CONTENTGUARD™

eXtensible rights Markup Language (XrML) 2.0 Specification
Part III: Standard Extension Schema

20 November 2001

Available formats: HTML and PDF. In case of a discrepancy, the HTML is considered definitive.

NOTE: To enable interactive browsing of the XrML schemas and examples, the XrML Specification and its companion Example Use Cases document use an HTML version that leverages the XML access functionality provided by the W3C Xpath recommendation. For this reason, you need to view these HTML documents with a browser that supports that recommendation (for example, Internet Explorer Version 6.0). If your browser does not support this functionality, please view the PDF versions of those documents.

Quick Table of Contents

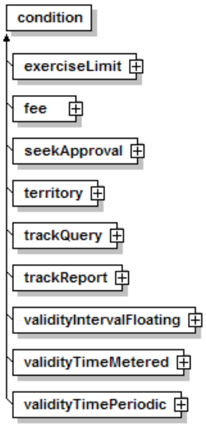
Full Table of Contents for Part III: Standard Extension Schema

6 Standard Extension

Some concepts arise commonly in many XrML usage scenarios, but the elements and types that are used to capture them do not really belong in the core. The extensions defined in the Standard Extension are classified according to the purpose that they serve. They are broadly classified into conditions, payment notions, name, and revocation extensions.

6.1 Condition Extensions

Condition Extensions



The standard extension schema defines nine extensions of the type `Condition`. Of these, six extensions require a notion of communication with an external authoritative value. To facilitate the definition of these extensions, a type `StatefulCondition` is defined as well. Condition extensions are constructed either by extending `StatefulCondition` type or by directly extending the `Condition` type. Naturally, what it means for a condition extension to be satisfied is left to its description.

6.1.1 StatefulCondition

Some conditions may be tied to an external authoritative piece of data. These arise when the exercise of a right is predicated upon querying or manipulating the value of this external piece of data. For example, the number of times some content may be rendered can be bounded by some value. To cover such usages, XrML2.0 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 `stateReference`, which indicates the means with which communication is made with the state or service. The state holds the piece of data and returns it in the form of one or more XML nodes when queried for its value. The schema that describes the XML nodes returned by the state is left to the specific `Condition` that extends `StatefulCondition`. Manipulations of the state value must also be permitted. Elements of type `StatefulCondition` (or a derivation thereof) must define specifically their interaction with the state.

Schema representation of StatefulCondition

```
-<xsd:complexType name="StatefulCondition">
  -<xsd:complexContent>
    -<xsd:extension base="r:Condition">
      -<xsd:sequence minOccurs="0">
          <xsd:element ref="sx:stateReference" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

6.1.1.1 StatefulCondition/stateReference

The element `stateReference` indicates the means by which the state is to be queried or manipulated. It has type `ServiceReference` as defined in the core.

6.1.2 StateReferenceValuePattern

A pattern that identifies a set of state reference values using pattern matching.

Schema representation of `StateReferenceValuePattern`

```
-<xsd:complexType name="StateReferenceValuePattern">
  -<xsd:complexContent>
    -<xsd:extension base="r:XmlPatternAbstract">
      -<xsd:sequence>
          <xsd:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `stateReferenceValuePattern`

```
-<xsd:element name="stateReferenceValuePattern" type="sx:StateReferenceValuePattern" substitutionGroup="r:xmlPatternAbstract">
  </xsd:element>
```

The element `stateReferenceValuePattern` specifies an unbounded number of XML element nodes. This element matches a `stateReference` if and only if for every XML element node that is a sub element of the `stateReferenceValuePattern` element, there exists an identical XML element node in the set of XML element nodes returned by the state designated by the `stateReference`, when queried for its value.

Using the stateReferenceValuePattern element

The element `stateReferenceValuePattern` is typically used with the `obtain` element. When the right to `obtain` a `grant` with a stateful condition s issued, then to specify the initial values the state value happen to take, a `stateReferenceValuePattern` is created containing the XML nodes hat correspond to the desired state value. This ensures that only `stateReference`s whose designated states have the requisite state values appear in the obtained `grant`.

The following example illustrates a `grant` issued to Alice that allows her to obtain another `grant` to play "A Clockwork Orange" up to twenty imes. The obtained `grant` uses the stateful condition `ExerciseLimit`.

Example use of stateReferenceValuePattern

```
-<grant>
  -<forAll varName="stateX">
    -<patternFromLicensePart>
      -<sx:stateReference>
        -<uddi>
          -<serviceKey>
              <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
            </serviceKey>
          </uddi>
        </sx:stateReference>
      </patternFromLicensePart>
    -<sx:stateReferenceValuePattern>
        <sx:count>20</sx:count>
      </sx:stateReferenceValuePattern>
    </forAll>
  -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <obtain/>
  -<grant>
    -<keyHolder licensePartIdRef="Alice">
      </keyHolder>
      <cx:play/>
      <cx:digitalