s* in *B(f)* is in *S(f)* if and only if for all elements *t* in the scope of *f* where *t*/@`varRef` equals *f*/@`varName` all of the following hold:

1. Either the expanded element name of *s* must exactly match that of *t* or *s* must be substitutable for *t* using substitution groups (that is, *t* is the head of a substitution group in which *s* resides).
2. Either the type of *s* must exactly match that of *t* or the type of *s* must derive (through any number of levels) from the type of *t* using type derivation.
3. If *t* is removed from its document and replaced with a copy of *s*, that document is (still) valid.

5.1.5.2 Variable Referencing

Variables are referenced using the `varRef` attribute of `LicensePart`s. Let *t* be a `LicensePart`, and suppose *t*/@`varRef` exists. Then it is required that *t* must be an empty element: from a conceptual perspective, the contents of *t* are determined by the binding of the variable that it references, not from local elements.
Moreover, the value in *t*/@`varRef` MUST be the name of some variable *v* whose scope includes *t*.

5.1.6 Conceptually Abstract

Certain elements and types in REL are designated as conceptually abstract. Conceptually abstract elements and types are used solely for the substitution heads and type bases, respectively, in XML Schema and, as such, do not refer to any concrete concepts. Common examples of concrete concepts are found in REL and include the `keyHolder` element and the `KeyHolder` type, though additional useful concrete concepts can be defined in extensions to REL. While a conceptually abstract element MAY appear with a concrete type and a concrete element MAY appear with a conceptually abstract type, a conceptually abstract element MUST NOT appear with a conceptually abstract type except in the form of a variable reference, as described in the preceding section.

5.1.7 Grant

A `Grant` is an XML structure that expresses an assertion that some **`Principal`** may exercise some **`Right`** against some **`Resource`**, subject, possibly, to some **`Condition`**. This structure is at the heart of the rights-management and authorization-policy semantics that REL is designed to express.

Especially in situations such as content-management scenarios, it is likely to be common practice that one `License` contain several `Grant`s to the same `Principal` pertaining to the same `Resource`, but differing in the specific `Right` being authorized. One `grant` might authorize a `play` right, while another might authorize a `print` right, for example. In other situations, such as those that might mirror the semantics of X.509 certificates, a set of `Grant`s in a `License` might share a `Principal` and a `Right` (perhaps the `PossessProperty` right), but differ in the `Resource` identified. In all such scenarios, it is expected that the syntactic mechanism of license parts, perhaps together with the use of the `inventory` in the `License`, will be often used to reduce verbosity and to increase the readability of the collective set of `Grant`s.

5.1.7.1 Grant/forAll

At the start of each `Grant` may reside an optional sequence of `forAll` elements. Because of the pattern matching facility therein, this powerful mechanism allows one authorized `Grant` instance to in fact authorize what would otherwise have to be authorized as a set of `Grant`s, a task which may be cumbersome or logistically impossible to actually carry out.

The effect of these `forAll` elements on the semantics of a `Grant` is straightforward. Let *g* be a `Grant` that contains at least one `forAll` child element, and let *f* be the first such child in *g*. Let *S(f)* be the set of eligible bindings of the variable *f*/@`varName`. For each *s* in *S(f)*, let *g'(s)* be a `Grant` which is equal to a copy of *g* except

1. (the copy of) *f* is not present in *g'(s)*, and
2. throughout the scope of *f* in *g*, all elements containing references to the variable *f*/@`varName` are replaced in *g'(s)* by *s*.

Then, to say that *g* is authorized means that for all such *s*, *g'(s)* is authorized.

Definition: a `Grant` which lacks any `forAll` children elements (or any constructs that are equivalent thereto, such as an `ExistsRight` condition with a `GrantPattern`) is considered *primitive*.

5.1.7.2 Grant/principal

The element in an instance of a `Grant` that validates against the `principal` particle thereof identifies the `Principal` that, under the authority of the `Issuer` of the `License`, may exercise the `Right` identified in the `Grant`.

While the `principal` particle of `Grant` is optional within the schema (primarily for the utility this provides to `GrantGroup`s), it is semantically very dangerous to in fact *authorize* a `Grant` which contains no `Principal` that validates against the `principal` particle. An authorized `Grant` which contains no `Principal` element is considered to be equivalent to an authorized `Grant` that contains an `allPrincipals` with zero children, which in turn authorizes the `Grant` to any entity that is authenticated as at least zero `Principal`s -- in short, any entity.

5.1.7.3 Grant/right

The element in an instance of a `Grant` which validates against the `right` particle thereof identifies what the `Issuer` of the containing `License` authorizes the indicated `Principal` to actually do.

5.1.7.4 Grant/resource

Many (but not all) `Right`s that might be issued are intended to be directed at and authorized against some particular target or `Resource`. For example, a content-management-related `Right` which authorizes a `Principal` to `print` must somehow identify exactly what digital resource the `Issuer` of the `License` intends may be printed. In REL, this target can be identified using the `resource` of a `Grant`. This is accomplished by providing in the `Grant` instance an element which validates against the `resource` particle thereof.

5.1.7.5 Grant/condition

`Issuer`s who authorize `Grant`s often desire the ability to somehow limit or constrain the situations in which the `Grant` may actually be used. The `condition` particle within a `Grant` provides a means by which this may be accomplished. If omitted, then no conditions are imposed: the authorized `Grant` may be used unconditionally. If a `Condition` is present, then the semantic obligations associated with the semantics of that particular `Condition` must be satisfied with respect to the indicated `Grant` before it may be used as the basis of an authorization decision.

5.1.7.6 Grant/delegationControl

Whenever a `Grant` is issued, the `Issuer` may optionally indicate in addition that the `Grant` may be delegated to others. This is accomplished by including in the `Grant` an element of type `DelegationControl`; absent such a `DelegationControl` element, a `Grant` is not (formally) delegable.
To say that an authorized `Grant` *g* is delegable means that the `Issuer` of *g* also authorizes every `Grant` *g′* where:

1. *g′*/`forAll`, *g′*/`delegationControl`, and *g′*/`condition` are all absent,
2. *g′*/`principal` is equal to *g*/`principal`,
3. *g′*/`right` is equal to the `issue` element
4. g'/`resource` is equal to a `Grant` g" where
   a. the (possibly empty) sequence of `forAll` elements that begins *g* appear as a prefix of the sequence of `forAll` elements that begins *g″*
   b. *g″*/`delegationControl` is compatible with *g*/`delegationControl`
   c. *g″*/`principal` is one of the allowable destination principals of *g*/`delegationControl`
   d. *g″*/`right` is equal to *g*/`right`
   e. *g″*/`resource` is equal to *g*/`resource`
   f. *g″*/`condition` is either equal to *g*/`condition`, or, if *g*/ `delegationControl/additionalConditionsProhibited` is absent, is equal to the equivalent of an `allConditions` element which contains at least *g*/`condition` (if present)

Additional policies which control the circumstances under which *g* is legally delegable are expressed by the semantics embodied in the `DelegationControl` element; these are explained in detail below. It is to be understood that *g* may be encrypted, and that in such situations the constraints listed here are to be adhered to by the clear-text form of *g*.

5.1.7.7 Grant/encryptedGrant

A mechanism is provided by which the contents of individual `Grant`s may be encrypted and so hidden from view from inappropriate parties. This mechanism makes straightforward use of XML Encryption Syntax and Processing (XML Encryption). Specifically, the XML content model of a `Grant` is a choice between a sequence containing the elements previously described in this section and an `encryptedGrant` element. `encryptedGrant` is of type `EncryptedContent` and represents the encryption of the contents of the `Grant` element.

5.1.8 GrantGroup

Within the REL architecture, `GrantGroup`s occupy much the same niche as do their more straightforward cousins, `Grant`s. That is, wherever a `Grant` may legally appear, it is (usually) the case that a `GrantGroup` may appear instead, where a `GrantPattern` may appear, a `GrantGroupPattern` may take its place, and so on. Indeed, from a point of view of the set of rights actually authorized, the semantics of a `GrantGroup` can be (and indeed are) specified in terms of the set of rights authorized by a particular set of related `Grant`s. However, from a point of view of pattern matching and inseparability under delegation, issuance, etc., `GrantGroup`s provide additional expressive power not otherwise found in `Grant`s.

5.1.8.1 GrantGroup/forAll

At the start of each `GrantGroup` may reside an optional sequence of `forAll` elements. Because of the pattern matching facility therein, this powerful mechanism allows one authorized `GrantGroup` instance to in fact authorize what would otherwise have to be authorized as a set of `GrantGroup`s, a task which may be cumbersome or logistically impossible to actually carry out.

The effect of these `forAll` elements on the semantics of a `GrantGroup` is straightforward. Let *g* be a `GrantGroup` that contains at least one `forAll` child element, and let *f* be the first such child in *g*. Let *S(f)* be the set of eligible bindings of the variable *f*/@`varName`. For each *s* in *S(f)*, let *g'(s)* be a `GrantGroup` which is equal to a copy of *g* except

1. (the copy of) *f* is not present in *g'(s)*, and
2. throughout the scope of *f* in *g*, all elements containing references to the variable *f*/@`varName` are replaced in *g'(s)* by *s*.

Then, to say that *g* is authorized means that for all such *s*, *g'(s)* is authorized.

5.1.8.2 GrantGroup/principal and GrantGroup/condition

Having indicated what it means to say that a `GrantGroup` containing a `forAll` element has been authorized, it remains to be specified what it means to say that a `GrantGroup` which *lacks* any `forAll` element has been authorized. Let *g* be such a `GrantGroup` lacking a `forAll` element, and consider the structure of *g*, which, as is evident in the REL core schema, can be thought of as a sequence containing:

1. an optional `DelegationControl` element *d*,
2. an optional `Principal` element *p*,

3. an optional `Condition` element *c*,
4. one or more contained `Grant` or `GrantGroup` elements *g′*.

To say that *g* has been authorized, then, means the following:

1. Consider each such *g′* in *g* where *g′* is a `Grant`. Let *p′* and *c′* be (respectively) the (possibly absent) `principal` and (possibly absent) `condition` contained in *g′*. Let *g″* be a `Grant` which is equal to *g′* except that
   a. within *g″*, *p′* is replaced by an element equivalent to an `allPrincipals` element *p″* which in turn contains
      i. *p* (if present)
      ii. *p′* (if present)
   b. within *g″*, *c′* is replaced by an element equivalent to an `allConditions` element*c″* which in turn contains
      i. *c* (if present)
      ii. *c′* (if present)

   Then to say that the `GrantGroup` *g* is authorized means that the `Grant` *g′* is authorized.
2. Similarly, consider each such *g′* in *g* where *g′* is a `GrantGroup`. Let *p′* and *c′* be (respectively) the (possibly absent) `principal` and (possibly absent) `condition` contained in *g′*. Let *g″* be a `GrantGroup` which is equal to *g′* except that
   a. within *g″*, *p′* is replaced by an element equivalent to an `allPrincipals` element *p″* which in turn contains
      i. *p* (if present)
      ii. *p′* (if present)
   b. within *g″*, *c′* is replaced by an element equivalent to an `allConditions` element*c″* which in turn contains
      i. *c* (if present)
      ii. *c′* (if present)

   Then to say that the `GrantGroup` *g* is authorized means that the `GrantGroup` *g′* is authorized.

The set of authorized `Grant`s which is related to the authorized `GrantGroup` *g* by means of exhaustive recursive application of Rules (1) and (2) is known as the set of *descendent* `Grant`s of *g*.

5.1.8.3 GrantGroup/delegationControl

Whenever a `GrantGroup` is issued, the `Issuer` may optionally indicate in addition that the `GrantGroup` may be delegated to others. This is accomplished by including in the `GrantGroup` an element of type `DelegationControl`; absent such a `DelegationControl` element, a `GrantGroup` is not (formally) delegable.

To say that an authorized `GrantGroup` *g* is delegable means that the `Issuer` of *g* also authorizes every `Grant` *g′* where:

1. *g′*/`forAll`, *g′*/`delegationControl`, and *g′*/`condition` are all absent,

2. *g'*/`principal` is equal to *g*/`principal`,
3. *g'*/`right` is equal to the `issue` element
4. *g'*/`resource` is equal to a `GrantGroup` *g"* where
   a. the (possibly empty) sequence of `forAll` elements that begins *g* appear as a prefix of the sequence of `forAll` elements that begins *g"*
   b. *g"*/`delegationControl` is compatible with *g*/`delegationControl`,
   c. *g"*/`principal` is one of the allowable destination principals of *g*/`delegationControl`
   d. *g"*/`condition` is either equal to *g*/`condition`, or, if *g*/`condition`/`additionalConditionsProhibited` is absent, is equal to the equivalent of an `allConditions` element which contains at least *g*/`condition` (if present)
   e. the `Grant`s and `GrantGroup`s contained as immediate children of *g"* are copies of those contained as immediate children of *g*.

Additional policies which control the circumstances under which *g* is legally delegable are expressed by the semantics embodied in the `DelegationControl` element; these are explained in detail below. It is to be understood that *g* may be encrypted, and that in such situations the constraints listed in this section are to be adhered to by the clear-text form of *g*.

5.1.8.4 GrantGroup/encryptedGrantGroup

A mechanism is provided by which the contents of a `GrantGroup` may be encrypted and so hidden from view from inappropriate parties. This mechanism makes straightforward use of XML Encryption Syntax and Processing (XML Encryption). Specifically, the XML content model of a `GrantGroup` is a choice between a sequence containing the elements previously described in this section and an `encryptedGrantGroup` element. `encryptedGrantGroup` is of type `EncryptedContent` and represents the encryption of the contents of the `GrantGroup` element.

5.1.9 DelegationControl

The use of elements of type `DelegationControl` provides the means by which policies which control and otherwise constrain the delegation of `Grant`s and `GrantGroup`s can be expressed.
Some such policies, namely those regarding constraints on delegated-to `Principal`s and whether additional `Condition`s may be present in delegated `Grant`s and `GrantGroup`s, were described previously herein. Other policies may be defined in types which are derived from the type `DelegationControl`.

5.1.9.1 Allowable Destination Principals

Part of the policy expressed by a `DelegationControl` element *d* is the set of allowable `Principal`s to whom the `Grant` or `GrantGroup` to which *d* is applied may be delegated.
If *d*/`to` is absent, then the set of *allowable destination principals* of *d* is the universe of all `Principal`s.
Otherwise, at least one *d*/`to` is present.
Let *z* be a `DelegationControl` that contains at least one `forAll` child element, and let *f* be the first such child in *z*. Let *S(f)* be the set of eligible bindings of the variable *f*/@`varName`. Let <u>*D*</u> be the universe of `DelegationControl` elements. Let *D(z)* be that

subset of <u>D</u> where *z'* in <u>D</u> is in *D(z)* if and only if there exists an *s* in *S(f)* so that *z'* is equal to a copy of *z* except

1. (the copy of) *f* is not present in *z'* and
2. throughout the scope of *f* in *z*, all elements containing references to the variable *f*/@`varName` are replaced in *z'* by *s*.

Now, consider a function *P* defined on the domain <u>D</u>. For any *z* in <u>D</u>, let *P(z)* be defined as follows:

1. If *z* has at least one `forAll` child element, then *P(z)* is the union, over all elements *z'* of the set *D(z)*, of *P(z')*.
2. If *z* does not have at least one `forAll` child element, then *P(z)* is that set whose members are the `Principal`s found in the `to` elements that are found in *z*.

Then the set of *allowable destination principals* of *d* is that set *P(d)*.

5.1.9.2 Compatibility of DelegationControl Elements

Let *d* and *d'* be `DelegationControl` elements. *d'* is said to be *compatible* with *d* if they are equal except for the following variations:

1. If *d*/`infinite` is present, then *d'*/`maxDepth` may be present (with any nonnegative value)
2. If *d*/`maxDepth` is present, then *d'*/`maxDepth` must be present, and must contain any nonnegative value which is less than the value contained in *d*/`maxDepth`.
3. If *d*/`additionalConditionsProhibited` is absent, then *d'*/`additionalConditionsProhibited` may be present.
4. If *d*/`to` is absent, then any number of *d'*/`to` may be present and identify any `Principal`s.
5. If at least *n d*/`to`'s are present where *n*>1, then any *n-1* of them may be omitted in *d'*.
6. If at least one *d*/`to` is present, then *d'*/`to` may contain any `Principal` which is equivalent to an `allPrincipals Principal` containing *d*/`to`/`principal` and zero or more arbitrary other `Principal`s.

Notice that "is compatible with" is an antisymmetric and transitive relationship.

5.1.10 EncryptedContent

`EncryptedContent` modifies the semantics of `enc:EncryptedDataType`, its base type, by simply restricting the use of the `enc:Type` attribute therein to be the value `http://www.w3.org/2001/04/xmlenc#Content`, which is the type associated with encrypting XML element content (XML Encryption). Thus, once decrypted, the plaintext of an element of type `EncryptedContent` is intended to semantically replace the `EncryptedContent` and thus become the content of said element's parent. In doing so, it must of course conform to the schema of the parent as a whole.

5.2 Core Principals

5.2.1 Principal

Within REL, instances of the type `Principal` (or a derivation thereof) represent the unique identification of an entity involved in the granting or exercising of rights. In a conceptual sense, they represent the "subject" that is permitted to carry out the action involved in exercising the `Right`.

The actual element `principal` is conceptually abstract. Also, the actual type `Principal` is conceptually abstract. That is, it does not indicate how a particular principal is actually identified and authenticated. Rather, this is carried out in types which are derivations of `Principal`. Such derived types may be defined in extensions to REL in order, for example, to provide a means by which `Principal`s who are authenticated using some proprietary logon mechanism may be granted certain `Right`s using the REL `License` mechanism. That said, two such derivations are important enough and central enough to be defined within the REL core itself.

5.2.2 The AllPrincipals Principal

Structurally, an `AllPrincipals Principal` is a simple container of zero or more other `Principal`s. Semantically, an `AllPrincipals` *a* represents the logical conjunct of the `Principal`s represented by all of its children. That is, *a* represents the set of its children *acting together* as one holistic identified entity. For example, if *a* is identified in some `Grant` as that `Principal` which must sign a certain bank loan application, then, conceptually, it is being required that each of the children of *a* act together as co-signers of the loan application.
A corollary of this definition is that an `AllPrincipals Principal` which contains zero children requires no particular `Principal` to act together as part of the entity that is identified, and thus the entire universe of entities is identified by such an empty `AllPrincipals Principal`. Where permitted by the schema in which it is used, such an empty `AllPrincipals Principal` is equivalent to said `Principal` in fact being absent. Note that there is no requirement that a normalization of an `AllPrincipals Principal` be carried out. That is, it is perfectly legal for an `AllPrincipals Principal` to contain other `AllPrincipals Principal`s.

5.2.3 The KeyHolder Principal

Instances of a `KeyHolder Principal` represent entities which are identified by their possession of a certain cryptographic key. For example, using a `KeyHolder`, a `Principal` which uses public-key cryptography may be conceptually identified as "that `Principal` which possesses the private key that corresponds to this-here public key." (Indeed, identification of `Principal`s in such a manner is expected to be very common).
This specification of REL does not itself specify the means by which the key relevant to a `KeyHolder` is identified. Rather, the `info` element (which is of type `dsig:KeyInfo`) within the type `KeyHolder` is defined by REL as the mechanism by which such information is conveyed, and the XML-Signature Syntax and Processing specification then specifies the means by which such conveyance is carried out.

5.3 Core Rights

5.3.1 Right

Within REL, instances of the type `Right` (or a derivation thereof) represent a "verb" that a `Principal` may be authorized to carry out under the authority conveyed by some authorized `Grant`. Typically, a `Right` specifies an action (or activity) that a `Principal` may perform on or using some associated target `Resource`. The semantic specification of each different particular kind of `Right` SHOULD indicate which kinds of `Resource` (if any) may be legally used in authorized `Grant`s containing that `Right`.

The actual element `right` is conceptually abstract. Also, the actual type `Right` is conceptually abstract. That is, the type `Right` itself does not indicate any actual action or activity that may be carried out. Rather, such actions or activities are to be defined in types which are derivations of `Right`. Such derived types will commonly be defined in extensions to REL, particularly those rights which are germane to a particular application domain. However, several `Right`s exist which are related to the domain of the REL core itself, and so are defined within the REL core.

5.3.2 The Issue Right

When an `Issue` element is used as the `right` in an authorized `Grant` *g*, it is required that *g*/`resource` against which the `Right` is applied in fact be a `Grant` or `GrantGroup` *g'*. The `Grant` *g* then conveys the authorization for the `Principal` *g*/`principal` to `Issue` *g'*; that is, it conveys the authorization, under the authority of the `Issuer` of the `License` *l* within which *g* is authorized, for *g*/`principal` to `Issue` other `License`s *l'* within which *g'* is authorized.

Use of the `Issue Right` is one of the basic mechanisms (along with delegation and trust of a `License` by some externally specified means) by which the REL Authorization Algorithm chains its processing from one `License` to another.

Those familiar with the X.509 certificate infrastructure will recognize that, in analogy, the `Principal` *g*/`principal` found in an authorized Grant *g* containing the `Issue Right` can conceptually be considered a "Certificate Authority."

At the instant a `License` is issued, the `Issue Right` must be held by the `Issuer` of the `License` with respect to all the `Grant`s and `GrantGroup`s directly authorized therein.

5.3.3 The Revoke Right

The authorized act of exercising the `Revoke Right` by a `Principal` *p* effects a retraction of a `dsig:Signature` that was previously issued (either by *p* or by some other `Principal` from which *p* received appropriate authorization to `Revoke`) and thus accomplishes a withdrawal of any authorization conveyed by that `dsig:Signature`.

There is, of course, commonly a latency, possibly a significant one, between the discovery of an issued `dsig:Signature` by some party wishing to rely on the authorization so conveyed and the subsequent discovery by that party of a later retraction thereof. In the interim, the relying party can and will consider the `dsig:Signature` as valid and binding.

Every `Issuer` of a `License`, by the act of affixing its `dsig:Signature` thereto, is implicitly and automatically authorized in a freely delegable manner to subsequently `Revoke` that `dsig:Signature`, should it choose to do so. By explicit use of the `Revoke Right`, an `Issuer` may convey that authorization to other `Principal`s of its choosing.

Although the REL core requires that when the `Revoke Right` is used that the associated `Resource` explicitly identify the to-be-revoked `dsig:Signature` in question, the core itself does not define a concrete XML data type by which this can be accomplished, instead choosing to leave such definitions to extensions of the core. The REL Standard Extension, though, does define the `Resource Revocable` which is useful in this role.

At the instant at which a `dsig:Signature` is formally revoked, the `Revoke Right` must be held by the revoking `Principal` with respect to the `dsig:Signature` being revoked.

5.3.4 The PossessProperty Right

The use of the `PossessProperty Right` within authorized `Grant`s allows the `Issuer`s thereof to straightforwardly express the fact that they authorize the association of property-like characteristics with certain `Principal`s. Put another way, the `PossessProperty Right` represents the `Right` for the associated `Principal` to claim ownership of a particular characteristic, which is listed as the `Resource` associated with this `Right`.
The `PossessProperty Right` imposes only two restrictions on the `Resource` with which it may be used within an authorized `Grant`:

- That that `Resource` is a `PropertyAbstract` and
- That that `PropertyAbstract` MUST NOT be omitted.

The REL core does not itself define any `PropertyAbstract`s which are particularly useful for use with the `PossessProperty Right`. However, several such `PropertyAbstract`s are defined within the REL Standard Extension; in particular, it defines several `PropertyAbstract`s which are useful for modeling the authorized binding of names to `Principal`s as is done in the X.509 certificate infrastructure.
Use of the `PossessProperty Right` is also very convenient in modeling notions of "group membership" found (among other places) in security systems of traditional operating systems. In this paradigm, in an REL extension one invents a `PropertyAbstract` *t* whose associated semantic is "is member of group". Then, straightforwardly, one issues `License`s with authorized `Grant`s that contain the `Right possessProperty` and the `PropertyAbstract` *t* in order to indicate that the associated `Principal` is in fact a member of the group.

5.3.5 The Obtain Right

When an `Obtain` element is used as the `Right` in an authorized `Grant` *g*, the `Resource` contained in *g* MUST be present and MUST either be a `Grant` or a `GrantGroup`. Let *g′* be that `Grant` or `GrantGroup`. Then the semantics conveyed by the authorization of *g* is that the `Issuer` thereof promises that the `Principal` *g*/`principal` can in fact obtain an issued version of *g′*, subject *only* to the limitation that *g*/`principal` must first satisfy the (possibly absent) `Condition` *g*/`condition`.
The means and manner by which such obtaining of *g′* is actually carried out is outside the scope of this specification, though `exerciseMechanism` provides a convenient way to bound this process. Additionally, it is instructive to note that, in practice, `Principal`s issuing and reading `Obtain Grant`s will likely want to use a `fulfiller` condition to indicate and determine the `Principal` who will `Issue` the resulting `Grant`.
The use of the `Obtain Right` can be conceptualized as an "offer" or "advertisement" for the "sale" of the contained `Grant`.

5.4 Core Resources

5.4.1 Resource

Continuing our grammatical analogy, an instance of type `Resource` (or a derivation thereof) represents the "direct object" against which the "subject" `Principal` of a `Grant` has the `Right` to perform some "verb." It should be noted that not all REL `Right`s make use of such target `Resource`s, just as not all verbs require direct objects.

The actual element `resource` is conceptually abstract. Also, the actual type `Resource` is conceptually abstract. That is, the type `Resource` itself does not indicate any actual object against which a `Right` may be carried out. Rather, such target objects are to be defined in types which are derivations of `Resource`. Such derived types will commonly be defined in extensions to REL, particularly those `Resource`s which are germane to a particular application domain. However, several `Resource`s exist which related to the domain of the REL core itself and so are defined within the REL core

5.4.2 DigitalResource

Use of a `DigitalResource` `Resource` in a `Grant` provides a means by which an arbitrary sequence of digital bits can be identified as being the target object of relevance within the `Grant`. Specifically, and importantly, such bits are not required to be character strings which conform to the XML specification, but may be arbitrary binary data. Conceptually, an instance *d* of `DigitalResource` defines an algorithm by which a sequence of bits *b* in question is to be *located*. The means by which this is accomplished breaks down into several cases:

1. The bits *b* are to be physically present within *d*. There are two sub-cases:

    a. If *b* is a character string which is a sequence of zero or more XML elements, then *b* MAY be represented using the `anXml` element within *d*, which is a simple container of arbitrary XML elements.

    b. Otherwise, *b* SHOULD be encoded in base64 and located within *d* by use of the `binary` element. Note that there is no requirement that a *b* which may be legally represented using the `anXml` element in fact be represented as such; base64 encoding may equally well be used, even for XML elements.

2. The bits are to be physically located at some external location outside of *d*. Perhaps, for example, they are located somewhere else within the XML document within which *d* is found, or perhaps at a location on a Web site. There are again two sub-cases:

    a. Though the bits may be external, *d* may still wish to indicate the *exact actual sequence* of bits being referred to. This is accomplished with use of the `secureIndirect` element.

    b. Otherwise, *d* wishes only to indicate the algorithm used to locate the bits, but is comfortable with the fact that differing actual executions of the algorithm may yield different sequences of bits. This is indicated by the use of the `nonSecureIndirect` element.

3. The means by which the bits are located is something else which is defined in an extension to REL. This is indicated within *d* by the use of an element which validates against the `xsd:any` particle therein.

The `secureIndirect` element straightforwardly makes use of the cryptographically-secure referencing mechanism designed as part of the XML Signature Syntax and Processing standard, specifically the type `dsig:ReferenceType` defined therein. The documentation of the semantics and processing associated with that type are not described in the present specification but rather are found in the specification of that standard.

The `nonSecureIndirect` element makes use of an REL-defined type `NonSecureReference`. The structure and attendant semantics of the `NonSecureReference` type are identical in every way to that of the aforementioned `dsig:ReferenceType` except that

1. `NonSecureReference` structurally lacks the `dsig:DigestMethod` and `dsig:DigestValue` elements found in `dsig:ReferenceType`, and

2. The processing semantics within `dsig:ReferenceType` that are associated with these two elements (in order to verify that the bits retrieved during the processing of the reference were exactly those expected) are omitted.

5.4.2.1 Authorization of Located Bits

Let *g* be any authorized `Grant` containing a `Resource` *d* which is a `DigitalResource`. Let *b* be the sequence of bits which is the result of any execution of the location algorithm of *d*. Then the Grant *g'* which is identical to *g* except that *d* is replaced by a `DigitalResource` which contains a child binary element which contains a base64 encoding of *b* is also authorized.

5.4.3 PropertyAbstract

An instance of type `PropertyAbstract` (or a derivation thereof) represents some sort of property that can be possessed by `Principal`s via `PossessProperty`.
The actual element `propertyAbstract` is conceptually abstract. Also, the actual type `PropertyAbstract` is conceptually abstract. That is, the type `PropertyAbstract` itself does not indicate any actual property that can be possessed. Rather, such target properties are to be defined in types which are derivations of `PropertyAbstract`. Such derived types will commonly be defined in extensions to REL.

5.5 Core Conditions

5.5.1 Condition

Within REL, instances of the type `Condition` (or a derivation thereof) represent a grammatical "terms & conditions" clause that a `Principal` must satisfy before it may take advantage of an authorization conveyed to it in a `Grant` containing the `Condition` instance. The semantic specification of each different particular kind of `Condition` MUST indicate the details of the terms, conditions, and obligations that use of the `Condition` actually imposes. When these requirements are fulfilled, the `Condition` is said to be *satisfied*.
When a particular `Condition` is used within an authorized `Grant`, REL processing systems that process the `Grant` MUST honor the request implied thereby that the terms, conditions, and obligations indicated in the semantic specification of the `Condition` be satisfied by the `Principal` indicated in the `Grant` before the `Grant` may be used as the basis of an authorization decision. A corollary of this requirement is the observation that should an REL processing system in the course of honoring such a request encounter a `Condition` defined in some REL extension of which it lacks semantic knowledge, the processing system MUST NOT consider the `Condition` to be satisfied.
The actual element `condition` is conceptually abstract. Also, the actual type `Condition` is conceptually abstract. That is, the type `Condition` itself does not indicate the imposition of any actual term or condition. Rather, such terms and conditions are to be defined in types which are derivations of `Condition`. Such derived types will commonly be defined in extensions to REL, particularly those `Condition`s which are germane to a particular application domain. However, several `Condition`s exist which are related to the domain of the REL core itself, and so are defined within the REL core.

5.5.2 The AllConditions Condition

Structurally, an `AllConditions` is a simple container of zero or more other `Condition`s. Semantically, the `AllConditions` represents a logical conjunct of the `Condition`s

represented by all of its children. That is, the Conditions imposed by each and every of these children must be satisfied in order for the AllConditions to be satisfied.
A corollary of this definition is that an AllConditions Condition which contains zero children is considered always to be satisfied. It is thus equivalent to the empty AllConditions Condition being absent.
Note that there is no requirement that a normalization of an AllConditions Condition be carried out. That is, it is perfectly legal for an AllConditions Condition to contain other AllConditions Conditions.

5.5.3 The ValidityInterval Condition

A ValidityInterval Condition indicates a contiguous, unbroken interval of time. The semantics of the Condition expressed is that the interval of the exercise of a Right to which a ValidityInterval is applied must lie wholly within this interval. The delineation of the interval is expressed by the presence, as children of the Condition, of up to two specific fixed time instants:

1. the optional notBefore element, of type xsd:dateTime, indicates the inclusive instant in time at which the interval begins; if absent, the interval is considered to begin at an instant infinitely distant in the past

2. the optional notAfter element, also of type xsd:dateTime, indicates the inclusive instant in time at which the interval ends; if absent, the interval is considered to end at an instant infinitely distant in the future.

5.5.4 The RevocationFreshness Condition

As was discussed previously, Issuers of REL Licenses may in a License indicate the means by which they will, should they later decide to Revoke their dsig:Signature, post notice of such revocation. As a practical matter, many if not most of the mechanisms used for such dissemination of revocation information involve a periodic polling on the part of REL processing systems to determine whether new revocation information is available. With such polling necessarily comes a latency of information dissemination. Use of a RevocationFreshness Condition in a Grant or GrantGroup can place an upper bound on the size of this polling latency whenever the Grant or GrantGroup is used as part of an authorization decision.
If a RevocationFreshness Condition found in an authorized Grant or GrantGroup *g* contains a maxIntervalSinceLastCheck element, and the length of the duration *d* indicated therein is greater than zero, then in order for the Condition to be satisfied, the length of real, wall-clock time that has elapsed between

1. the last time that the dsig:Signature on the License *l* in which *g* was authorized was polled to check for revocation, and

2. the time at which *l* is passed as a relevant input License to the REL Authorization Algorithm

must be less than or equal to *d*. If the length of such duration *d* is zero, then in order for the Condition to be satisfied, a poll to check for revocation must be carried out each and every time *l* is passed as a relevant input License in a non-recursive call to the REL Authorization Algorithm. The length of the duration *d* MUST NOT be less than zero.
A RevocationFreshness Condition containing a noCheckNecessary element is defined to be semantically equivalent to what a RevocationFreshness Condition containing a maxIntervalSinceLastCheck element with an infinite duration would signify, but for the fact that the XML Schema xsd:duration data type cannot express such infinite

durations of time. This policy is an explicit affirmation that revocation need not ever be explicitly polled, in contrast to an omitted `RevocationFreshness` condition, which leaves the tolerable polling latency to be determined by other means.

5.5.5 The ExistsRight Condition

5.5.5.1 Some Grants Containing ExistsRight Conditions Are Not Primitive

Let *c* be a `Condition` of type `ExistsRight`, and let *g* be a `Grant` containing *c*. Suppose *c*/`grantPattern` or *c*/`grantGroupPattern` exists, and let *e* be this element.
Then, as was previously mentioned, *g* is not primitive.
Define the `Grant` *g'* as being a copy of *g* except for the transformations defined as follows:

1. an additional new `forAll` element *f* is inserted at the end of the (possibly empty) sequence of `forAll` elements that begins *g'*, where
2. *f*/@`varName` contains a new variable name which is different from the name of any other variable defined within *g', and*
3. the contents of *f* is the element *e*, and
4. the element *e* within *c* is replaced with an empty (respectively) `grant` or `grantGroup` element which contains a reference to the variable named in *f*/@`varName`.

If *g* is authorized, then *g'* is also authorized.

5.5.5.2 Satisfaction of ExistsRight

Let the functions *P* and *Q*, and the notation *allPrincipals(P)* be as defined in the REL Authorization Algorithm. Let t_0 be the present time.
Let *c* be an `ExistsRight` condition returned from a call to the REL Authorization Algorithm whose inputs were *(p, r, t, v, L, R, C, T)*. It follows that either *c*/`grant` or *c*/`grantGroup` exists; let *h* be that element. Then, in order for *c* to be satisfied,

1. If *c*/`trustedIssuer` exists, it must be established that there exists a time instant *i* prior to *v* and a `Principal` *p'* from those that conform to the policy articulated within the element c/`trustedIssuer` such that *P(p')* is a subset of $Q(h, i, v, L, C, t_0)$.
2. If c/`trustedIssuer` does not exist, it must be established that there exists a time instant *i* prior to *v* for which the call to the REL Authorization Algorithm with inputs:

   ($allPrincipals(Q(h, i, v, L, C, t_0))$, the `issue` element, *h, i, L, R, C, T* union {*h*})

   either

   a. returns ***yes***, or

   b. returns ***maybe*** together with a set *C'* of `Condition`s, and at least one `Condition` *c'* in *C'* can be shown (possibly with the help of *C*) to have been satisfied during *i* with respect to this issuance.

5.5.6 The PrerequisiteRight Condition

The `PrerequisiteRight Condition` is related to the `ExistsRight Condition`, but they differ in many respects. While the `ExistsRight Condition` deals with determining if certain `Grant`s and `GrantGroup`s are directly and correctly authorized by some `trustedIssuer`, the `PrerequisiteRight Condition` deals with determining that (under the authorization of some `trustedIssuer`) a given `Principal` has a given `Right` to a given `Resource` subject to either no `Condition` or a `Condition` that can be shown to be satisfied.

5.5.6.1 Satisfaction of PrerequisteRight

Let the functions *P* and Q, and the notation *allPrincipals(P)* be as defined in the REL Authorization Algorithm. Let t_0 be the present time.
Let *c* be a `PrerequisiteRight Condition` returned from a call to the REL Authorization Algorithm whose inputs were *(p, r, t, v, L, R, C, T)*. Then, in order for *c* to be satisfied, it must be shown that there exists some `Grant` or `GrantGroup` *h* such that

1. If *c*/`trustedIssuer` exists, it must be established that there exists a time instant *i* prior to *v* and a `Principal` *p'* from those that conform to the policy articulated within the element c/`trustedIssuer` such that *P(p')* is a subset of *Q(h, i, v, L, C,* t_0*)*.

2. If c/`trustedIssuer` does not exist, it must be established that there exists a time instant *i* prior to *v* for which the call to the REL Authorization Algorithm with inputs:

   (*allPrincipals(Q(h, i, v, L, C,* t_0*))*, the `issue` element, *h, i, L, R, C, T* union {*h*})

   either

   a. returns ***yes***, or

   b. returns ***maybe*** together with a set C' of `Condition`s, and at least one `Condition` *c'* in *C'* can be shown (possibly with the help of *C*) to have been satisfied during *i* with respect to this issuance.

3. There exists a primitive `Grant` *g* such that *g*/`principal` equals *c*/`principal` (or both are absent), *g*/`right` equals *c*/`right`, *g*/`resource` equals *c*/`resource` (or both are absent), the authorization of *h* implies the authorization of *g*, and *g*/`condition` is shown (possibly with the help of *C*) to have been satisfied with respect to the aforesaid algorithm inputs.

5.5.7 The Fulfiller Condition

A `Fulfiller Condition` allows one to specify that the exercise of certain `Right`s that require some other `Principal` to perform some duty (such as the `obtain Right`) are permitted only if that other `Principal` that provides fulfillment is the specified `Principal`.
A `Fulfiller Condition` is satisfied if and only if the `Principal` fulfilling the exercise is the one specified therein. If there is no fulfilling `Principal` (for instance, if one isn't required) the `Fulfiller Condition` is considered not satisfied.
The `Fulfiller Condition` is particularly useful when used in conjunction with the `obtain Right`. For instance, in a superdistribution scenario, users may be permitted to `obtain grant`s from one particular distributor. When that distributor signs and issues those `grant`s, it is acting as the fulfiller for the `obtain Right`. It is important to note, however, that `Fulfiller` has other uses as well. For instance, a physician (as an agent in a health insurance plan) may permit a patient to get medicine, but only if fulfilled by a particular in-network pharmacist.

5.5.8 The ExerciseMechanism Condition

An `ExerciseMechanism Condition` allows one to limit the way in which a `Right` is exercised.

An `ExerciseMechanism` `Condition` is satisfied if and only if the mechanism of exercising is the one specified therein. The type `ExerciseMechanism` defines two ways to specify this mechanism:

1. `exerciseService` specifies a service to use to effect the exercise.
2. `xsd:any` allows others to specify other mechanisms.

The `ExerciseMechanism` `Condition` is particularly useful when used in conjunction with the `obtain` `Right`. For instance, in a superdistribution scenario, it is common for the users who wish to exercise an `obtain` `Right` to be very removed from the original channels of distribution. An `ExerciseMechanism` could direct the user back to an official distribution channel. It is important to note that `ExerciseMechanism` has other uses as well. For instance, 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. Or, an airline company may permit its frequent flyers to ticket for a reduced fare when ticketing via a particular online travel service.

5.6 Other Core Types and Elements

5.6.1 TrustedPrincipal

Elements of type `TrustedPrincipal` (or a derivation thereof) indicate a policy by which `Principal`s are identified as having the appropriate and necessary qualifications in order to be trusted for use in certain situations (see, for example, the use of `TrustedPrincipal` in the `ExistsRight` `Condition`).
Within `TrustedPrincipal`, this policy is indicated in one of two ways:

1. If the element `TrustedPrincipal/principal` is present, then the set of identified `Principal`s is exactly that one `Principal`.
2. If the element `TrustedPrincipal/any` is present, then the set of identified `Principal`s is any of the `Principal`s contained therein.

It is often usefully the case that the `Principal`s within a `TrustedPrincipal` contain references to variables which denote a set of `Principal`s by means of a pattern within a `forAll` element.

5.6.2 ServiceReference

The term *service* as used in this specification refers to an active body of software, execution of which is distinguished from that of *client* software which wishes to make use of it.
It is the role of an instance of `ServiceReference` to indicate the location and the means and manner by which a client is to interact with a specific service. Specifically, a `ServiceReference` instance does the following:

1. Identifies the location or *address* at which the service is found.
2. Identifies a greater or lesser amount of *metadata* about the semantics of the service and the rules that must be adhered to by a client that interacts with it.
3. Optionally specifies a set of concrete *parameters* that are to be provided when a client interacts with the service by dereferencing this particular `ServiceReference`. These parameters provide a means by which a service might at run time distinguish between its uses from different REL contexts.

5.6.2.1 ServiceDescription

REL does not itself invent significant new infrastructure for describing services; rather, it draws on the considerable work being done in this area by others. To do this, the REL core defines a conceptually abstract element, `serviceDescription`, and a conceptually abstract type, `ServiceDescription`. A `ServiceDescription` provides the location and metadata information for a `ServiceReference`. The standard extension defines three concrete `ServiceDescription`s for utilizing the following two technologies within `ServiceReference`s:

1. WSDL, the Web Services Definition Language (WSDL), and
2. UDDI, the Universal Description, Discovery, and Integration (UDDI) directory infrastructure.

5.6.2.2 Parameters

Let *r* be a `ServiceReference`. Then *r* may contain an ordered sequence of contextual parameters which, per the metadata associated with the service, may be necessary in order to successfully interact with the it. Such parameters may be specified using the sequence contained within the *r*/`serviceParameters` element.
r/`serviceParameters` contains a sequence of pairs of elements. Each pair contains a `datum` element and an optional `transforms` element. Each such `datum` element defines a raw parameter for the service. This raw parameter may be processed to form an actual parameter for the service by applying the sequence of transformations to the raw parameter optionally indicated in the accompanying `transforms` element (if no such transformations are indicated, then the actual parameter is the same as the raw parameter). The specification of the sequence of transformations to be carried out makes use of a mechanism designed as part of XML Signature Syntax and Processing (XML Digital Signature), specifically the type `dsig:TransformsType` defined therein. The documentation of the semantics and processing associated with that type are found in the specification of that standard, but the following modifications are made thereto:

1. The input to the first `dsig:Transform` is a raw parameter, manifest as an XPath node-set containing the one raw parameter element (that is, the child of the `datum` element) in-place in the context of its XML document (thus navigation from the parameter node to elsewhere in the XML document containing the parameter is feasible).
2. The output of the last `dsig:Transform` is the corresponding actual parameter.

`ServiceReference` parameter transformation is defined to take place after all `LicensePart` and variable reference processing has occurred. The use of the parameter transformation facility is in fact particularly convenient in order to be able to discern and communicate to the service the result of such other processing actions. The actual interpretation, detailed processing, and passing to the service of the sequence of actual parameters is necessarily service-specific, and is thus not defined here.

5.6.3 LicenseGroup

Instances of the type `LicenseGroup` are simple and straightforward containers of `License`s. No inherent semantic is conveyed by the presence of two particular `License`s within the same `LicenseGroup`. This type exists due merely to the observation that it is often handy and convenient to be able to use such a container in XML instances and schemas. No use of it is made in the remainder of REL.

5.7 The REL Authorization Algorithm

At the heart of any implementation of software which makes an authorization decision using REL `License`s in the decision-making process lies a central question "Is such-and-such a `Principal` authorized to exercise such-and-such a `Right` against such-and-such a `Resource`?" The "Authorization Algorithm" illustrates how that question can be answered by applying the semantics defined in this part of ISO/IEC 21000.
It is important to understand that the Authorization Algorithm works in terms of potentialities. That is, it colloquially answers the question "If the principal wanted to ..., could he?". A question which is quite a different one is "The principal is about to ...; can he?" The former question addresses a potentiality that might later come to pass; the latter question carries with it the implication that the `Principal` has already committed itself to try to carry out the act. This difference in perspective may be subtle, but could have important implications as to the details of how and when the evaluation of certain kinds of `Condition`s are carried out.
It is also important to understand that the algorithm operates on clear-text `License`s, `Grant`s, and `GrantGroup`s. Encrypted forms of these are to be treated as if they were actually their clear-text equivalent.
Finally, it is important to understand that the approach by which the specification of the Authorization Algorithm in this section is described and documented is by no means intended to be the best or most efficient manner in which the algorithm can in fact be *implemented*. It is, rather, merely the most succinct and straightforward exposition that the authors of this specification found to communicate the essential details of the algorithm.

5.7.1 Input to the Authorization Algorithm

The Authorization Algorithm takes a number of pieces of information as input:

1. A `Principal` ***p***, which is the identity of the entity whose authorization to perform an act is being called into question,

2. A `Right` ***r***, which embodies the semantics of the action to be performed or otherwise carried out,

3. An (optional) `Resource` ***t***, which is the target of the action ***r*** being carried out by ***p***,

4. An interval ***v*** of time during which the execution of ***r*** by ***p*** is considered to take place. This may either be an instantaneous point in time, or may be a contiguous, unbroken interval of time.

5. A set ***L*** of relevant `License`s. The algorithm will attempt to find authorized `Grant`s and `GrantGroup`s within these `License`s that it can use to establish a basis for an affirmative authorization decision,

6. An additional set ***R*** of "root" `Grant`s that are considered by the algorithm to be authorized under the authority of an omnipotent issuer. These are authorized `Grant`s that are to be trusted by some decision making process that is outside of the scope of REL itself.

7. A (possibly empty) set ***C*** of other appropriate contextual information. This contextual information is not processed or manipulated directly by the core Authorization Algorithm, and the details of such information are not herein specified, but its existence is established in order to clearly allow for the provision of additional contextual information necessary to evaluate authorization decisions based on `Principal`s, `Right`s, `Resource`s, and `Condition`s that might be defined in extensions to REL.

8. A set ***T*** of traversed `Grant`s and `GrantGroup`s. This set is used to ensure that the Authorization Algorithm terminates. The `Grant`s and `GrantGroup`s in this set have already

been traversed by parent recursive calls to the algorithm. As such, their authorization should be considered not provable in child calls, and no further recursion should be carried out in an attempt to prove their authorization.

This input can be considered as an eight-tuple:

(***p, r, t, v, L, R, C, T***)

5.7.2 Output of the Authorization Algorithm

The output of the Authorization Algorithm is either:

1. The result ***no***, indicating that the Algorithm could not establish that the `Principal` had the indicated authorization, or

2. Either

   a. the result ***yes***, indicating that the Algorithm established that the `Principal` unequivocally has the indicated authorization, or

   b. the result ***maybe*** together with a non-empty set of alternative `Condition`s, indicating that the `Principal` has the indicated authorization provided that at least one of the indicated alternative `Condition`s is satisfied.

It is important to notice that the core Authorization Algorithm herein described does not itself consider whether or not any particular `Condition` has in fact been satisfied with respect to the input authorization request; such processing and evaluation is (from a specification perspective at least) left to higher level algorithms of the REL processing system which consumes the output of the Authorization Algorithm. That said, in the chaining steps of the Authorization Algorithm, where recursive use of the algorithm is made, such evaluation of `Condition`s output from the recursion is indeed carried out; however, it is there done with respect to rights involved in the authority to `issue` REL `License`s in the input set ***L*** (a `Right` which has been exercised), not the input `Right` ***r*** being requested by the input `Principal` ***p*** (a `Right` that may only *potentially* be exercised).

5.7.3 Execution of the Authorization Algorithm

The execution of the Authorization Algorithm proceeds as follows. We begin with the definition of several important concepts.

Let

- ***<u>P</u>*** be the universe of `Principal`s,
- ***<u>C</u>*** be the universe of `Condition`s,
- ***<u>G</u>*** be the universe of `Grant`s,
- ***<u>GG</u>*** be the universe of `GrantGroup`s,
- ***<u>I</u>*** be the universe of time instants,
- ***<u>V</u>*** be the universe of time intervals
- ***<u>L</u>*** be the universe of `License`s
- ***<u>CC</u>*** be the universe of Authorization Algorithm input contexts

Let ***<u>H</u>*** be the union of ***<u>G</u>*** and ***<u>GG</u>***.

Consider a function *P* defined on the domain ***<u>P</u>*** union ***<u>H</u>***. For any *p* in ***<u>P</u>***, let *P(p)* be defined as follows:

1. If *p* is of type `AllPrincipals`, then *P(p)* is the union, over all children *p'* of *p*, of *P(p')*.
2. If *p* is not of type `AllPrincipals`, then *P(p)* is the one-element set containing *p*.

Colloquially, *P(p)* is the set of `Principal`s obtained by collapsing any `AllPrincipals` elements in *p*. Similarly, for any *h* in ***H***, let *P(h)* be defined as follows:

1. If *h*/`principal` is absent, *P(h)* is the empty set
2. If *h*/`principal` is not absent, *P(h)* is defined to be *P(h*/`principal`*)*

Colloquially, *P(h)* is the set of `Principal`s, acting together, to whom a `Grant` or `GrantGroup` is issued.
Let *S* be any finite subset of ***P***. Then, let the notation *allPrincipals(S)* denote an `allPrincipals` element which contains as children exactly the elements of *S*.
Let *PG* be that subset of ***G*** where *g* in ***G*** is in *PG* if and only if *g* is primitive. Let *EPG* be that subset of *PG* where *g* in *PG* is in *EPG* if and only if:

1. *P(g)* is a subset of *P(**p**)*,
2. *g*/`right` is equal to **r**,
3. either *g*/`resource` is equal to ***t*** or both are absent

EPG can be considered the set of "eligible" primitive `Grant`s.
Let *LH* be that subset of ***H*** where *h* in ***H*** is in *LH* if and only if there exists a `License` *l* in ***L*** in which *h* is directly authorized. Let *ULH* be that subset of *LH* where *h* in *LH* is in *ULH* if and only if *h* is not in ***T***. *ULH* can be considered the set of "usable licensed `Grant`s and `GrantGroup`s."
We define a notion for the set of `Principal`s that have directly authorized a `Grant` or `GrantGroup` prior to a certain time instant. Let *Q* be the function with domain ***H*** x ***I*** x ***V*** x ***L*** x ***CC*** x ***I*** and range in ***P*** which is defined as follows: For any *h* in ***H***, *i* and *t* in ***I***, *v* in ***V***, *L* a set of `License`s, and *C* an authorization context, if *p* is in ***P***, then *p* is in *Q(h, i, v, L, C, t)* if and only if there exists a `License` *l* in *L* such that

1. *h* is directly authorized within *l*
2. *l* is issued by *p*, and such issuance is not known to have been revoked as of the minimum of times *t* and the end of *v*
3. *p* can be demonstrated to have issued *l* prior to *i* by means of:
    a. a trusted (according to the context *C*) counter-signature for the signature of *p* on *l* attesting to this fact,
    b. *i* being greater than the time *t*
    c. any other method using *C*

We consider the subset of the usable licensed `Grant`s and `GrantGroup`s which are in fact authorized. Let t_0 be the time at which the execution of the Authorization Algorithm occurs. Let *AULH* be that subset of *ULH* where *h* in *ULH* is in *AULH* if and only if there exists a *i* in ***I*** prior to the start of ***v*** for which a recursive call to the Authorization Algorithm with inputs

(*allPrincipals(Q(h, i, **v**, **L**, **C**, t_0))*, the `issue` element, *h*, *i*, ***L***, ***R***, ***C***, ***T*** union {*h*})

either

1. returns ***yes***, or

2. return ***maybe*** together with a set *C′* of `Condition`s, and at least one `Condition` *c* in *C′* can be shown (possibly with the help of ***C***) to have been satisfied during *i* with respect to this issuance.

Let *AEPG* be the set of affirmatively authorized eligible primitive `Grant`s defined as follows: *g* in *EPG* is in *AEPG* if and only if there exists an *h* in (*AULH* union ***R***) such that the authorization of *h* implies the authorization of *g*.
If *AEPG* is empty, the Authorization Algorithm returns ***no***.
If there exists a *g* in *AEPG* such that *g*/`condition` is equivalent to an `AllConditions Condition` that has no children, then the Authorization Algorithm returns ***yes***.
Otherwise, the Authorization Algorithm returns ***maybe*** together with a set *C* of `Condition`s, where *C* is that subset of ***C*** where *c* in ***C*** is in *C* if and only if there exists a `Grant` *g* in *AEPG* with *g*/`condition` equal to *c*.
This concludes the specification of the Authorization Algorithm.

6 REL Standard Extension

6.1 Right Extensions: RightUri

The standard extension schema defines a type called `RightUri`. A `RightUri` indicates a right using a URI rather than an XML Schema element or type. The semantics of the right being indicated by a `RightUri` *r* are determined by the URI value of *r*/@`definition`.

6.2 Resource Extensions

6.2.1 Property Extensions

6.2.1.1 PropertyUri

The standard extension schema defines a type called `PropertyUri`. A `PropertyUri` indicates a property using a URI rather than an XML Schema element or type. The semantics of the property being indicated by a `PropertyUri` *t* are determined by the URI value of *t*/@`definition`.

6.2.1.2 Name

A property indicating a name from some name space.
`Name` is an extension of the type `PropertyAbstract`, and, as such, can be used along with the `PossessProperty Right` to associate a `Name` with a `Principal`. (This is useful for modeling the X.509 certificate-like binding of names to principals). Such associations allow (other) `Grant`s to be made to, colloquially speaking, `Principal`s described by their `Name`s.
Both the element `name` and type `Name` are conceptually abstract.

6.2.1.3 Name Extensions

6.2.1.3.1 EmailName

An Internet email address (per rfc822/rfc2822) associated with the entity.

6.2.1.3.2 DnsName

A name in the DNS name space, with trailing period omitted. For example, "xyz.com"

6.2.1.3.3 CommonName

A name by which an entity is colloquially known. Intended to be used as the CN name part from X400.

6.2.1.3.4 X509SubjectName

The subject name of some X509 certificate associated with the entity. Intended to address legacy interoperability issues involving X509 certificates.

6.2.1.3.5 X509SubjectNamePattern

A pattern that identifies a set of X509 subject names using pattern matching. `X509SubjectNamePattern` is not a derivation of the type `Name`. Rather, it is a derivation of the type `ResourcePatternAbstract`. It matches any `X509SubjectName` for which it is the root of the `X509SubjectName` tree. This element can be used to enforce constraints similar to the X509 specification.

6.2.2 Revocable

Identifies a `dsig:SignatureValue` that can be `Revoke`d. The `dsig:SignatureValue` can be identified literally or by reference. In the latter case, the result of dereferencing the reference must be of type `dsig:SignatureType`; the `dsig:SignatureValue` able to be `Revoke`d is the one `dsig:SignatureValue` therein.
A `Revocable` is a `Resource` for use with the `Revoke Right`.

6.3 Condition Extensions

The standard extension schema defines several extensions of the type `Condition`. Of these, six extensions require a notion of state. To facilitate the definition of these extensions, a type `StatefulCondition` is defined as well. `Condition` extensions are constructed either by extending the type `StatefulCondition` or by directly extending the type `Condition`. What it means for a `Condition` extension to be satisfied is left to its description.

6.3.1 StatefulCondition

Some `Condition`s may be tied to a notion of state. For example, the number of times some content may be rendered can be bounded by some value. To cover such usages, standard extension defines a type called `StatefulCondition`. The type is an extension of the type `Condition` defined in the core. It includes a child element `serviceReference`, which indicates a service to be used to manage state. The details of this interaction is necessarily service-specific and so is not discussed here.

6.3.2 StateReferenceValuePattern

A pattern that identifies a set of `ServiceReference`s using pattern matching by dereferencing their values.
A `StateReferenceValuePattern` *p* matches a `ServiceReference` if and only if it can be determined that the present state of the service can be represented by the children of *p*. The means for determining this is necessarily service-specific and so is not discussed here.
The element `stateReferenceValuePattern` is typically used with the `obtain` element. When the `Right` to `obtain` a `grant` with a `StatefulCondition` is issued, then (to give the user exercising the `obtain Right` an idea about the initial state a certain `serviceReference` will happen to have) a `stateReferenceValuePattern` is created to

make such an indication. This ensures that only `serviceReference`s in the proper state will appear in the obtained `grant`.

6.3.3 The ExerciseLimit Condition

Indicates a limit on the number of times that certain exercises may occur.
An `ExerciseLimit` is satisfied if the current exercise is one of *n* allowed exercises, where these allowed exercises (and hence the number *n*) are managed by the specified `serviceReference`.

6.3.4 The TransferControl Condition

Represents a constraint requiring ownership of some specified virtual token. A `TransferControl` is satisfied if the specified `serviceReference` indicates that the `Principal` performing the exercise is the current owner of the required virtual token (which is typically specified within the `serviceParameters` in a service-specific fashion, but MAY be known to the service by other means). Typically, the specified `serviceReference` MAY also be used to cause the transfer of the ownership of the virtual token to some other `Principal`.

6.3.5 The SeekApproval Condition

Indicates that the specified service must give its approval for this `Condition` to be satisfied.
A `SeekApproval` is satisfied if the specified `serviceReference` allows it to be.

6.3.6 The TrackReport Condition

Indicates that exercises must be tracked with a designated tracking service.
A `TrackReport` *c* is satisfied while either

- the exercise is considered tracked with the designated `serviceReference` or
- both of the following are true
    - *c*/`communicationFailurePolicy` is "lax" and
    - there is a state of communication failure with the tracking service.

6.3.7 The TrackQuery Condition

Represents a `Condition` based on some stateful integral value.
A `TrackQuery` is satisfied if the stateful integral value managed by the specified `serviceReference` is within the range as specified by the two child elements `notMoreThan` and `notLessThan`.
`TrackQuery` is commonly used to predicate the possibility of one exercise on the occurrence (or non-occurrence) of other exercises. In particular, a `trackReport` applied to one exercise might cause the value of a `trackQuery` applied to another exercise to increase by one, thus possibly changing the satisfaction of the `trackQuery`.

6.3.8 The ValidityIntervalFloating Condition

Represents an interval of allowed exercising that begins with the first exercise.
A `ValidityIntervalFloating` is satisfied during the interval of allowed exercising, where that interval (and hence both its start and duration) are managed by the specified `serviceReference`.

6.3.8.1 Other Forms of Validity Intervals

A `ValidityIntervalFloating Condition` is useful to express floating intervals that become fixed on first use. Often times (either for business model reasons or to minimize the amount of state keeping done by a compact device) it is also useful to be able to express floating intervals that get fixed during some `License` issuance step. For this reason, the standard extension defines two types for use in conjunction with `ValidityInterval` (from the core) to accomplish this goal. The `ValidityIntervalDurationPattern` pattern and `ValidityIntervalStartsNow Condition` are useful in this respect when placed on a `Grant` to `Issue`.

6.3.8.1.1 The ValidityIntervalDurationPattern Pattern

A `ValidityInterval` *v* matches a `ValidityIntervalDurationPattern` *d* if the duration of time represented by *v* is equal to *d*/`duration`.

6.3.8.1.2 The ValidityIntervalStartsNow Condition

A `ValidityIntervalStartsNow Condition` *c* is said to be satisfied at time *t* if all of the following hold:

- If *c*/`backwardTolerance` is present, then *c*/`validityInterval`/`notBefore` is present and greater than or equal to the result of *t* set backwards by the value *c*/`backwardTolerance`.
- If *c*/`forwardTolerance` is present, then *c*/`validityInterval`/`notBefore` is either
  - absent or
  - present and less than or equal to the result of *t* set forward by the value *c*/`forwardTolerance`.

6.3.9 The ValidityTimeMetered Condition

Represents a constraint on the cumulative exercise time over all exercises. Unlike `ValidityIntervalFloating`, `ValidityTimeMetered` deals with a length of time that is not necessarily contiguous. A `ValidityTimeMetered` is satisfied at time *t* if *t* lies within the intervals of time that make up the cumulative time duration, where those intervals are managed by the specified `serviceReference`. If the `quantum` child of a `ValidityTimeMetered` is specified, it specifies the minimum length that one can expect each such interval to have.

6.3.10 The ValidityTimePeriodic Condition

Indicates a validity time window that recurs periodically. For example, this condition can be used to express time windows such as "every weekend" or "the second week of every month".

6.3.10.1 ValidityTimePeriodic/start

A locally defined element of type `xsd:dateTime`. Indicates the start of the time, typically date, from which the periods designated in this right become meaningful.

6.3.10.2 ValidityTimePeriodic/period

A locally defined element of type `xsd:duration`. This indicates the frequency with which the exercise time window recurs.

6.3.10.3 ValidityTimePeriodic/phase

A locally defined element of type `xsd:duration`. This is used to indicate a period of latency before beginning each time window. When this value is positive then it directly specifies the duration of latency. When this value is negative, then the duration of latency is equal to the length of period minus the absolute value specified herein.

6.3.10.4 ValidityTimePeriodic/duration

A locally defined element of type `xsd:duration`. This indicates the actual length of the time window.

6.3.10.5 ValidityTimePeriodic/periodCount

A locally defined element of type `xsd:integer`. Indicates a bound on the number of time windows. This element is optional.

6.3.10.6 Satisfaction of the ValidityTimePeriodic Condition

Suppose *c* is a `ValidityTimePeriodic Condition` and let

- *s* be *c*/`start`,
- *p* be *c*/`period`,
- *h* be *c*/`phase`,
- *d* be *c*/`duration`, and
- *n* be *c*/`periodCount`

then *c* is satisfied at time *t* if $s + i*p + h <= t <= s + i*p + h + d$ for some *i* satisfying all of the following:

- if *h* is positive, then $i >= 0$,
- if *h* is negative, then $i >= 1$,
- if *h* is positive and *n* is not absent, then $i <= n - 1$, and
- if *h* is negative and *n* is not absent, then $i <= n$.

6.3.11 The Fee Condition

Indicates that there is a fee for the exercise.

6.3.11.1 Fee/paymentAbstract

This element of type `paymentAbstract` is abstract. It is meant to be substitutable. For more details and examples of extensions see `paymentAbstract`.

6.3.11.2 Fee/min

This locally defined optional element contains a `paymentAbstract` element. It specifies the minimum amount due. If the total amount paid is less than the value of this element, a new payment in the amount of the difference is due.

6.3.11.3 Fee/max

This locally defined optional element contains a `paymentAbstract` element. It specifies the maximum amount due. If the total amount paid is greater than the value of this element, a new credit in the amount of the difference is due. If the total amount paid is equal to the value of this element all other payments resulting from this `Fee` are void until the value of `max` increases.

6.3.11.4 Fee/to

This locally defined element is of type `AccountPayable`. It indicates the party to whom, and the means by which, payment is to be made. To allow for the rare cases where this is discovered from context, this element is left optional.

6.3.11.4.1 AccountPayable

`AccountPayable` is used to identify a party to whom one can transfer a sum of money, along with an identification of the means by which such a transfer is to take place. While there are undoubtedly many ways this can be done, two ways are explicitly defined. The type `AccountPayable` is defined as a choice between three options. The first, a `paymentService`, identifies the party to whom the payment is made. The second option, `aba`, identifies a bank in the US banking system. As a provision to support other forms of banking mechanisms, an option is made for specifying other banking means as well. This is realized by the `xsd:any` particle.

6.3.11.4.1.1 AccountPayable/paymentService

The locally defined element `paymentService` contains a `serviceReference`. This identifies the party to whom the payment is to be made, and the interface to the service indicates the necessary payment mechanism.

6.3.11.4.1.2 AccountPayable/aba

The locally defined element identifies an account within a US banking institution by means of conventions established by the American Banking Association. It defines two child elements, institution and account. The banking institution is identified by its nine-digit banking routing number using the institution. The account within that institution is identified with account.

6.3.11.5 Using the fee condition

A `Grant` can predicate an exercise on a `Fee`. Typically `Fee`s involve a payment and a designation of the party the payment is made to. Options about what form the payment should take (such as periodic or one-time or usage-based or metered) are reflected by the `paymentAbstract` element. A child element `to` of type `AccountPayable` is used to characterize both the payment mechanism and the payee.
There are two other optional elements. The optional `min` price specification indicates the minimum price to be paid if the right is exercised at all. The optional `max` price specification refers to the maximum price to be paid. When both `max` and `min` price specifications are given, the `max` price specification dominates.
Suppose that the `Fee`s in `paymentAbstract`, `min`, and `max` independently amount to *p*, *min*, and *max* respectively due for the exercise as defined by their respective `paymentAbstract` extensions. Then, let *x* be

- *min*, if *p* <= *min* < *max*;
- *p*, if *min* < *p* < *max*;
- *max*, if *max* <= *p* or *max* <= *min*.

Then the `Fee Condition` is satisfied if and only if the amount *x* is paid to the entity identified by `to` for this purpose.

Not all forms of `paymentAbstract` extensions are directly comparable. For the `Fee Condition` to be sensibly evaluated, it is necessary that the extensions of the `paymentAbstract` elements in `paymentAbstract`, `min`, and `max` are comparable. In the standard extension, the `paymentAbstract` extensions either contain `serviceReference`s or they do not. Those that do are considered stateful. Stateful `paymentAbstract` extensions are not comparable with stateless ones, so it is required that the `paymentAbstract` extensions in `paymentAbstract`, `min`, and `max` are either all stateful or all stateless. The corresponding `Fee Condition`s are respectively termed as stateful and stateless.

6.3.12 The Territory Condition

Indicates a constraint on the geographic or virtual space where the exercised may occur.

6.3.12.1 Territory/location

This element defines inline the optional elements `region`, `country`, `state`, `city`, `postalCode` and `street` that designate a physical location.

6.3.12.1.1 Territory/location/country

Specifies a country by qualified name. The qualified name can be any qualified name suitable for indicating a country. Clause 6.6 defines some such qualified names.

6.3.12.1.2 Territory/location/region

Specifies a country subdivision by qualified name. The qualified name can be any qualified name suitable for indicating a country subdivision. Clause 6.6 defines some such qualified names.

6.3.12.1.3 Territory/location/state

This element is a two-letter code for US states.

6.3.12.1.4 Territory/location/city

This element designates a city.

6.3.12.1.5 Territory/location/postalCode

This element designates a postal code (e.g. a zip code).

6.3.12.1.6 Territory/location/street

This element designates a street.

6.3.12.2 Territory/domain

This element defines inline an element `uri` to designate a digital location.

6.3.12.2.1 Territory/domain/uri

This element, has type `xsd:anyUri` and can be any URI that designates a digital location.

6.3.12.3 Satisfaction of the Territory Condition

Since it may be desirable to have an exercise occur in more than one location, physical or digital, the content model for `Territory` allows for an unbounded number of children.

A `Territory` is satisfied if the exercise can be shown to be occurring in at least one of the locations or domains.

6.4 PaymentAbstract and its Extensions

6.4.1 PaymentAbstract

The notion of payment is abstracted in the Standard Extension. The element `paymentAbstract` has type `PaymentAbstract`. Both are conceptually abstract.
In place of `PaymentAbstract`, more concrete forms of payment (`PaymentFlat`, `BestPriceUnder`, `CallForPrice`, `Markup`, etc.) are to be used. The standard extension defines seven different kinds of payments. Each of these defines its own notion of when a payment has been made that works toward a `Fee Condition` being satisfied, as described in the section on `Fee`.

6.4.2 Rate

A fixed amount of money in a designated currency. If the currency is not specified, then the default is USD.

6.4.2.1 Rate/amount

Specifies an amount (as a float number).

6.4.2.2 Rate/currency

Specifies a currency by qualified name. The qualified name can be any qualified name suitable for indicating a currency. Clause 6.6 defines some such qualified names.

6.4.3 PaymentFlat

Specifies a payment due upon exercising. The contained `serviceReference` is used to determine if the fee has already been paid and to keep record of payment. The contained `rate` specifies the monetory value of the payment to be made.

6.4.4 PaymentMetered

Specifies a payment due for each time interval during which an exercise *occurs*. The contained `rate` and `per` elements together specify the charge per period. The contained `by` element indicates the quantum by which time is measured for the computation of the amount. The contained `phase` element is used for rounding purposes; it indicates what portion of time may elapse before somone is billed for the whole duration defined with `by`.
The element `phase` is used for rounding the units of time that are smaller than the duration described by the `by` element: a value of zero would have the effect of rounding up while a value equal to the value of `by` would have the effect of rounding down.
Formally, after normalilzing all of the duration values to seconds, suppose the `rate` is *r*, `per` in seconds is *p*, `by` in seconds is *b*, and `phase` in seconds is *h*. Then, if the interval of exercise of the right is *t* seconds, then the payment due is given by the expression *r* * (*p*/*b*) * (*floor(t*/*b)* + *round*) where *round* is 1 if *t*%*b* is greater than *h* and is 0 otherwise.

6.4.5 PaymentPerInterval

Specifies a payment due for each time interval during which the *ability* to exercise is desired. The enclosed `serviceReference` is used to keep track of the time through which payment has been made. The contained `rate` and `per` elements together specify the charge per period.

6.4.6 PaymentPerUse

Specifies a payment due each time a right is exercised. The contained `rate` and `initialNumberOfUses` elements together specify the charge per package of uses. If `initialNumberOfUses` is absent, it defaults to 1. The `serviceReference` is used to keep track of the number of prepaid uses remaining.

6.4.7 BestPriceUnder

Specifies the maximum price that ultimately must be paid without specifying the ultimate price exactly. The ultimate price is determined through a later, unspecified settlement mechanism. While `max` overrides `min` if `max` is less than `min`, `min` overrides `BestPriceUnder` if `BestPriceUnder` is less than `min`. If an implementation cannot determine the ultimate price exactly, it is free to voluntarily take the payment contained herein as the ultimate price.

6.4.8 CallForPrice

Identifies an entity with whom a price must be negotiated before exercising. Any one of the specified `serviceReference`s can be used to negotiate a price. Unlike `BestPriceUnder`, price determination is required for `CallForPrice`.

6.4.9 Markup

Specifies a fee due each time some other fees are due. The `fraction` element indicates the fractional rate at which markup is calculated.
Let *m* be a `Markup`. Then *m* requires the payment of any *m*/`fee`s and *m*/`feeForResource`s which may be present therein. In addition, for each such payment, an additional payment equal to the product of that payment and the value *m*/`fraction` is due.
The presence of a `feeForResource` implies a context in which multiple resources are being used simultaneously. The total price for using the specified *m*/`feeForResource`/`resource` *t* is calculated (and such amount is paid, as defined by any `License`s for that *t*). The price is then marked up by the *m*/`fraction`, and the markup is paid as specified by the containing `Fee` element. If *t* is not used in conjunction with this exercise, then *m* is not fulfilled and the containing `Fee` is not satisfied.

6.5 ServiceDescription Extensions

6.5.1 WSDL

6.5.1.1 Background

Briefly (see WSDL for details), a WSDL language expression occurs in an WSDL `definitions` element. This element is a container of *`service`s*. Each WSDL `service` is a container of *`port`s*, each of which denotes a different aspect or sub-service of the `service`. Each `port` is associated with a particular abstract *`portType`* and also indicates a *`binding`* of that abstract `portType` to concrete message formats and protocol details. Within a `service`, all `port`s that share a `portType` are to be considered as semantically equivalent by clients. The linkage between `port`s and `portType`s, `port`s and `binding`s,

etc. is by name. Indeed, structurally, a `definitions` element is a physical container for `service`s, `binding`s, `portType`s, and (perhaps considerably) other metadata.
REL allows for two stylistically different approaches to using WSDL. Which of these two approaches is appropriate depends on the operational details and logistics of the context in which a given service might actually be used. In some situations one will be more useful, in different situations, the other will be.

6.5.1.2 WsdlComplete

Let *w* be a `WsdlComplete` element. In the first approach, the *w*/`wsdl` element, which is of type `DigitalResource`, is used to locate a WSDL `definitions` element *d*. The *w*/`service` element then indicates the name of a particular WSDL `service` *s* that is defined in *d*. Optionally, the *w*/`portType` element disambiguates which subset of the possibly many `port`s of *s* the `WsdlComplete` *w* intends to refer to.

6.5.1.3 WsdlAddress

Let *w* be a `WsdlAddress` element. In the second approach, the *w*/`kind`/`wsdl` element again locates a `definitions` element *d*. The element *w*/`kind`/`binding` then indicates the name of a WSDL `binding` which is defined within *d*. The `binding` in turn indicates the abstract `portType` of the `service`, together with a mapping to concrete message formats and protocol details. Remaining to be specified is a concrete endpoint or address at which the software executing the service may be found. This is indicated in the element *w*/`address`.

6.5.2 UDDI

6.5.2.1 Background

UDDI defines (see UDDI for details) the notion of a *`registry`* as a particular service replicated over a set of nodes. Each `registry` is a database or directory containing possibly many *`businessEntities`*. Each `businessEntity` contains possibly many *`businessServices`*. Each `businessService` has possibly several *`bindingTemplates`*, each of which may contain explicit endpoint information and also other arbitrary metadata about the `businessService` using data in the form of what are known as *`tModels`*. Several components of the UDDI data model have associated primary keys by which their instances are independently retrievable. These include `businessEntities`, `businessServices`, `bindingTemplates`, and `tModels`.
To uniquely identify a service which is specified using UDDI, one need only identify the `registry`, then identify the primary key of the `businessService` in question within that registry.

6.5.2.2 Uddi

Let *u* be a `Uddi`. If *u*/`registry` is omitted, then the registry in question is the Universal Business Registry (the UBR, which is publicly accessible on the Internet and operated by a consortium of companies including IBM, Microsoft, and others). If *u*/`registry` is present, then the value found therein indicates the name (note: not the *location*) of the registry to be used. This name is assumed to be drawn from a list of names of non-UBR UDDI registries known to and useful within the context of usage of the `License` in which *u* is found.
u/`serviceKey` indicates the primary key of the `businessService` in question within the identified registry. Depending on which version of UDDI is used, one of two different

types of primary key is appropriate. UDDI v1 and v2 use XOpen DCE UUIDs as keys; UDDI v3 and above allows for the use of URIs. Each is available as a choice under *u*/`serviceKey`.

6.6 Country, Region, and Currency Qualified Names

6.6.1 Namespace URI Structure

In order to support the consistent meaning of `License`s over time, countries, regions, and currencies are indicated by qualified names. Utilizing the facilities provided by ISO 3166 and ISO 4217, this part of ISO/IEC 21000 defines an algorithm for generating a number of such qualified names, grouped into a number of namespaces. Each of these namespaces is indicated by a URI of the form `urn:mpeg:mpeg21:2002:01-REL-SX-NS:`*`YYYY-MM-DD`*`:`*`TYPE`*, where *`YYYY-MM-DD`* indicates any day after or including 2003-07-25 and *`TYPE`*, which is either `country`, `region`, or `currency`, indicates the type of qualified names the namespace contains.

6.6.2 Country Qualified Names

The local part of the country qualified names defined in this part of ISO/IEC 21000 is the ISO 3166-1 country code. The country namespace `urn:mpeg:mpeg21:2002:01-REL-SX-NS:2003-07-25:country` contains one qualified name for each ISO 3166-1 country code in use on 2003-07-25. The other country namespaces defined by this part of ISO/IEC 21000 contain one qualified name for each ISO 3166-1 country code that is assigned a new meaning in an ISO publication on the respective date. The meaning of any country qualified name *q* defined in this part of ISO/IEC 21000 is the same as the meaning that the ISO 3166-1 country code corresponding to the local part of *q* had on the date corresponding to the namespace of *q*.

6.6.3 Region Qualified Names

The local part of the region qualified names defined in this part of ISO/IEC 21000 is the ISO 3166-2 region code. The region namespace `urn:mpeg:mpeg21:2002:01-REL-SX-NS:2003-07-25:region` contains one qualified name for each ISO 3166-2 region code in use on 2003-07-25. The other region namespaces defined by this part of ISO/IEC 21000 contain one qualified name for each ISO 3166-2 region code that is assigned a new meaning in an ISO publication on the respective date. The meaning of any region qualified name *q* defined in this part of ISO/IEC 21000 is the same as the meaning that the ISO 3166-2 region code corresponding to the local part of *q* had on the date corresponding to the namespace of *q*.

6.6.4 Currency Qualified Names

The local part of the currency qualified names defined in this part of ISO/IEC 21000 is the ISO 4217 currency code. The currency namespace `urn:mpeg:mpeg21:2002:01-REL-SX-NS:2003-07-25:country` contains one qualified name for each ISO 4217 currency code in use on 2003-07-25. The other currency namespaces defined by this part of ISO/IEC 21000 contain one qualified name for each ISO 4217 currency code that is assigned a new meaning in an ISO publication on the respective date. The meaning of any currency qualified name *q* defined in this part of ISO/IEC 21000 is the same as the meaning that the ISO 4217 currency code corresponding to the local part of *q* had on the date corresponding to the namespace of *q*.

6.7 The `matches` XPath Function

The REL standard extension defines a function for use with the XPath expression language to match regular expressions. Though this function is modeled after work carried out in the XPath2 (XPath2) design effort , no normative reference is made here to those drafts. The XPath (XPath) specification defines that the names of XPath library functions are namespace-qualified. To that end, the `matches` function is defined to reside in the REL standard extension namespace:

`http://www.xrml.org/schema/2002/05/xrml2sx.`

The syntax of this function is

```
sx:matches($input as string?, $pattern as string?) as boolean?
sx:matches($input as string?, $pattern as string?, $flags as string?) as
boolean?
```

The effect of calling the first version of this function (omitting the argument `$flags`) is the same as the effect of calling the second version with the `$flags` argument set to a zero-length string.

The function returns true if `$input` matches the regular expression supplied as `$pattern`; otherwise, it returns false.

If any of the arguments is an empty sequence, the result is an empty sequence.

Unless the metacharacters `^` and `$` are used as anchors, the string is considered to match the pattern if any substring matches the pattern. But, if anchors are used, the anchors must match the start/end of the string (in string mode), or the start/end of a line (in multiline mode).

Note: This is different from the behavior of patterns in XML Schema, where regular expressions are implicitly anchored.

An error is raised ("Invalid matches argument") if the value of `$pattern` or of `$flags` does not conform to the required syntax defined in clause 6.7.1.

Note: Regular expression matching is defined on the basis of Unicode code-points; it takes no account of collations.

6.7.1 Regular Expression Syntax

The regular expression syntax used by these functions is defined in terms of the regular expression syntax specified in Part 2 of XML Schema, which in turn is based on the established conventions of languages such as Perl. However, because XML Schema uses regular expressions only for validity checking, it has omitted some facilities that are widely-used with languages such as Perl, and this section therefore describes extensions to the XML Schema regular expressions syntax that re-instate these capabilities.

The regular expression syntax and semantics for these functions are identical to those defined in Part 2 of XML Schema with the following additions:

- Two modes are defined, string mode and multiline mode.
- Two meta-characters, `^` and `$` are added. In string mode, the metacharacter `^` matches the start of the entire string, while `$` matches the end of the entire string. In multiline mode, `^` matches the start of any line (that is, the start of the entire string, and the position immediately after a newline character), while `$` matches the end of any line (that is, the end of the entire string, and the position immediately before a newline character).

- In string mode, the metacharacter `.` matches any character whatsoever. In multiline mode, the metacharacter `.` matches any character except a newline character.
- Reluctant quantifiers are supported, specifically:
    - `X??` matches *X*, once or not at all
    - `X*?` matches *X*, zero or more times
    - `X+?` matches *X*, one or more times
    - `X{n}?` matches *X*, exactly *n* times
    - `X(n,}?` matches *X*, at least *n* times
    - `X{n,m}?` matches *X*, at least *n* times, but not more than *m* times

  The effect of these quantifiers is that the regular expression matches the shortest possible substring (consistent with the match as a whole succeeding). In the absence of these quantifiers, the regular expression matches the longest possible substring.
  To achieve this, the production in XML Schema:
  ```
  [4] quantifier ::= [?*+] | ( '{' quantity '}' )
  ```
  is changed to:
  ```
  [4] quantifier ::= ( [?*+] | ( '{' quantity '}' ) ) '?'?
  ```
- Sub-expressions (groups) within the regular expression are recognized. The regular expression syntax defined by XML Schema allows a regular expression to contain parenthesized sub-expressions, but attaches no special significance to them. Some functions described here allow access to the parts of the input string that matched a sub-expression (called captured substrings). The sub-expressions are numbered according to the position of the opening parenthesis in left-to-right order within the top-level regular expression: the first opening parenthesis identifies group 1, the second group 2, and so on. If a sub-expression matches more than one substring (because it is within a construct that allows repetition) then only the last substring that it matched will be captured.

Note: Reluctant quantifiers have no effect on the results of the boolean `sx:matches` function, since this is only interested in discovering whether a match exists, and not where it exists.
To enable conforming implementations to make use of existing regular expression library routines, this specification does not disallow extensions to the regular expression syntax described here. However, such extensions should only be provided if they conform to an existing recognized specification. All regular expressions that conform to the syntax described here must be accepted, and must implement the semantics described here.

6.7.2 Flags

`sx:matches` provides an optional parameter, `$flags`, to set options for the interpretation of the regular expression. The parameter is a string, in which individual letters are used to set options. The presence of a letter within the string indicates that the option is on, its absence indicates that the option is off. Letters may appear in any order and may be repeated. If there are letters present that are not defined here, then an error is raised ("Invalid regular expression syntax").
The following options are defined:

- `m`: If present, the match operates in multiline mode. Otherwise, the match operates in string mode.

- `i`: If present, the match operates in case-insensitive mode. Otherwise, the match operates in case-sensitive mode. The detailed rules for character matching in case-insensitive mode are implementation-dependent (and they may be locale-dependent).

6.7.3 Examples (Informative)

- `sx:matches("abracadabra", "bra")` returns true
- `sx:matches("abracadabra", "^a.*a$")` returns true
- `sx:matches("abracadabra", "^bra")` returns false

Given the source document:

```
<poem author="Wilhelm Busch"> Kaum hat dies der Hahn gesehen, Fängt er auch
schon an zu krähen: «Kikeriki! Kikikerikih!!» Tak, tak, tak! - da kommen
sie. </poem>
```

the following function calls produce the following results, with the `poem` element as the context node:

- `sx:matches(., "Kaum.*krähen")` returns true
- `sx:matches(., "Kaum.*krähen", "m")` returns false
- `sx:matches(., "^Kaum.*gesehen,$", "m")` returns true
- `sx:matches(., "^Kaum.*gesehen,$")` returns false
- `sx:matches(., "kiki", "i")` returns true

7 REL Multimedia Extension

7.1 Rights

7.1.1 Semantics

7.1.1.1 Modify

`Modify` represents the right to change a resource, preserving the alterations made.

With `Modify`, a single resource is preserved at the end of the process. Changes may include the addition and removal of elements of the original resource, including the embedding of other resources.

`Modify` may be used with `Condition`s requiring specific resource attributes of the resource to be preserved or changed. The specific resource attributes may be on a list or may be called out by using a list. Lists may be inclusive (for example, "Attributes a and b must be changed") or exclusive (for example, "Everything except attributes c and d must be changed"). Resource attributes that are not otherwise constrained by `Condition`s may be changed.

7.1.1.2 Enlarge

`Enlarge` represents the right to modify a resource by adding to it.

With `Enlarge`, a single resource is preserved at the end of the process. Changes may 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.

7.1.1.3 Reduce

`Reduce` represents the right to modify a resource by taking away from it.

With `Reduce`, a single resource is preserved at the end of the process. Changes may include only the removal of existing elements of the original resource.

7.1.1.4 Move

`Move` represents the right to relocate a resource from one place to another.

When `Move` is applied to a resource, at least the resource's location is changed.

`Move` may be used with `Condition`s requiring specific resource attributes of the resource to be preserved or changed. The specific resource attributes may be on a list or may be called out by using a list. Lists may be inclusive (for example, "Attributes a and b must be changed") or exclusive (for example, "Everything except attributes c and d must be changed"). Resource attributes that are not otherwise constrained by `Condition`s may be changed.

7.1.1.5 Adapt

`Adapt` represents the right to change transiently an existing resource to derive a new resource.

With `Adapt`, two distinct resources will exist at the end of the process, one of which is the original resource in unchanged form, and one which is newly made. Changes may include the addition and removal of elements of the original resource, including the embedding of other resources. Changes may 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.

`Adapt` may be used with `Condition`s requiring specific resource attributes of the resource to be preserved or changed. The specific resource attributes may be on a list or may be called out by using a list. Lists may be inclusive (for example, "Attributes a and b must be changed") or exclusive (for example, "Everything except attributes c and d must be changed"). Resource attributes that are not otherwise constrained by `Condition`s may be changed. Annex C shows an example of how `Adapt` may be used with `Condition`s to effect semantics generally known as "copy."

7.1.1.6 Extract

`Extract` represents the right to take a part out of an existing resource to derive a new resource.

With `Extract`, two distinct resources will exist at the end of the process, one of which is the original resource in unchanged form, and one which is newly made and whose content is equivalent to a part of the original resource. Changes may be made temporarily to the original resource in the course of the extract process, but such changes are not saved in the original resource at the end of the process.

`Extract` may be used with `Condition`s requiring specific resource attributes of the resource to be preserved or changed. The specific resource attributes may be on a list or may be called out by using a list. Lists may be inclusive (for example, "Attributes a and b must be changed") or exclusive (for example, "Everything except attributes c and d must be changed"). Resource attributes that are not otherwise constrained by `Condition`s may be changed.

7.1.1.7 Embed

`Embed` represents the right to put a resource into another resource.

The resource into which a resource is embedded may be pre-existing, or may be created by the act of combining this resource with one or more others. `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 `Adapt` and `Embed` are required.

7.1.1.8 Play

`Play` represents the right to derive a transient and directly perceivable representation of a resource.

`Play` may cover the making of any forms of transient representation that may be perceived directly (that is, without any intermediary process) with at least one of the five human senses. `Play` includes playing a video or audio clip, displaying an image or text document, or creating transient representations that may be touched, or perceived to be touched. When `Play` is applied to a digital resource, content may be rendered in any order or sequence according to the technical constraints of the digital resource and renderer.

`Play` is a render right.

7.1.1.9 Print

`Play` represents the right to derive a rixed and directly perceivable representation of a resource.

`Print` refers to the making of a fixed physical representation, such as hard-copy prints of images or text, that may be perceived directly (that is, without any intermediary process) with one or more of the five human senses.

`Print` is a render right.

7.1.1.10 Execute

`Execute` represents the right to execute a digital resource.

`Execute` refers to the primitive computing process of executing. `Execute` applies only to a digital resource.

7.1.1.11 Install

`Install` represents the right to follow the instructions provided by an installing resource.

An installing resource is a resource that provides instructions which when followed result in one or more resources that are new, or enabled, or both new and enabled.

7.1.1.12 Uninstall

`Uninstall` represents the right to follow the instructions provided by an uninstalling resource.

An uninstalling resource is a resource that provides instructions which when followed result in one or more resources that had previously been installed being disabled or deleted.

7.1.1.13 Delete

`Delete` represents the right to destroy a digital resource.

`Delete` applies only to digital resources. Delete is not capable of reversal. After a delete process, an "undelete" action is impossible.

7.1.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the types and their corresponding elements defined in the preceeding section.

7.2 Resources

7.2.1 Digital Item Resources

7.2.1.1 Semantics

7.2.1.1.1 DiReference

`DiReference` indicates a resource that is a Container, Descriptor, Digital Item, Component, or Fragment as declared within a Digital Item Declaration. Let *r* be a `DiReference` and let *i* be the value of *r*/`identifier`. Then the Resource indicated by *r* is the Container, Descriptor, Digital Item, Component, or Fragment identified by *i* according to ISO/IEC IS 21000-3 within a Digital Item Declaration.

7.2.1.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

7.3 Conditions

7.3.1 Resource Attribute Conditions

7.3.1.1 Semantics

7.3.1.1.1 RequiredAttributeChanges

A `RequiredAttributeChanges` *c* is said to be satisfied if the attributes of the resource that are changed during the exercise include all those attributes indicated by each of the resource attribute set definition elements (see Clause 7.4) that are children of *c*.

7.3.1.1.2 ProhibitedAttributeChanges

A `ProhibitedAttributeChanges` *c* is said to be satisfied if the attributes of the resource that are changed during the exercise exclude all those attributes indicated by each of the resource attribute set definition elements (see Clause 7.4) that are children of *c*.

7.3.1.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

7.3.2 Digital Item Conditions

7.3.2.1 Semantics

7.3.2.1.1 DiCriteria

A `DiCriteria` *c* is said to be satisfied if the Digital Item Declaration XML Document Subset that declares the Container, Descriptor, Digital Item, Component, or Fragment indicated by *c*/`diReference` satisfies each of the patterns *c*/`r:anXmlPatternAbstract`.

7.3.2.1.2 DiPartOf

A `DiPartOf` *p* is said to be satisfied if the Container, Descriptor, Digital Item, Component, or Fragment indicated by the first child of *p* is (through any number of levels) a part of the Container, Descriptor, Digital Item, Component, or Fragment indicated by the second child of *p*.

7.3.2.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

7.3.3 Marking Conditions

7.3.3.1 Semantics

7.3.3.1.1 IsMarked

An `IsMarked` *m* is satisfied if *m*/`r:resource` is already marked as described by the children of *m*.

7.3.3.1.2 Mark

A `Mark` *m* is satisfied if, during the course of the exercise, *m*/`r:resource` becomes marked as described by the children of *m*.

7.3.3.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

7.3.4 Security Conditions

7.3.4.1 Semantics

7.3.4.1.1 Source

Indicates the source secured repository or device to use when exercising a right. Used with all rights except for render rights.

7.3.4.1.2 Destination

Indicates the repositories to which a work can be moved. Used with rights that involve movement of digital works (all rights except render rights).

7.3.4.1.3 Helper

Indicates the software that can be used during an exercise. A `Helper` *h* is satisfied if every piece of software helping in the exercise has a `Principal` description that either

- Is equal to one of the *h*/`principal` elements, or
- Matches each of the patterns contained within one of the *h*/`wildcard` elements.

7.3.4.1.4 Renderer

Identifies the device that can be used to render a work. Used with render rights.

7.3.4.1.5 ResourceSignedBy

A `ResourceSignedBy` *c* is satisfied if and only if some `dsig:Signature` *s* is found such that all of the following are 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` *r* such that all of the following are true:
    - *c*/`dsig:Transforms` is equal to *r*/`dsig:Transforms` (or both are absent),
    - *c*/`dsig:DigestMethod` is equal to *r*/`dsig:DigestMethod`, and
    - Digital Signature Reference Validation of *s* succeeds when the data object to be digested for `dsig:Reference` *r* is obtained (as called for in the Digital Signature Specification Clause 3.2.1.2.1) using, instead, *c*/`resource`.
- Digital Signature Signature Validation of *s* succeeds when the keying information is obtained (as called for in the Digital Signature Specification Clause 3.2.2.1) using, instead, *c*/`principal`.

7.3.4.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section. The diagrams in Annex C informatively describe the XML Syntax for each of the elements and types defined in the preceeding section.

7.3.5 Transactional Conditions

7.3.5.1 Semantics

7.3.5.1.1 Transaction

A `Transaction` *t* is satisfied for an exercise *e* if the specified service *t*/`serviceReference` is utilized to perform one transaction for the purposes of *e*.

7.3.5.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

7.4 Resource Attribute Set Definitions

7.4.1 Semantics

7.4.1.1 Complement

A `complement` element *s* indicates the set that is the complement of the set indicated by the resource attribute set definition element that is the child of *s*.

7.4.1.2 Intersection

An `intersection` element *s* indicates the set that is the intersection of the sets indicated by the resource attribute set definition elements that are the children of *s*.

7.4.1.3 Set

A `set` element *s* indicates the set that is defined by the URI *s*/@`definition`. If that definition requires any parameters, they are provided as the content of *s*.

7.4.1.4 Union

A `union` element *s* indicates the set that is the union of the sets indicated by the resource attribute set definition elements that are the children of *s*.

7.4.2 XML Syntax

The schema in Annex B normatively defines the XML Syntax for each of the elements and types defined in the preceeding section.

Annex A

(normative)

XML Schemas

This annex contains a listing of the three schemas that define the XML syntax of the types and elements defined throughout this part of ISO/IEC 21000. The texts in the schema annotations are informative.

Core

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:sccns="urn:uddi-org:schemaCentricC14N:2002-07-10"
xmlns:r="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" 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:annotation>
         <xsd:documentation>A container of other conditions, all of which must be
met simultaneously.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="allPrincipals" type="r:AllPrincipals"
substitutionGroup="r:principal">
      <xsd:annotation>
         <xsd:documentation>Identifies a principal who must present several
credentials to be authenticated.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="condition" type="r:Condition"
substitutionGroup="r:licensePart">
      <xsd:annotation>
         <xsd:documentation>Specifies the terms, conditions, and obligations under
which rights can be exercised. The conditions in a grant can have a known structure
specifying that they all must be met simultaneously (a conjunction) or only one of
them must be met. This known structure enables a generic engine with no specific
knowledge of the semantics of the rights or conditions to compute the grants in
effect through a chain of delegated rights.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="conditionPattern" type="r:ConditionPattern">
      <xsd:annotation>
```

```
          <xsd:documentation>Optional constraint on the conditions of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
condition.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="conditionPatternAbstract" type="r:ConditionPatternAbstract"
substitutionGroup="r:anXmlPatternAbstract">
       <xsd:annotation>
          <xsd:documentation>A substitution head for condition patterns. Elements
that replace this element must represent a pattern that identifies conditions based
pattern matching.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="datum" type="r:Datum">
       <xsd:annotation>
          <xsd:documentation>Defines one raw parameter to be passed to the
service.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="delegationControl" type="r:DelegationControl"
substitutionGroup="r:licensePart">
       <xsd:annotation>
          <xsd:documentation>Specifies the circumstances under which an associated
grant may be delegated. If delegationControl is absent for a grant, that grant may
not be delegated (unless that permission is conveyed by some other mechanism not yet
defined).</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="digitalResource" type="r:DigitalResource"
substitutionGroup="r:resource">
       <xsd:annotation>
          <xsd:documentation>Provides the means to identify and retrieve the bits
that comprise a particular digital resource.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="everyone" type="r:Everyone"
substitutionGroup="r:principalPatternAbstract">
       <xsd:annotation>
          <xsd:documentation>Identifies a possibly qualified universe of principals.
The propertyAbstract qualification under this element enables the specification of
prerequisiteRight conditions similar to "everyone, possessProperty,
propertyAbstract".</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="exerciseMechanism" type="r:ExerciseMechanism"
substitutionGroup="r:condition">
       <xsd:annotation>
          <xsd:documentation>Indicates the mechanism by which a right must be
exercised.  This is extensible to allow for other such mechanism.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="existsRight" type="r:ExistsRight"
substitutionGroup="r:condition">
       <xsd:annotation>
          <xsd:documentation>Requires that a grant or grantGroup exists that is the
specified grant or grantGroup or matches the specified grantPattern or
grantGroupPattern and that is issued (as one of the license's direct children) by one
of the identified trusted issuers.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="forAll" block="#all" substitutionGroup="r:licensePart"
final="#all">
       <xsd:annotation>
           <xsd:documentation>Applies a universal quantifier to the referenced
licensePart.</xsd:documentation>
       </xsd:annotation>
       <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:annotation>
           <xsd:documentation>Specifies that the fulfiller of an exercise must be the
principal specified herein.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="grant" type="r:Grant" substitutionGroup="r:resource">
       <xsd:annotation>
           <xsd:documentation>The quantum within the license that bestows an
authorization upon some principal. It conveys to a particular principal the sanction
to exercise an identified right against an identified resource, possibly subject to
first fulfilling some conditions.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="grantGroup" type="r:GrantGroup"
substitutionGroup="r:resource">
       <xsd:annotation>
           <xsd:documentation>A container of several grants. A grantGroup does not
define any semantic association, ordering relationship, and so on, between the grants
it contains.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="grantGroupPattern" type="r:GrantGroupPattern"
substitutionGroup="r:resourcePatternAbstract">
       <xsd:annotation>
           <xsd:documentation>A structure representing a predicate expression that can
be evaluated against a grantGroup. All conditions imposed by this element's children
are ANDed together to form the overall grantGroupPattern. Children that are singleton
principals, conditions, grants, and so on, are compared for equality against their
resource by canonicalizing both using a canonicalization algorithm and comparing the
output as binary bit streams. </xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="grantPattern" type="r:GrantPattern"
substitutionGroup="r:resourcePatternAbstract">
       <xsd:annotation>
           <xsd:documentation>Represents a predicate expression that can be evaluated
against a grant. All conditions imposed by this element's children are ANDed together
to form the overall grantPattern. Children that are singleton rights, resources, and
so on are compared for equality against their resource by canonicalizing both using a
canonicalization algorithm and comparing the output as binary bit streams.
</xsd:documentation>
```

```
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="issue" type="r:Issue" substitutionGroup="r:right">
      <xsd:annotation>
         <xsd:documentation>Represents the right to issue licenses corresponding to
the attached resource, which must be a grant or grantGroup. This right can be used to
embody the notion of being a certificate authority.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="issuer" type="r:Issuer">
      <xsd:annotation>
         <xsd:documentation>Identifies the entity who signs the license, attesting
to its validity. If more than one issuer signs the license, it is as if each signed
it independently; one license with several issuers is equivalent to several copies of
the same license, each with one issuer. Indeed, such a syntactic transformation can
feasibly be made while preserving the signature validity.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="keyHolder" type="r:KeyHolder" substitutionGroup="r:principal">
      <xsd:annotation>
         <xsd:documentation>Identifies a principal who possesses a particular key.
Typically, the key is a private key corresponding to a public key identified by this
element, but it may be a symmetric key. The public key can be identified by several
mechanisms defined in the XML Digital Signature specification. </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="license" type="r:License">
      <xsd:annotation>
         <xsd:documentation>A container of one or more grants, each of which conveys
to a principal a right to a resource under certain conditions. The license also
specifies its issuer and other administrative information.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="licenseGroup" type="r:LicenseGroup">
      <xsd:annotation>
         <xsd:documentation>A container of licenses. A licenseGroup does not define
any semantic association, ordering relationship, and so on, between the licenses it
contains.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="licensePart" type="r:LicensePart">
      <xsd:annotation>
         <xsd:documentation>An abstract element from which the various specific
parts of a license are derived. This element defines attributes common to all parts
of a license.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="obtain" type="r:Obtain" substitutionGroup="r:right">
      <xsd:annotation>
         <xsd:documentation>Represents the right to obtain the grant, grantGroup, or
grantPattern specified as a resource associated with this right. Typically, this
right is associated with conditions, such as a fee or validity interval.
</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="patternFromLicensePart" type="r:PatternFromLicensePart"
substitutionGroup="r:anXmlPatternAbstract">
      <xsd:annotation>
```

```
          <xsd:documentation>A structure representing a predicate expression that can
be evaluated against any license part. A comparison is made to the contained
licensePart using the REL equality comparison.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="possessProperty" type="r:PossessProperty"
substitutionGroup="r:right">
      <xsd:annotation>
          <xsd:documentation>Represents the right to claim ownership of particular
characteristics, which are listed as resources associated with this
right.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="prerequisiteRight" type="r:PrerequisiteRight"
substitutionGroup="r:condition">
      <xsd:annotation>
          <xsd:documentation>Requires that another right be possessed before
exercising the associated right.  The specified principal must be able to exercise
the right on the resource under the authorization of the
trustedIssuer.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="principal" type="r:Principal" substitutionGroup="r:resource">
      <xsd:annotation>
          <xsd:documentation>Represents the unique identification of a party involved
in granting or exercising rights. Each principal identifies exactly one
party.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="principalPattern" type="r:PrincipalPattern">
      <xsd:annotation>
          <xsd:documentation>Optional constraint on the principal of the
grant.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="principalPatternAbstract" type="r:PrincipalPatternAbstract"
substitutionGroup="r:resourcePatternAbstract">
      <xsd:annotation>
          <xsd:documentation>A substitution head for principal patterns. Elements
that replace this element must represent a pattern that identifies principals based
pattern matching.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="propertyAbstract" type="r:PropertyAbstract"
substitutionGroup="r:resource">
      <xsd:annotation>
          <xsd:documentation>A substitution head for properties that can be possessed
via PossessProperty.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="resource" type="r:Resource" substitutionGroup="r:licensePart">
      <xsd:annotation>
          <xsd:documentation>The "noun" to which a principal can be granted a right.
A resource can be a digital work (such as an e-book, an audio or video file, or an
image), a service (such as an email service or B2B transaction service), or even a
piece of information that can be owned by a principal (such as a name or an email
address). </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="resourcePattern" type="r:ResourcePattern">
```

```
        <xsd:annotation>
            <xsd:documentation>Optional constraint on the resource of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
resource.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="resourcePatternAbstract" type="r:ResourcePatternAbstract"
substitutionGroup="r:anXmlPatternAbstract">
        <xsd:annotation>
            <xsd:documentation>A substitution head for resource patterns. Elements that
replace this element must represent a pattern that identifies resources based pattern
matching.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="revocationFreshness" type="r:RevocationFreshness"
substitutionGroup="r:condition">
        <xsd:annotation>
            <xsd:documentation>A maximum interval specifying how recently a signature
on the license containing this grant must be checked for revocation. Beyond this
interval, the grant may not be used as part of a proof of
authorization.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="revoke" type="r:Revoke" substitutionGroup="r:right">
        <xsd:annotation>
            <xsd:documentation>Represents the right to revoke a statement that one has
made previously. The act of issuing a license implicitly grants one the right to
revoke it. With this right, one may explicitly delegate that right to
others.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="right" type="r:Right" substitutionGroup="r:licensePart">
        <xsd:annotation>
            <xsd:documentation>The "verb" that a principal can be granted to exercise
against some resource under some condition. Typically, a right specifies an action
(or activity) or a class of actions that a principal may perform on or using the
associated resource. </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="rightPattern" type="r:RightPattern">
        <xsd:annotation>
            <xsd:documentation>Optional constraint on the right of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
right.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="rightPatternAbstract" type="r:RightPatternAbstract"
substitutionGroup="r:anXmlPatternAbstract">
        <xsd:annotation>
            <xsd:documentation>A substitution head for right patterns. Elements that
replace this element must represent a pattern that identifies rights based pattern
matching.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="serviceDescription" type="r:ServiceDescription">
        <xsd:annotation>
            <xsd:documentation>Identifies the metadata and location of a
service.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
```

```
   <xsd:element name="serviceReference" type="r:ServiceReference"
substitutionGroup="r:resource">
      <xsd:annotation>
         <xsd:documentation>Provides the means to locate and interact with a
concrete service. Specifically, this element identifies both an endpoint/address at
which the service is located and meta information by which the type or interface for
the endpoint can be understood.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="trustedIssuer" type="r:TrustedPrincipal">
      <xsd:annotation>
         <xsd:documentation>Defines a trust model based on a principal or set of
principals who are trusted.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityInterval" type="r:ValidityInterval"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Identifies the time interval during which the associated
right is valid.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="anXmlPatternAbstract" type="r:AnXmlPatternAbstract"
substitutionGroup="r:resource">
      <xsd:annotation>
         <xsd:documentation>Elements that replace this element must represent a
pattern that identifies a set of valid XML trees based pattern
matching.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="anXmlExpression" type="r:AnXmlExpression"
substitutionGroup="r:anXmlPatternAbstract">
      <xsd:annotation>
         <xsd:documentation>A Boolean expression in some identified XML expression
language. The default language is XPath1.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <!--Complex Types-->
   <xsd:complexType name="AllConditions">
      <xsd:annotation>
         <xsd:documentation>A container of other conditions, all of which must be
met simultaneously.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>Identifies a principal who must present several
credentials to be authenticated.</xsd:documentation>
      </xsd:annotation>
      <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="Condition">
      <xsd:annotation>
         <xsd:documentation>Specifies the terms, conditions, and obligations under
which rights can be exercised. The conditions in a grant can have a known structure
specifying that they all must be met simultaneously (a conjunction) or only one of
them must be met. This known structure enables a generic engine with no specific
knowledge of the semantics of the rights or conditions to compute the grants in
effect through a chain of delegated rights.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ConditionPattern">
      <xsd:annotation>
         <xsd:documentation>Optional constraint on the conditions of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
condition.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A substitution head for condition patterns. Elements
that replace this element must represent a pattern that identifies conditions based
pattern matching.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Datum">
      <xsd:annotation>
         <xsd:documentation>Defines one raw parameter to be passed to the
service.</xsd:documentation>
      </xsd:annotation>
      <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="DelegationControl">
      <xsd:annotation>
         <xsd:documentation>Specifies the circumstances under which an associated
grant may be delegated. If delegationControl is absent for a grant, that grant may
not be delegated (unless that permission is conveyed by some other mechanism not yet
defined).</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart">
```

```
             <xsd:sequence>
                <xsd:choice>
                   <xsd:element name="maxDepth" type="xsd:nonNegativeInteger">
                      <xsd:annotation>
                         <xsd:documentation>Specifies the maximum depth of
delegation chaining. A value of zero indicates that this grant may not be delegated.
When a grant with this constraint is delegated, the contained count must be
decremented by one.</xsd:documentation>
                      </xsd:annotation>
                   </xsd:element>
                   <xsd:element name="infinite">
                      <xsd:annotation>
                         <xsd:documentation>Specifies that an infinite chain of
delegation is permitted.</xsd:documentation>
                      </xsd:annotation>
                   </xsd:element>
                </xsd:choice>
                <xsd:element name="additionalConditionsProhibited" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Indicates whether the delegated copy can
specify conditions not contained in the original copy. If omitted, additional
conditions can be specified; if present, the delegated copy must contain the same
conditions as the original copy.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element ref="r:forAll" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="to" minOccurs="0" maxOccurs="unbounded">
                   <xsd:annotation>
                      <xsd:documentation>Specifies a principal to whom the grant
may be delegated. If more than one "to" element is specified, the principal may be
any of those identified: all the "to" elements are ORed together.</xsd:documentation>
                   </xsd:annotation>
                   <xsd:complexType>
                      <xsd:sequence>
                         <xsd:element ref="r:principal"/>
                      </xsd:sequence>
                   </xsd:complexType>
                </xsd:element>
             </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="DigitalResource">
       <xsd:annotation>
          <xsd:documentation>Provides the means to identify and retrieve the bits
that comprise a particular digital resource.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="r:Resource">
             <xsd:choice minOccurs="0">
                <xsd:element name="nonSecureIndirect" type="r:NonSecureReference">
                   <xsd:annotation>
                      <xsd:documentation>A non-cryptographically-secure reference
to the bits that comprise a digital resource. </xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="secureIndirect" type="dsig:ReferenceType">
                   <xsd:annotation>
```

```
                        <xsd:documentation>An indirect, non-URI reference to the
digital resource. The coupling to the referenced resource is made secure and
unambiguous using cryptographic techniques.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="binary" type="xsd:base64Binary">
                    <xsd:annotation>
                        <xsd:documentation>The bits that comprise the digital
resource.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="anXml">
                    <xsd:annotation>
                        <xsd:documentation>An embedded digital resource, cast as an
XML document fragment, within the current document. There is no standard way to embed
an arbitrary full XML document within another due to issues such as local entities,
character set differences, and document-global ID scope.</xsd:documentation>
                    </xsd:annotation>
                    <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:annotation>
                        <xsd:documentation>A locator scheme invented by others.
</xsd:documentation>
                    </xsd:annotation>
                </xsd:any>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EncryptedContent">
    <xsd:annotation>
        <xsd:documentation>Represents an encryption of the XML element's
contents.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="enc:EncryptedDataType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Everyone">
    <xsd:annotation>
        <xsd:documentation>Identifies a possibly qualified universe of principals.
The propertyAbstract qualification under this element enables the specification of
prerequisiteRight conditions similar to "everyone, possessProperty,
propertyAbstract".</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="r:PrincipalPatternAbstract">
            <xsd:sequence minOccurs="0">
                <xsd:element ref="r:propertyAbstract"/>
                <xsd:element ref="r:trustedIssuer" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

```
    <xsd:complexType name="ExerciseMechanism">
        <xsd:annotation>
            <xsd:documentation>Indicates the mechanism by which a right must be
exercised.  This is extensible to allow for other such mechanism.</xsd:documentation>
        </xsd:annotation>
        <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:annotation>
            <xsd:documentation>Requires that a grant or grantGroup exist and that is
issued (as one of the license's direct children) by one of the identified trusted
issuers.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="r:Condition">
                <xsd:sequence minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>The grant element identifies shape grant that
expresses the right which must be held in order to satisfy the existsRight. The
trustedIssuer identifies one or more principals trusted to issue the
right.</xsd:documentation>
                    </xsd:annotation>
                    <xsd:choice>
                        <xsd:element ref="r:grant"/>
                        <xsd:element ref="r:grantPattern"/>
                        <xsd:element ref="r:grantGroup"/>
                        <xsd:element ref="r:grantGroupPattern"/>
                    </xsd:choice>
                    <xsd:element ref="r:trustedIssuer" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Fulfiller">
        <xsd:annotation>
            <xsd:documentation>Specifies that the fulfiller of an exercise must be the
principal specified herein.</xsd:documentation>
        </xsd:annotation>
        <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:annotation>
          <xsd:documentation>The quantum within the license that bestows an
authorization upon some principal. It conveys to a particular principal the sanction
to exercise an identified right against an identified resource, possibly subject to
first fulfilling some conditions.</xsd:documentation>
       </xsd:annotation>
       <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:annotation>
                      <xsd:documentation>The encrypted contents of a grant. When
decrypted, the clear text logically becomes the entire contents of the grant,
replacing this encryptedGrant element. As specified in XML ENCRYPT, the encyptedGrant
element must contain the "type" attribute with a value of
"http://www.w3.org/2001/04/xmlenc#Content Type".</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
             </xsd:choice>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="GrantGroup">
       <xsd:annotation>
          <xsd:documentation>A container of several grants. A grantGroup does not
define any semantic association, ordering relationship, and so on, between the grants
it contains.</xsd:documentation>
       </xsd:annotation>
       <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:annotation>
                      <xsd:documentation>The encrypted contents of a grantGroup.
When decrypted, the clear text logically becomes the entire contents of the
grantGroup, replacing this encryptedGrantGroup element. As specified in XML ENCRYPT,
the encyptedGrantGroup element must contain the "type" attribute with a value of
"http://www.w3.org/2001/04/xmlenc#Content Type".</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
             </xsd:choice>
          </xsd:extension>
```

```
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="GrantGroupPattern">
      <xsd:annotation>
         <xsd:documentation>A structure representing a predicate expression that can
be evaluated against a grantGroup. All conditions imposed by this element's children
are ANDed together to form the overall grantGroupPattern. Children that are singleton
principals, conditions, grants, and so on, are compared for equality against their
resource by canonicalizing both using a canonicalization algorithm and comparing the
output as binary bit streams. </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:ResourcePatternAbstract">
            <xsd:sequence>
               <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:annotation>
                     <xsd:documentation>Optional constraint imposed on the
grantGroup as a whole, evaluated against the subtree rooted at the grantGroup. This
element specifies a  pattern (such as an Xpath) to evaluate against the resource
grantGroup as a whole.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="GrantPattern">
      <xsd:annotation>
         <xsd:documentation>Represents a predicate expression that can be evaluated
against a grant. All conditions imposed by this element's children are ANDed together
to form the overall grantPattern. Children that are singleton rights, resources, and
so on are compared for equality against their resource by canonicalizing both using a
canonicalization algorithm and comparing the output as binary bit streams.
</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:ResourcePatternAbstract">
            <xsd:sequence>
               <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:annotation>
                        <xsd:documentation>Optional constraint imposed on the grant
as a whole, evaluated against the subtree rooted at the grant. This element specifies
a  pattern (such as an Xpath) to evaluate against the resource grant as a
whole.</xsd:documentation>
                     </xsd:annotation>
                  </xsd:element>
               </xsd:sequence>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Inventory">
         <xsd:annotation>
            <xsd:documentation>A container used to define elements frequently used
throughout a license. These elements are defined in the inventory, and then
referenced by ID wherever they are needed within the license.</xsd:documentation>
         </xsd:annotation>
         <xsd:sequence>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element ref="r:licensePart"/>
            </xsd:choice>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Issue">
         <xsd:annotation>
            <xsd:documentation>Represents the right to issue licenses corresponding to
the attached resource, which must be a grant or grantGroup. This right can be used to
embody the notion of being a certificate authority.</xsd:documentation>
         </xsd:annotation>
         <xsd:complexContent>
            <xsd:extension base="r:Right"/>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="Issuer">
         <xsd:annotation>
            <xsd:documentation>Describes information associated with each issuer
(signer) of a license. The SignedInfo in the Signature must contain a Reference that
covers the whole license except for its immediate issuer children. Optionally, the
SignedInfo may contain a second Reference that covers the details of the specific
issuer. Boilerplate XPATH Transforms can be used for each
Reference.</xsd:documentation>
         </xsd:annotation>
         <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:annotation>
               <xsd:documentation>Issuer-specific contributions to the
license.</xsd:documentation>
            </xsd:annotation>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="IssuerDetails">
      <xsd:annotation>
         <xsd:documentation>Issuer-specific contributions to the
license.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element name="timeOfIssue" type="xsd:dateTime" minOccurs="0">
            <xsd:annotation>
               <xsd:documentation>The date at which the license was issued, as
attested to by this issuer. For many purposes, validators cannot rely on this
assertion, but instead require some disinterested third part to attest to the date of
issuance.</xsd:documentation>
            </xsd:annotation>
         </xsd:element>
         <xsd:element name="revocationMechanism" type="r:RevocationMechanism"
minOccurs="0" maxOccurs="unbounded">
            <xsd:annotation>
               <xsd:documentation>A mechanism by which the issuer may post notice
of license revocation. Software checking for revocation may use any one of the
identified mechanisms to check for revocation.</xsd:documentation>
            </xsd:annotation>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="KeyHolder">
      <xsd:annotation>
         <xsd:documentation>Identifies a principal who possesses a particular key.
Typically, the key is a private key corresponding to a public key identified by this
element, but it may be a symmetric key. The public key can be identified by several
mechanisms defined in the XML Digital Signature specification. </xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A container of one or more grants, each of which conveys
to a principal a right to a resource under certain conditions. The license also
specifies its issuer and other administrative information. The optional licenseID
attribute uniquely and globally identify this license over space and time. Note (by
way of comparison to validity intervals in, say, X509) that as a pragmatic matter,
each right in a license usually contains a time condition to limit its validity
time.</xsd:documentation>
      </xsd:annotation>
      <xsd:choice>
```

```
          <xsd:sequence>
             <xsd:element name="title" type="r:LinguisticString" minOccurs="0"
maxOccurs="unbounded">
                <xsd:annotation>
                   <xsd:documentation>A handy set of phrases that describe this
license. The intent is that these can be shown to human beings in user interfaces in
which licenses need to be managed, such as pick-lists.</xsd:documentation>
                </xsd:annotation>
             </xsd:element>
             <xsd:element name="inventory" type="r:Inventory" minOccurs="0">
                <xsd:annotation>
                   <xsd:documentation>A container used to define elements frequently
used throughout a license. These elements are defined in the inventory, and then
referenced by ID wherever they are needed within the license.</xsd:documentation>
                </xsd:annotation>
             </xsd:element>
             <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:annotation>
                   <xsd:documentation>Specifies any other information to be conveyed
in a license, such as information peripherally related to authentication and
authorization, but not part of the core infrastructure. These extended elements
typically fall under the license signature(s). However,  recipients at their
discretion can and will choose to ignore these extensions.</xsd:documentation>
                </xsd:annotation>
                <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:annotation>
                <xsd:documentation>The encrypted contents of a License. When
decrypted, the clear text logically becomes the entire contents of the License,
replacing this encryptedData element. The encyptedLicense element must, per XML
ENCRYPT, contain the 'type' attribute of 'http://www.w3.org/2001/04/xmlenc#Content
Type'.</xsd:documentation>
             </xsd:annotation>
          </xsd:element>
       </xsd:choice>
       <xsd:attribute name="licenseId" type="xsd:anyURI" use="optional"/>
       <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
    <xsd:complexType name="LicenseGroup">
       <xsd:annotation>
          <xsd:documentation>A container of licenses. A licenseGroup does not define
any semantic association, ordering relationship, and so on, between the licenses it
contains.</xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
          <xsd:element ref="r:license" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
```

```
   <xsd:complexType name="LicensePart">
      <xsd:annotation>
         <xsd:documentation>An abstract element from which the various specific
parts of a license are derived. This element defines attributes common to all parts
of a license. A license part can have an identifier or reference an identifier
defined elsewhere in this license. This mechanism reduces verbosity by defining
commonly-used elements in one place and referencing them elsewhere.  However, this is
a purely syntactic shorthand; no semantic connection between the definition site and
use site is implied. </xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A string and an optional xml:lang indication of the
language in which it resides, which enables embedded XML structured
content.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A reference similar to dsig:ReferenceType, but lacking
the cryptographic connection.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>Represents the right to obtain the grant, grantGroup, or
grantPattern specified as a resource associated with this right. Typically, this
right is associated with conditions, such as a fee or validity interval.
</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PatternFromLicensePart">
      <xsd:annotation>
         <xsd:documentation>A structure representing a predicate expression that can
be evaluated against any license part. A comparison is made to the contained
licensePart using the REL equality comparison.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract">
```

```
            <xsd:sequence>
               <xsd:element ref="r:licensePart"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PossessProperty">
      <xsd:annotation>
         <xsd:documentation>Represents the right to claim ownership of particular 
characteristics, which are listed as resources associated with this 
right.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PrerequisiteRight">
      <xsd:annotation>
         <xsd:documentation>Requires that another right be possessed before 
exercising the associated right.  The specified principal must be able to exercise 
the right on the resource under the authorization of the 
trustedIssuer.</xsd:documentation>
      </xsd:annotation>
      <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:trustedIssuer" minOccurs="0"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Principal">
      <xsd:annotation>
         <xsd:documentation>Represents the unique identification of a party involved 
in granting or exercising rights. Each principal identifies exactly one 
party.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Resource"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PrincipalPattern">
      <xsd:annotation>
         <xsd:documentation>Optional constraint on the principal of the 
grant.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A substitution head for principal patterns. Elements 
that replace this element must represent a pattern that identifies principals based 
pattern matching.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:ResourcePatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PropertyAbstract">
      <xsd:annotation>
         <xsd:documentation>An abstract type for properties that can be possessed
via PossessProperty.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Resource"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Resource">
      <xsd:annotation>
         <xsd:documentation>The "noun" to which a principal can be granted a right.
A resource can be a digital work (such as an e-book, an audio or video file, or an
image), a service (such as an email service or B2B transaction service), or even a
piece of information that can be owned by a principal (such as a name or an email
address). </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ResourcePattern">
      <xsd:annotation>
         <xsd:documentation>Optional constraint on the resource of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
resource.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A substitution head for resource patterns. Elements that
replace this element must represent a pattern that identifies resources based pattern
matching.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RevocationFreshness">
      <xsd:annotation>
         <xsd:documentation>A maximum interval specifying how recently a signature
on the license containing this grant must be checked for revocation. Beyond this
interval, the grant may not be used as part of a proof of
authorization.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:sequence minOccurs="0">
               <xsd:choice>
                  <xsd:element name="maxIntervalSinceLastCheck"
type="xsd:duration">
                     <xsd:annotation>
```

```
                                <xsd:documentation>Indicates the maximum amount of time
that may elapse since the last time the grant was checked for revocation. A value of
zero indicates that the grant must be explicitly checked each time it is
exercised.</xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                        <xsd:element name="noCheckNecessary">
                            <xsd:annotation>
                                <xsd:documentation>Indicates that for this use of this
condition, a check for revocation is not needed.</xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:choice>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="RevocationMechanism">
        <xsd:annotation>
            <xsd:documentation>Indicates a mechanism through which notice of revocation
of licenses may be communicated. To allow others to define their own revocation
communication mechanism, this element is extensible.</xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:element name="querySignature">
                <xsd:annotation>
                    <xsd:documentation>Indicates a service instance  to query for the
status of a signature.</xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="r:serviceReference"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="revocationListDistributionPoint">
                <xsd:annotation>
                    <xsd:documentation>Indicates a service through which to obtain a
revocation list.</xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="r:serviceReference"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:any namespace="##other" processContents="lax">
                <xsd:annotation>
                    <xsd:documentation>Represents revocation mechanisms invented by
others.</xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="Revoke">
        <xsd:annotation>
            <xsd:documentation>Represents the right to revoke a statement that one has
made previously. The act of issuing a license implicitly grants one the right to
revoke it. With this right, one may explicitly delegate that right to
others.</xsd:documentation>
```

```
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Right">
      <xsd:annotation>
         <xsd:documentation>The "verb" that a principal can be granted to exercise
against some resource under some condition. Typically, a right specifies an action
(or activity) or a class of actions that a principal may perform on or using the
associated resource. </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RightPattern">
      <xsd:annotation>
         <xsd:documentation>Optional constraint on the right of the grant. This
pattern is evaluated against the subtree of the grant rooted at the
right.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A substitution head for right patterns. Elements that
replace this element must represent a pattern that identifies rights based pattern
matching.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ServiceDescription">
      <xsd:annotation>
         <xsd:documentation>Identifies the metadata and location of a
service.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ServiceReference">
      <xsd:annotation>
         <xsd:documentation>Provides the means to locate and interact with a
concrete service. Specifically, this element identifies both an endpoint/address at
which the service is located and meta information by which the type or interface for
the endpoint can be understood.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:sequence minOccurs="0">
               <xsd:element ref="r:serviceDescription"/>
               <xsd:element name="serviceParameters" minOccurs="0">
                  <xsd:annotation>
```

```
                        <xsd:documentation>Provides contextual parameters that may be
needed to interact with the service. The exact interpretation of each parameter is
specific to the semantics of the service and is not specified
here.</xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                            <xsd:element ref="r:datum"/>
                            <xsd:element name="transforms" type="dsig:TransformsType"
minOccurs="0">
                                <xsd:annotation>
                                    <xsd:documentation>Lists optional transformations
to be applied in sequence over datum.</xsd:documentation>
                                </xsd:annotation>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TrustedPrincipal">
    <xsd:annotation>
        <xsd:documentation>Indicates the means for identifying a principal trusted
to perform a certain action. Extensions to this type might specify additional
policies not articulated here.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="r:LicensePart">
            <xsd:choice minOccurs="0">
                <xsd:element ref="r:principal"/>
                <xsd:element name="any">
                    <xsd:annotation>
                        <xsd:documentation>Specifies a list of principals, any of
which can be used.</xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="r:principal" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ValidityInterval">
    <xsd:annotation>
        <xsd:documentation>Identifies the time interval during which the associated
right is valid.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="r:Condition">
            <xsd:sequence>
                <xsd:element name="notBefore" type="xsd:dateTime" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Identifies the beginning of the
interval.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="notAfter" type="xsd:dateTime" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Identifies the end of the
interval.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AnXmlPatternAbstract">
    <xsd:annotation>
        <xsd:documentation>The substitution head for all patterns in XrML. Elements
that replace this element must represent a pattern that identifies a set of valid XML
trees based pattern matching.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="r:Resource"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AnXmlExpression" mixed="true"
sccns:embeddedLangAttribute="r:lang">
    <xsd:annotation>
        <xsd:documentation>A Boolean expression in some identified XML expression
language. The default language is XPath1.</xsd:documentation>
    </xsd:annotation>
    <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>
<!-- Simple Types-->
<xsd:simpleType name="LicensePartId">
    <xsd:annotation>
        <xsd:documentation>Identifier for a license part. Using this identifier,
commonly-used elements can be defined in one place and referenced elsewhere, thus
reducing verbosity.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:NCName"/>
</xsd:simpleType>
<xsd:simpleType name="VariableName">
    <xsd:annotation>
        <xsd:documentation>The name of a variable.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:NCName"/>
</xsd:simpleType>
<xsd:simpleType name="Uuid">
    <xsd:annotation>
        <xsd:documentation>A DCE Uuid. For example, 1FAC02A2-9C46-4ceb-ABD2-
9D569A379218</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:length value="36"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Standard Extension

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:sx="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:r="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xsd:import namespace="http://www.xrml.org/schema/2002/05/xrml2core"
schemaLocation="xrml2core.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="commonName" type="sx:CommonName" substitutionGroup="sx:name">
      <xsd:annotation>
         <xsd:documentation>A name by which an entity is colloquially known.
Intended to be used as the CN name part from X400.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="bestPriceUnder" type="sx:BestPriceUnder"
substitutionGroup="sx:paymentAbstract">
      <xsd:annotation>
         <xsd:documentation>Specifies the maximum fee that ultimately must be paid
without specifying the ultimate fee exactly. The ultimate fee is determined through a
later, unspecified settlement mechanism.  While Max overrides Min if Max is less than
Min, Min overrides BestPriceUnder if BestPriceUnder is less than
Min.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="callForPrice" type="sx:CallForPrice"
substitutionGroup="sx:paymentAbstract">
      <xsd:annotation>
         <xsd:documentation>Identifies an entity with whom a price must be
negotiated before exercising the right. Any one of the services can be
contacted.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="dnsName" type="sx:DnsName" substitutionGroup="sx:name">
      <xsd:annotation>
         <xsd:documentation>A name in the DNS name space, with trailing period
omitted. For example, "xyz.com"</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="emailName" type="sx:EmailName" substitutionGroup="sx:name">
      <xsd:annotation>
         <xsd:documentation>An Internet email address (per rfc822/rfc2822)
associated with the entity.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="exerciseLimit" type="sx:ExerciseLimit"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Indicates a maximum number of times that the right may
be exercised.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="fee" type="sx:Fee" substitutionGroup="r:condition">
      <xsd:annotation>
```

```
          <xsd:documentation>Indicates that a fee must be paid before a right is
exercised.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="markup" type="sx:Markup"
substitutionGroup="sx:paymentAbstract">
       <xsd:annotation>
          <xsd:documentation>Specifies a fee due each time some other fees are
due.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="name" type="sx:Name" substitutionGroup="r:propertyAbstract">
       <xsd:annotation>
          <xsd:documentation>A property indicating a name from some name
space.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="paymentAbstract" type="sx:PaymentAbstract">
       <xsd:annotation>
          <xsd:documentation>The head of a substitution group chain for the
PaymentAbstract type.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="paymentFlat" type="sx:PaymentFlat"
substitutionGroup="sx:paymentAbstract">
       <xsd:annotation>
          <xsd:documentation>Specifies a payment due upon exercising a right when the
value in paymentRecord is False. The contained service reference should be used to
determine if the fee has already been paid and to keep record of
payment.</xsd:documentation>

       </xsd:annotation>
   </xsd:element>
   <xsd:element name="paymentMetered" type="sx:PaymentMetered"
substitutionGroup="sx:paymentAbstract">
       <xsd:annotation>
          <xsd:documentation>Specifies a payment due for each time interval during
which the right is actually exercised.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="paymentPerInterval" type="sx:PaymentPerInterval"
substitutionGroup="sx:paymentAbstract">
       <xsd:annotation>
          <xsd:documentation>Specifies a payment due for each time interval during
which the ability to exercise the right is desired. The enclosed serviceReference
indicates the time through which payment has been made. While the value stored in the
contained serviceReference is greater than the global official time, no additional
fee is due. Each time a payment is made, the value stored in the contained
serviceReference time should be updated. If the value represents a future time, it is
increased by the "per" period each time it is updated. If the value represents a past
time, it is set to the sum of the global official time and the "per"
period.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="paymentPerUse" type="sx:PaymentPerUse"
substitutionGroup="sx:paymentAbstract">
       <xsd:annotation>
          <xsd:documentation>Specifies a payment due each time a right is
exercised.</xsd:documentation>
       </xsd:annotation>
```

```
    </xsd:element>
    <xsd:element name="propertyUri" type="sx:PropertyUri"
substitutionGroup="r:propertyAbstract">
        <xsd:annotation>
            <xsd:documentation>Indicates a property using a URI.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="rate" type="sx:Rate" substitutionGroup="r:licensePart">
        <xsd:annotation>
            <xsd:documentation>A fixed amount of money in a designated
currency.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="revocable" type="sx:Revocable" substitutionGroup="r:resource">
        <xsd:annotation>
            <xsd:documentation>Identifies a SignatureValue that can be revoked. The
SignatureValue can be identified literally or by reference. In the latter case, the
result of dereferencing the reference must be of type dsig:SignatureType; the
SignatureValue being revoked is the one SignatureValue therein.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="rightUri" type="sx:RightUri" substitutionGroup="r:right">
        <xsd:annotation>
            <xsd:documentation>Indicates a right using a URI.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="seekApproval" type="sx:SeekApproval"
substitutionGroup="r:condition">
        <xsd:annotation>
            <xsd:documentation>Indicates that the specified service must be contacted
and its approval gained before exercising the associated right.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="stateReferenceValuePattern"
type="sx:StateReferenceValuePattern" substitutionGroup="r:anXmlPatternAbstract">
        <xsd:annotation>
            <xsd:documentation>A pattern that identifies a set of service references
using pattern matching by dereferencing their values.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="territory" type="sx:Territory"
substitutionGroup="r:condition">
        <xsd:annotation>
            <xsd:documentation>Indicates a geographic or virtual space within which the
associated right may be exercised.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="trackQuery" type="sx:TrackQuery"
substitutionGroup="r:condition">
        <xsd:annotation>
            <xsd:documentation>Represents a condition on the tracking state updated by
TrackReport. For example, this condition can be used to predicate the granting of one
right on the successful exercise of another.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="trackReport" type="sx:TrackReport"
substitutionGroup="r:condition">
        <xsd:annotation>
            <xsd:documentation>Indicates that exercising a right must be reported to a
designated tracking service.</xsd:documentation>
```

```
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="transferControl" type="sx:TransferControl"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Indicates that the specified service must be contacted
to confirm ownership (by the principal) of some virtual token upon which the
exercisability of this grant depends.  Typically, this service may also be used to
cause a transfer of ownership of the virtual token to another
principal.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="uddi" type="sx:Uddi" substitutionGroup="r:serviceDescription">
      <xsd:annotation>
         <xsd:documentation>Specifies that the UDDI Business Registry (or possibly a
private UDDI registry) be used for protocol and endpoint information. Contains the
information for a UddiServiceIdentifier as defined in the UDDI
specification.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityIntervalDurationPattern"
type="sx:ValidityIntervalDurationPattern"
substitutionGroup="r:conditionPatternAbstract">
      <xsd:annotation>
         <xsd:documentation>A pattern matching any ValidityInterval of the specified
duration.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityIntervalFloating" type="sx:ValidityIntervalFloating"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Represents an interval of time that begins with the
first exercise of a right.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityIntervalStartsNow" type="sx:ValidityIntervalStartsNow"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>The ValidityIntervalStartsNow is satisfied at time t if
the specified ValidityInterval starts within the specified tolerances of time
t.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityTimeMetered" type="sx:ValidityTimeMetered"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Represents an accumulative period of time. A user can
start and stop exercising a right, and the metering clock runs only when the right is
being exercised. The right can be exercised as long as the total remaining time has
not been used.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="validityTimePeriodic" type="sx:ValidityTimePeriodic"
substitutionGroup="r:condition">
      <xsd:annotation>
         <xsd:documentation>Indicates a validity time window that recurs
periodically. For example, this condition can be used to express time windows such as
"every weekend" or "the second week of every month".</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="wsdlAddress" type="sx:WsdlAddress" 
substitutionGroup="r:serviceDescription">
       <xsd:annotation>
          <xsd:documentation>Uses a specified portion of the identified WSDL file to 
indicate the metadata and protocol information while allowing the endpoint addressing 
information to be specified separately.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="wsdlComplete" type="sx:WsdlComplete" 
substitutionGroup="r:serviceDescription">
       <xsd:annotation>
          <xsd:documentation>Uses a specified portion of the identified WSDL file to 
indicate the full protocol and endpoint information.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="x509SubjectName" type="sx:X509SubjectName" 
substitutionGroup="sx:name">
       <xsd:annotation>
          <xsd:documentation>The subject name of some X509 certificate associated 
with the entity. Intended to address legacy interoperability issues involving X509 
certificates.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <xsd:element name="x509SubjectNamePattern" type="sx:X509SubjectNamePattern" 
substitutionGroup="r:resourcePatternAbstract">
       <xsd:annotation>
          <xsd:documentation>A pattern that identifies a set of X509 subject names 
using pattern matching.</xsd:documentation>
       </xsd:annotation>
   </xsd:element>
   <!-- Complex Types -->
   <xsd:complexType name="AccountPayable">
       <xsd:annotation>
          <xsd:documentation>Identifies an entity to whom one can transfer a sum of 
money and the means by which that transfer is to take place. Since many transfer 
mechanisms have been identified, this element is extensible.</xsd:documentation>
       </xsd:annotation>
       <xsd:choice>
          <xsd:element name="paymentService">
             <xsd:annotation>
                <xsd:documentation>Identifies an accounts payable service. The 
interface to the service indicates the transfer mechanism.</xsd:documentation>
             </xsd:annotation>
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:element ref="r:serviceReference"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="aba">
             <xsd:annotation>
                <xsd:documentation>Identifies an account within a US banking 
institution using conventions established by the American Banking 
Association.</xsd:documentation>
             </xsd:annotation>
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:element name="institution">
                      <xsd:annotation>
```

```
                        <xsd:documentation>Identifies a US banking institution by
its ABA routing number.</xsd:documentation>
                    </xsd:annotation>
                    <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:annotation>
                        <xsd:documentation>Identifies an account at a US banking
institution.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:any namespace="##other" processContents="lax">
        <xsd:annotation>
            <xsd:documentation>Other banking mechanisms may be specified
here.</xsd:documentation>
        </xsd:annotation>
    </xsd:any>
</xsd:choice>
</xsd:complexType>
<xsd:complexType name="BestPriceUnder">
    <xsd:annotation>
        <xsd:documentation>Specifies the maximum fee that ultimately must be paid
without specifying the ultimate fee exactly. The ultimate fee is determined through a
later, unspecified settlement mechanism.  While Max overrides Min if Max is less than
Min, Min overrides BestPriceUnder if BestPriceUnder is less than
Min.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
                <xsd:element ref="sx:paymentAbstract"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="CallForPrice">
    <xsd:annotation>
        <xsd:documentation>Identifies an entity with whom a price must be
negotiated before exercising the right.  Any one of the services can be
contacted.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="sx:PaymentAbstract">
            <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:annotation>
        <xsd:documentation>A name by which an entity is colloquially known.
Intended to be used as the CN name part from X400.</xsd:documentation>
```

```
        </xsd:annotation>
        <xsd:complexContent mixed="true">
            <xsd:extension base="sx:Name"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="DnsName" mixed="true">
        <xsd:annotation>
            <xsd:documentation>A name in the DNS name space, with trailing period
omitted. For example, "xyz.com"</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent mixed="true">
            <xsd:extension base="sx:Name"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="EmailName" mixed="true">
        <xsd:annotation>
            <xsd:documentation>An Internet email address (per rfc822/rfc2822)
associated with the entity.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent mixed="true">
            <xsd:extension base="sx:Name"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ExerciseLimit">
        <xsd:annotation>
            <xsd:documentation>Indicates a maximum number of times that the right may
be exercised.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Fee">
        <xsd:annotation>
            <xsd:documentation>Indicates that a fee must be paid before a right is
exercised.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="r:Condition">
                <xsd:sequence minOccurs="0">
                    <xsd:element ref="sx:paymentAbstract"/>
                    <xsd:element name="min" minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation>Specifies the minimum amount due the fee
recipient. If the total amount paid is less than the value of this element, a new
payment in the amount of the difference is due.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element ref="sx:paymentAbstract"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="max" minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation>Specifies the maximum amount due the fee
recipient. If the total amount paid is greater than the value of this element, a new
credit in the amount of the difference is due.  If the total amount paid is equal to
the value of this element, all other payments resulting from this fee are void until
the max increases.</xsd:documentation>
```

```
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="sx:paymentAbstract"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="to" type="sx:AccountPayable" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Indicates the entity to whom the fee must
be paid and the payment mechanism. In the rare case that this element is absent,
payment information must be identified by context.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Markup">
    <xsd:annotation>
        <xsd:documentation>Specifies a fee due each time some other fees are
due.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
                <xsd:element name="fraction" type="xsd:float">
                    <xsd:annotation>
                        <xsd:documentation>Indicates the fractional rate to calculate
the markup. For example, a value of 0.05 indicates a fee of 5% above  the underlying
fees specified within this markup element.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:choice maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation>Fees underlying the markup, which also
must be paid to the respective parties.</xsd:documentation>
                    </xsd:annotation>
                    <xsd:element ref="sx:fee"/>
                    <xsd:element name="feeForResource">
                        <xsd:annotation>
                            <xsd:documentation>Identifies another resource that must
be used (and paid for) in conjunction with the resource specified in the grant
containing this markup. </xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element ref="r:resource"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:choice>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Name">
    <xsd:annotation>
        <xsd:documentation>A logically abstract type for which concrete name kinds
may be defined as subtypes. </xsd:documentation>
```

```
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:PropertyAbstract"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PaymentAbstract">
      <xsd:annotation>
         <xsd:documentation>An abstract type indicating a quantity of money, perhaps
to be paid at a certain rate, etc.</xsd:documentation>
      </xsd:annotation>
   </xsd:complexType>
   <xsd:complexType name="PaymentFlat">
      <xsd:annotation>
         <xsd:documentation>Specifies a payment due upon exercising a right when the
value in paymentRecord is False. The contained service reference should be used to
determine if the fee has already been paid and to keep record of
payment.</xsd:documentation>

      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
               <xsd:element ref="sx:rate"/>
               <xsd:element ref="r:serviceReference"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PaymentMetered">
      <xsd:annotation>
         <xsd:documentation>Specifies a payment due for each time interval during
which the right is actually exercised.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
               <xsd:element ref="sx:rate"/>
               <xsd:element name="per" type="xsd:duration">
                  <xsd:annotation>
                     <xsd:documentation>The time period at which the rate is
applied.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
               <xsd:element name="by" type="xsd:duration">
                  <xsd:annotation>
                     <xsd:documentation>The billing time
period.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
               <xsd:element name="phase" type="xsd:duration">
                  <xsd:annotation>
                     <xsd:documentation>The amount of time that may elapse before
the first billing occurs.  A value of 0 has the effect of rounding up.  A value equal
to the "by" period has the effect of rounding down.  A value equal to half of the
"by" period has the effect of rounding to the nearest "by"
quantum.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PaymentPerInterval">
      <xsd:annotation>
         <xsd:documentation>Specifies a payment due for each time interval during
which the ability to exercise the right is desired. The enclosed serviceReference
indicates the time through which payment has been made. While the value stored in the
contained serviceReference is greater than the global official time, no additional
fee is due. Each time a payment is made, the value stored in the contained
serviceReference time should be updated. If the value represents a future time, it is
increased by the "per" period each time it is updated. If the value represents a past
time, it is set to the sum of the global official time and the "per"
period.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
               <xsd:element ref="sx:rate"/>
               <xsd:element name="per" type="xsd:duration">
                  <xsd:annotation>
                     <xsd:documentation>The time period allocated for each payment
of the rate.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
               <xsd:element ref="r:serviceReference"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PaymentPerUse">
      <xsd:annotation>
         <xsd:documentation>Specifies a payment due each time a right is
exercised.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="sx:PaymentAbstract">
            <xsd:sequence>
               <xsd:element ref="sx:rate"/>
               <xsd:element name="allowPrePay" minOccurs="0">
                  <xsd:annotation>
                     <xsd:documentation>Indicates whether the rate can be prepaid.
If omitted, the rate must be paid each time the right is exercised. The value stored
in the contained serviceReference represents the remaining number of prepaid uses of
the right.  Each time a payment is made, this value should be incremented by the
initialNumberOfUses.  When the right is exercised, this value should be decremented
by one.</xsd:documentation>
                  </xsd:annotation>
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="initialNumberOfUses" type="xsd:integer"
minOccurs="0">
                           <xsd:annotation>
                              <xsd:documentation>Specifies the number of uses
allowed for each payment of the rate.  Defaults to 1.</xsd:documentation>
                           </xsd:annotation>
                        </xsd:element>
                        <xsd:element ref="r:serviceReference"/>
                     </xsd:sequence>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PropertyUri">
      <xsd:annotation>
         <xsd:documentation>Indicates a property using a URI.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>A fixed amount of money in a designated currency (using
ISO 4217). If the currency is not specified, then the default is USD with its most
recent interpretation.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:LicensePart">
            <xsd:sequence minOccurs="0">
               <xsd:element name="amount" type="xsd:float">
                  <xsd:annotation>
                     <xsd:documentation>Specifies an amount.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
               <xsd:element name="currency" type="xsd:QName" minOccurs="0">
                  <xsd:annotation>
                     <xsd:documentation>Identifies the currency by
QName.</xsd:documentation>
                  </xsd:annotation>
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Revocable">
      <xsd:annotation>
         <xsd:documentation>Identifies a SignatureValue that can be revoked. The
SignatureValue can be identified literally or by reference. In the latter case, the
result of dereferencing the reference must be of type dsig:SignatureType; the
SignatureValue being revoked is the one SignatureValue therein.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Resource">
            <xsd:choice minOccurs="0">
               <xsd:element ref="dsig:SignatureValue"/>
               <xsd:element ref="dsig:Reference"/>
            </xsd:choice>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="RightUri">
      <xsd:annotation>
         <xsd:documentation>Indicates a right using a URI.</xsd:documentation>
      </xsd:annotation>
      <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:annotation>
         <xsd:documentation>Indicates that the specified service must be contacted
and its approval gained before exercising the associated right.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="sx:StatefulCondition"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="StatefulCondition">
      <xsd:annotation>
         <xsd:documentation>A condition that requires that some state be referenced
or manipulated to be satisfied.</xsd:documentation>
      </xsd:annotation>
      <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:complexType name="StateReferenceValuePattern">
      <xsd:annotation>
         <xsd:documentation>A pattern that identifies a set of service references
using pattern matching by dereferencing their values.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:AnXmlPatternAbstract">
            <xsd:sequence>
               <xsd:any namespace="##any" processContents="lax"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Territory">
      <xsd:annotation>
         <xsd:documentation>Indicates a geographic or virtual space within which the
associated right may be exercised.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="r:Condition">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="location">
                  <xsd:annotation>
                     <xsd:documentation>Specifies a physical
location.</xsd:documentation>
                  </xsd:annotation>
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="country" type="xsd:QName"
minOccurs="0">
                           <xsd:annotation>
                              <xsd:documentation>Identifies the country by
QName.</xsd:documentation>
```

```
                              </xsd:annotation>
                           </xsd:element>
                           <xsd:element name="region" type="xsd:QName" minOccurs="0">
                              <xsd:annotation>
                                 <xsd:documentation>Identifies the region (country
subdivision) by QName.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                           <xsd:element name="state" type="xsd:string" minOccurs="0">
                              <xsd:annotation>
                                 <xsd:documentation>Identifies the state for a
geographical location.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                           <xsd:element name="city" type="xsd:string" minOccurs="0">
                              <xsd:annotation>
                                 <xsd:documentation>Identifies the city for a
geographical location.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                           <xsd:element name="postalCode" type="xsd:string"
minOccurs="0">
                              <xsd:annotation>
                                 <xsd:documentation>Identifies the postal code for a
geographical location.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                           <xsd:element name="street" type="xsd:string"
minOccurs="0">
                              <xsd:annotation>
                                 <xsd:documentation>Identifies the street address
for a geographical location.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                        </xsd:sequence>
                     </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="domain">
                     <xsd:annotation>
                        <xsd:documentation>Specifies a virtual location in the
digital domain.</xsd:documentation>
                     </xsd:annotation>
                     <xsd:complexType>
                        <xsd:sequence>
                           <xsd:element name="uri" type="xsd:anyURI">
                              <xsd:annotation>
                                 <xsd:documentation>Specifies the URI that
identifies a virtual location.</xsd:documentation>
                              </xsd:annotation>
                           </xsd:element>
                        </xsd:sequence>
                     </xsd:complexType>
                  </xsd:element>
               </xsd:choice>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="TrackQuery">
         <xsd:annotation>
```

```
          <xsd:documentation>Represents a condition on the tracking state updated by
TrackReport. For example, this condition can be used to predicate the granting of one
right on the successful exercise of another.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="sx:StatefulCondition">
             <xsd:sequence minOccurs="0">
                <xsd:element name="notLessThan" type="xsd:integer" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Specifies the minimum value that can be
contained in the state reference for this condition to be satisfied. If absent, the
value is assumed to be zero.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="notMoreThan" type="xsd:integer" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Specifies the maximum value that can be
contained in the state reference for this condition to be satisfied. If absent, the
value is assumed to be infinite.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
             </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="TrackReport">
       <xsd:annotation>
          <xsd:documentation>Indicates that exercising a right must be reported to a
designated tracking service.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="sx:StatefulCondition">
             <xsd:sequence minOccurs="0">
                <xsd:element name="communicationFailurePolicy" default="required"
minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Specifies the policy to implement if an
attempt to update the tracking information fails (for instance, if the tracking
service is offline). If the value of this element is "required" (the default), the
report must be completed successfully before this condition is considered satisfied.
If the value of this element is "lax", communication failures may be
ignored.</xsd:documentation>
                   </xsd:annotation>
                   <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:annotation>
          <xsd:documentation> Indicates that the specified service must be contacted
to confirm ownership (by the principal) of some virtual token upon which the
exercisability of this grant depends.  Typically, this service may also be used to
```

```
cause a transfer of ownership of the virtual token to another
principal.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Uddi">
        <xsd:annotation>
            <xsd:documentation>Specifies that the UDDI Business Registry (or possibly a
private UDDI registry) be used for protocol and endpoint information. Contains the
information for a UddiServiceIdentifier as defined in the UDDI
specification.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="r:ServiceDescription">
                <xsd:sequence>
                    <xsd:element name="serviceKey" type="sx:UddiKey">
                        <xsd:annotation>
                            <xsd:documentation>Indicates the service's key  in the
registry. This value should be passed to the get_serviceDetail API of the public UDDI
registry to locate the service.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="registry" type="xsd:anyURI" minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation>Identifies the UDDI registry in which the
service is located. Intended for private UDDI deployments. If absent, the global UDDI
Business Registry is implied.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="UddiKey">
        <xsd:annotation>
            <xsd:documentation>A key that identifies a business, service, or other
entity inside of UDDI.</xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:element name="uuid" type="r:Uuid">
                <xsd:annotation>
                    <xsd:documentation>The universally unique identifier (UUID), which
is used by UDDI versions 1 and 2. For more information, refer to the UDDI
specification.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="uri" type="xsd:anyURI">
                <xsd:annotation>
                    <xsd:documentation>The uniform resource identifier (URI), which is
used by UDDI version 3. For more information, refer to the UDDI
specification.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="ValidityIntervalDurationPattern">
        <xsd:annotation>
```

```
            <xsd:documentation>A pattern matching any ValidityInterval of the specified
duration.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="r:ConditionPatternAbstract">
                <xsd:sequence minOccurs="0">
                    <xsd:element name="duration" type="xsd:duration">
                        <xsd:annotation>
                            <xsd:documentation>The duration that matching
validityIntervals must express.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ValidityIntervalFloating">
        <xsd:annotation>
            <xsd:documentation>Represents an interval of time that begins with the
first exercise of a right.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="sx:StatefulCondition"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ValidityIntervalStartsNow">
        <xsd:annotation>
            <xsd:documentation>The ValidityIntervalStartsNow is satisfied at time t if
the specified ValidityInterval starts within the specified tolerances of time
t.</xsd:documentation>
        </xsd:annotation>
        <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:annotation>
                            <xsd:documentation>If present, indicates that the notBefore
element of the specified ValidityInterval must be present and greater than or equal
to the result of the current time set backwards by the value specified
here.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="forwardTolerance" type="xsd:duration"
minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation>If present, indicates that, if the
notBefore element of specified ValidityInterval is present, it must be less than or
equal to the result of the current time set forward by the value specified
here.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ValidityTimeMetered">
        <xsd:annotation>
```

```
          <xsd:documentation>Represents an accumulative period of time. A user can
start and stop exercising a right, and the metering clock runs only when the right is
being exercised. The right can be exercised as long as the total remaining time has
not been used.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="sx:StatefulCondition">
             <xsd:sequence minOccurs="0">
                <xsd:element name="quantum" type="xsd:duration" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Indicates the minimum amount of time
consumed on the meter whenever the right is exercised. If this element is absent,
minimum amount of time consumed is implementation dependent.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
             </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ValidityTimePeriodic">
       <xsd:annotation>
          <xsd:documentation>Indicates a validity time window that recurs
periodically. For example, this condition can be used to express time windows such as
"every weekend" or "the second week of every month".</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="r:Condition">
             <xsd:sequence minOccurs="0">
                <xsd:element name="start" type="xsd:dateTime">
                   <xsd:annotation>
                      <xsd:documentation>The start date and time from which the
recurrences of this time interval are calculated.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="period" type="xsd:duration">
                   <xsd:annotation>
                      <xsd:documentation>The frequency at which this time window
recurs. For example,  "P7D" represents a weekly recurrence.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="phase" type="xsd:duration" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>An optional latency from the beginning or
end of each period before the time window opens. A positive duration is measured from
the start of the period. A negative duration is measured from the end of the period.
This distinction can be important for variable sized periods such as months. If the
phase element is absent, the window opens at the start of the time
period.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="duration" type="xsd:duration">
                   <xsd:annotation>
                      <xsd:documentation>The duration of the window once it
opens.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="periodCount" type="xsd:nonNegativeInteger"
minOccurs="0">
                   <xsd:annotation>
```

```
                    <xsd:documentation>Indicates the number of periods that this
interval covers.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="WsdlAddress">
    <xsd:annotation>
      <xsd:documentation>Uses a specified portion of the identified WSDL file to
indicate the metadata and protocol information while allowing the endpoint addressing
information to be specified separately.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="r:ServiceDescription">
        <xsd:sequence minOccurs="0">
          <xsd:element name="kind">
            <xsd:annotation>
              <xsd:documentation>Identifies the abstract type of the web
service, independent of its endpoint.</xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
              <xsd:annotation>
                <xsd:documentation>Elements of this type indicate a
particular type of web service without indicating where an instance of that web
service is specifically available.</xsd:documentation>
              </xsd:annotation>
              <xsd:sequence>
                <xsd:element name="wsdl" type="r:DigitalResource">
                  <xsd:annotation>
                    <xsd:documentation>Identifies the WSDL in which the
type of the service is defined.</xsd:documentation>
                  </xsd:annotation>
                </xsd:element>
                <xsd:element name="binding" type="xsd:QName">
                  <xsd:annotation>
                    <xsd:documentation>Indicates the relevant portType
and protocol binding in the WSDL file.</xsd:documentation>
                  </xsd:annotation>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="address">
            <xsd:annotation>
              <xsd:documentation>Indicates the actual endpoint at which the
service is located.</xsd:documentation>
            </xsd:annotation>
            <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:annotation>
          <xsd:documentation>Uses a specified portion of the identified WSDL file to
indicate the full protocol and endpoint information.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent>
          <xsd:extension base="r:ServiceDescription">
             <xsd:sequence minOccurs="0">
                <xsd:element name="wsdl" type="r:DigitalResource">
                   <xsd:annotation>
                      <xsd:documentation>Identifies a particular WSDL
file.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="service" type="xsd:QName">
                   <xsd:annotation>
                      <xsd:documentation>Identifies a particular service within the
WSDL file. WSDL services have zero or more ports and a binding between each port and
an endpoint address. All ports of the same portType are considered
equivalent.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
                <xsd:element name="portType" type="xsd:QName" minOccurs="0">
                   <xsd:annotation>
                      <xsd:documentation>Identifies a specific port type if the
WSDL service contains ports of more than one portType. For more information, refer to
the WSDL specification.</xsd:documentation>
                   </xsd:annotation>
                </xsd:element>
             </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="X509SubjectName" mixed="true">
       <xsd:annotation>
          <xsd:documentation>The subject name of some X509 certificate associated
with the entity. Intended to address legacy interoperability issues involving X509
certificates.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent mixed="true">
          <xsd:extension base="sx:Name"/>
       </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="X509SubjectNamePattern" mixed="true">
       <xsd:annotation>
          <xsd:documentation>A pattern that identifies a set of X509 subject names
using pattern matching.</xsd:documentation>
       </xsd:annotation>
       <xsd:complexContent mixed="true">
          <xsd:extension base="r:ResourcePatternAbstract"/>
       </xsd:complexContent>
    </xsd:complexType>
    <!-- Groups -->
    <!-- Simple Types -->
</xsd:schema>
```

Multimedia Extension

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema targetNamespace="urn:mpeg:mpeg21:2002:01-REL-NS"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:r="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:sx="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:mx="urn:mpeg:mpeg21:2002:01-REL-NS" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xsd:import namespace="http://www.xrml.org/schema/2002/05/xrml2sx"
schemaLocation="xrml2sx.xsd"/>
   <xsd:import namespace="http://www.xrml.org/schema/2002/05/xrml2core"
schemaLocation="xrml2core.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="Extract">
      <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="extract" type="mx:Extract" 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="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="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 minOccurs="0">
               <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 minOccurs="0">
               <xsd:element ref="r:principal"/>
            </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">
        <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">
        <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">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="r:LicensePart">
                <xsd:sequence minOccurs="0">
                   <xsd:any namespace="##any" 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">
       <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

(informative)

Example Rights Expressions

B.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 targetted interpretation) that would make such participation not only possible, but exceedingly practical as well.

Interpretation of the distribution license is somewhat more involved than the end user license, but this is the tradeoff 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 consortia 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 favorite 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.

B.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"?>
<license
xmlns="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:sx="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:mx="urn:mpeg:mpeg21:2002:01-REL-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:01-REL-NS mpeg-rel.xsd">
   <grant>
      <keyHolder licensePartId="John">
         <info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>KtdToQ...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </info>
      </keyHolder>
      <mx:play/>
      <mx:diReference>
         <mx:identifier>urn:grid:a1-abcde-1234567890-f</mx:identifier>
      </mx:diReference>
      <validityInterval>
         <notBefore>2003-01-01T00:00:00</notBefore>
         <notAfter>2003-12-31T12:59:59</notAfter>
      </validityInterval>
   </grant>
   <!--The license is issued by Xin, the distributor.-->
   <issuer>
      <keyHolder>
         <info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>X0j9q9...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </info>
      </keyHolder>
   </issuer>
</license>
```

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

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

The `issuer` is Xin, who authorizes the `grant`. (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 `issuer` to guarantee the authenticity and integrity inline.) The `grant` contains four parts:

- `keyHolder`: identifies that John is being granted the rights.
- `play`: identifies the right that John is being granted.
- `diReference`: identifies the digital item, `urn:grid:a1-abcde-1234567890-f`, that John is being granted rights to.
- `validityInterval`: identifies the time interval, 2003, during which John is allowed to play the digital item.

B.3 Distribution License Example

In order to issue the end user license shown in Clause B.2, Xin needs to have authorization from Alice to do so. Alice, however, does not want to give Xin a separate license for each possible person he may want to issue to. 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 authorization in the following ways:

- Each time Xin issues a grant pursuant to her authorization, Xin must pay her $1.
- The grants that Xin issues pursuant to her authorization 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 grants that Xin issues pursuant to her authorization must be grants to principals who are members of Acme music club at the time he issues the grants.

Alice gives Xin the following license:

```
<?xml version="1.0" encoding="UTF-8"?>
<license
xmlns="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:sx="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:mx="urn:mpeg:mpeg21:2002:01-REL-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:01-REL-NS mpeg-rel.xsd">
   <grant>
      <forAll varName="AcmeMusicClubMember">
         <everyone>
            <sx:propertyUri definition="urn:acme:musicClubMember"/>
            <trustedIssuer>
               <keyHolder licensePartId="Acme">
                  <info>
                     <dsig:KeyValue>
                        <dsig:RSAKeyValue>
                           <dsig:Modulus>aaaM4c...yzA==</dsig:Modulus>
                           <dsig:Exponent>AQABAA==</dsig:Exponent>
                        </dsig:RSAKeyValue>
                     </dsig:KeyValue>
                  </info>
               </keyHolder>
            </trustedIssuer>
         </everyone>
      </forAll>
      <keyHolder licensePartId="Xin">
         <info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>X0j9q9...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </info>
      </keyHolder>
      <issue/>
      <grant>
         <principal varRef="AcmeMusicClubMember"/>
         <mx:play/>
         <mx:diReference>
            <mx:identifier>urn:grid:a1-abcde-1234567890-f</mx:identifier>
         </mx:diReference>
         <validityInterval>
            <notBefore>2003-01-01T00:00:00</notBefore>
            <notAfter>2003-12-31T12:59:59</notAfter>
         </validityInterval>
      </grant>
      <sx:fee>
         <sx:paymentPerUse>
            <sx:rate>
               <sx:amount>1.00</sx:amount>
            </sx:rate>
         </sx:paymentPerUse>
         <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:fee>
   </grant>
   <!--The license is issued and signed by Alice, the author of the digital item.-->
   <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=
                     "http://www.xrml.org/schema/2002/05/xrml2core#license"/>
               </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>DFgqOh...5QQ==</dsig:SignatureValue>
         <dsig:KeyInfo>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>gOyM4c...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </dsig:KeyInfo>
      </dsig:Signature>
   </issuer>
</license>
```

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

```
<license ...>
   <grant>
      <forAll varName="AcmeMusicClubMember">...</forAll>
      <keyHolder licensePartId="Xin">...</keyHolder>
      <issue/>
      <grant>
         <principal varRef="AcmeMusicClubMember"/>
         <mx:play/>
         <mx:diReference>...</mx:diReference>
         <validityInterval>...</validityInterval>
      </grant>
      <sx:fee>...</sx:fee>
   </grant>
   <issuer>
      <dsig:Signature>...</dsig:Signature>
   </issuer>
</license>
```

In this case, the `issuer` contains Alice's signature (XML Digital Signature) over the license, thus authorizing the `grant`. The `grant` contains five parts:

- `forAll`: this defines the variable which will allow Alice to issue one license that Xin can use to distribute licenses to several different end users.
- `keyHolder`: identifies that Xin is being granted rights.
- `issue`: identifies the right that Xin is being granted.
- `grant`: indicates that Xin is allowed to issue a grant.
- `fee`: identifies the account to which and amount that Xin must pay each time he issues a grant. (This is how Alice indicates that each time Xin issues a grant pursuant to her authorization, Xin must pay her $1.)

Of these five parts of the outer `grant`, the inner `grant` deserves closer inspection: it contains four parts.

The last three of these four parts (of the inner `grant`) are the same as the last three parts of the `grant` in the end user license. This is the way that Alice indicates to Xin that the end user grants he issues pursuant to her authorization 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 first of these four parts (of the inner `grant`) has a `varRef`, which is a reference to a variable ("AcmeClubMember") that Xin must resolve when he issues a grant. To determine what values this variable may take on, Xin must find a `forAll` element that declares the corresponding `varName`.

Looking at the `forAll` element that declares "AcmeMusicClubMember", we notice the presence of an `everyone` pattern that contains two parts: a `propertyUri` and a `trustedIssuer` set to Acme, as shown in the following skeletal outline.

```
<forAll varName="AcmeMusicClubMember">
   <everyone>
      <sx:propertyUri definition="urn:acme:musicClubMember"/>
      <trustedIssuer>...</trustedIssuer>
   </everyone>
</forAll>
```

This `forAll` element signifies that the valid values for the "AcmeMusicClubMember" variable are all those principals for which another `license` exists in which the trusted issuer grants that principal the right to possess the indicated property.

Because John is requesting a license from Xin, Xin will want to determine if John is a valid value for the "AcmeMusicClubMember" variable. Therefore, before Xin issues a grant to John, Xin must find a license issued by Acme that grants John the right to possess the property `urn:acme:musicClubMember`. This `license` may look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<license
xmlns="http://www.xrml.org/schema/2002/05/xrml2core"
xmlns:sx="http://www.xrml.org/schema/2002/05/xrml2sx"
xmlns:mx="urn:mpeg:mpeg21:2002:01-REL-NS"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:01-REL-NS mpeg-rel.xsd">
   <grant>
      <keyHolder licensePartId="John">
         <info>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>KtdToQ...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </info>
      </keyHolder>
      <possessProperty/>
      <sx:propertyUri definition="urn:acme:musicClubMember"/>
   </grant>
   <!--The license is issued and signed by Acme.-->
   <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=
                     "http://www.xrml.org/schema/2002/05/xrml2core#license"/>
               </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>ABCqOh...5QQ==</dsig:SignatureValue>
         <dsig:KeyInfo>
            <dsig:KeyValue>
               <dsig:RSAKeyValue>
                  <dsig:Modulus>aaaM4c...yzA==</dsig:Modulus>
                  <dsig:Exponent>AQABAA==</dsig:Exponent>
               </dsig:RSAKeyValue>
            </dsig:KeyValue>
         </dsig:KeyInfo>
      </dsig:Signature>
   </issuer>
</license>
```

As illustrated in the following skeletal license outline below, the above `license` has a `grant` of three parts (that says John has the right to possess the property `urn:acme:musicClubMember`) and an `issuer` that contains the signature of Acme.

```
<license ...>
   <grant>
      <keyHolder licensePartId="John">...</keyHolder>
      <possessProperty/>
      <sx:propertyUri definition="urn:acme:musicClubMember"/>
   </grant>
   <issuer>
      <dsig:Signature>...</dsig:Signature>
   </issuer>
</license>
```

If Xin has these two licenses (the one from Acme plus the one from Alice) and pays $1 according to the license from Alice, Xin can issue John the simple end user license shown in Clause B.2.

Annex C

(informative)

Extension Mechanisms for Introducing New Rights

This annex illustrates how a new right can be introduced into REL expressions, using several extension mechanisms provided in REL.

The illustrative right is "copy" as defined by a hypothetical institution Acme as:
"Make a single bit-for-bit version of a digital resource".

This right can be introduced into REL expressions using any one of the following four extension mechanisms:

- use existing rights and conditions
- use `rightUri`
- use type extension (`xsi:type`)
- use element extension (`substitutionGroup`)

C.1 Use Existing Rights and Conditions

In many cases, a new right can be introduced implicitly by using existing (or extended) conditions to specialize (constrain or contextualize) existing rights.

According to the semantics of Acme's "copy", an exercise of this right on a digital resource results in generating a new digital resource and maintaining the original unchanged. This is a specialization of the existing REL right `mx:adapt`. What is special about Acme's "copy" is a constraint about the comparison between the new and original resources (they must have the same number of bits and match "bit-for-bit"). This implies that a list of attributes related to the bit-to-bit equivalence to the original resource must be preserved for the new resource. Thus, Acme's "copy" right can be introduced as a specialization of `mx:adapt` using `mx:prohibitedAttributeChanges` for the list of attributes to be preserved.

Now the issue is how to identify this attribute list within `mx:prohibitedAttributeChanges`. If Acme has the list specified, say at some URI `urn:acme:copy-preservedAttributes`, then Acme's "copy" right can be introduced as a specialization of `mx:adapt` using this list, as shown in the following `r:grant`:

```
<r:grant>
   <r:keyHoler>
      …
   </r:keyHoler>
   <mx:adapt/>
   <mx:diReference>
      <mx:identifier>someResourceUri</mx:identifier>
   </mx:diReference>
   <mx:prohibitedAttributeChanges>
      <mx:set definition="urn:acme:copy-preservedAttributes"/>
   </mx:prohibitedAttributeChanges>
</r:grant>
```

The list can also be specified using some hypothetically existing RDD attributes. For instance, Acme can consider two RDD attributes, `numberOfBits` and `bitSequence`, as the ones that must be preserved. Assume that these two RDD attributes have been defined in RDD with Term IDs as 2346 and 2347 and that they can be identified with the respective URIs:

```
urn:mpeg:mpeg21:2002:01-RDD-NS:2346     (numberOfBits)
urn:mpeg:mpeg21:2002:01-RDD-NS:2347     (bitSequence)
```

Then, Acme’s “copy” right can be introduced as a specialization of `mx:adapt` using `mx:prohibitedAttributeChanges` as shown in the following `r:grant`:

```
<r:grant>
   <r:keyHoler>
      …
   </r:keyHoler>
   <mx:adapt/>
   <mx:diReference>
      <mx:identifier>someResourceUri</mx:identifier>
   </mx:diReference>
   <mx:prohibitedAttributeChanges>
      <set definition=”urn:mpeg:mpeg21:2002:01-RDD-NS:2346”/>
      <set definition=”urn:mpeg:mpeg21:2002:01-RDD-NS:2347”/>
   </mx:prohibitedAttributeChanges>
</r:grant>
```

C.2 Use `rightUri`

The standard extension schema of REL defines a type called `sx:RightUri`. An `sx:RightUri` indicates a right using a URI rather than an XML Schema element or type. The semantics of the right being indicated by a `r:RightUri` is determined by the URI value of its attribute `definition`.

Using this mechanism, Acme’s right “copy” can be introduced as in the following `r:grant` (note that the URI value "`urn:acme:copy`" is used only for illustration):

```
<r:grant>
   <r:keyHoler>
      …
   </r:keyHoler>
   <sx:rightUri definition="urn:acme:copy"/>
   <mx:diReference>
      <mx:identifier>someResourceUri</mx:identifier>
   </mx:diReference>
</r:grant>
```

If Acme’s “copy” is mapped into RDD (according to the RDD mapping process), the corresponding RDD Term ID, say `urn:mpeg:mpeg21:2002:01-RDD-NS:2348`, can then be specified as the URI value of `definition` as shown below:

```
<r:grant>
   <r:keyHoler>
      …
   </r:keyHoler>
   <sx:rightUri definition="urn:mpeg:mpeg21:2002:01-RDD-NS:2348"/>
   <mx:diReference>
      <mx:identifier>someResourceUri</mx:identifier>
   </mx:diReference>
</r:grant>
```

C.3 Use Type Extension (`xsi:type`)

The W3C XML Schema mechanism of deriving types by extension using the attribute `xsi:type` is another way to add new rights (and other building blocks) into REL. The semantics of a new right introduced in this manner is determined by the extension type specified.

Assume that Acme’s right “copy” has already had an XML Schema type definition in some XML namespace called “`acme`” that extends the XML Schema type `r:Right` in REL, as follows:

```
   <xsd:complexType name="Copy">
      <xsd:complexContent>
         <xsd:extension base="r:Right"/>
      </xsd:complexContent>
   </xsd:complexType>
```

Then, Acme's right "copy" can be introduced as in the following `r:grant`:

```
   <r:grant>
      <r:keyHoler>
         …
      </r:keyHoler>
      <r:right xsi:type="acme:Copy"/>
      <mx:diReference>
         <mx:identifier>someResourceUri</mx:identifier>
      </mx:diReference>
   </r:grant>
```

However, this mechanism only presents the generic element `r:right` (though with a specific, new type); it does not provide the flexibility to introduce new right elements such as `acme:copy`.

C.4 Use Element Extension (`substitutionGroup`)

New rights can be introduced as derivations of the type `r:Right` and substitutions of the element `r:right` in REL, via a W3C XML Schema extension mechanism, `xsd:substitutionGroup`, that is built in the W3C XML Schema.

Acme's right "copy" can be introduced using this mechanism within a namespace `acme`, as follows:

```
<xsd:element name="copy" type="acme:Copy" substitutionGroup="r:right"/>
```

Then this right element `acme:copy` can be used in an `r:grant` like the following to grant some principal the right to "copy" a digital item:

```
   <r:grant>
      <r:keyHoler>
         …
      </r:keyHoler>
      <acme:copy/>
      <mx:diReference>
         <mx:identifier>someResourceUri</mx:identifier>
      </mx:diReference>
   </r:grant>
```

Annex D

(informative)

Relationship Between ISO/IEC 21000-5 (REL) and ISO/IEC 21000-6 (RDD)

There are four specific mechanisms by which Terms defined within the RDD may be represented in the REL.

D.1 REL “Multimedia Rights” and RDD ActTypes

REL defines a set of XML Schema Complex Types that, in the XML Schema sense, derive from (either extend or restrict) the conceptually abstract type `Right` (from the REL core namespace). Some of these types reside in the `urn:mpeg:mpeg21:2002:01-REL-NS` namespace. For convenience, in this Clause these types are called “Multimedia Rights”.

Each activity has a context that can be related to particular ActType(s) within the RDD ontology. An activity is said to be within the scope of a particular Multimedia Right if the activity’s context is a contextualization of the ActType corresponding to that Multimedia Right. The ActTypes corresponding to the Multimedia Rights are given in the following table.

REL “Multimedia Right”	**RDD ActType**
`Modify`	Modify
`Enlarge`	Enlarge
`Reduce`	Reduce
`Move`	Move
`Adapt`	Adapt
`Extract`	Extract
`Embed`	Embed
`Play`	Play
`Print`	Print
`Execute`	Execute
`Install`	Install
`Uninstall`	Uninstall
`Delete`	Delete

D.2 Other RDD ActTypes as REL Rights

RDD ActTypes other than those in the table above can be expressed using REL with XML Schema Complex Types that derive from the conceptually abstract type `Right` (from the REL core namespace).

D.3 RDD ResourceTypes as REL Resources

RDD ResourceTypes can be expressed using REL with XML Schema Complex Types that derive from the conceptually abstract type `Resource` (from the REL core namespace).

D.4 RDD ContextTypes as REL Conditions

RDD ContextTypes can be indicated using REL with XML Schema Complex Types that derive from the conceptually abstract type `Condition` (from the REL core namespace).

Bibliography

[1] XML, *Extensible Markup Language 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[2] XML Schema, *XML Schema Part 1: Structures and Part 2: Datatypes*, W3C Recommendation, 2 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502.

[3] Canonical XML, *Canonical XML Version 1.0*, W3C Recommendation, 15 March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315.

[4] *Multimedia Framework* ISO/IEC 21000 (all parts).

[5] RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC 2396, August 1998.

[6] RFC 2141, *Uniform Resource Names (URN)*, IETF RFC 2141, May 1997.

[7] RFC 1738, *Uniform Resource Locators (URL)*, IETF RFC 1738, December 1994.

[8] RFC 2119, *Key words for use in RFCs to Indicate Requirements Levels*, IETF RFC 2119, March 1997.

[9] XML Digital Signature, *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212.

[10] Exclusive XML Canonicalization, *Exclusive XML Canonicalization Version 1.0*, W3C Recommendation, 18 July 2002, http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718.

[11] XML Namespaces, *Namespaces in XML*, W3C Recommendation, 14 January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114.

[12] Schema Centric XML Canonicalization, *Schema Centric XML Canonicalization Version 1.0*, UDDI Version 3, 10 July 2002, http://uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm.

[13] UDDI, *Universal Description, Discovery, and Integration (UDDI)*, http://www.uddi.org/.

[14] WSDL, *Web Services Definition Language (WSDL) 1.1*, W3C Note, 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[15] XML Encryption, *XML-Encryption Syntax and Processing*, W3C Recommendation, 10 December 2002, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210.

[16] XML Infoset, *XML-Information Set*, W3C Recommendation, 24 October 2001, http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[17] XPath, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116.

[18] XPath2, *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C Working Draft, 30 April 2002, http://www.w3.org/TR/2002/WD-xquery-operators-20021115.

[19] *Codes for the representation of names of countries and their subdivisions*, ISO 3166 (all parts).

[20] *Codes for the representation of currencies and funds*, ISO 4217 (all parts).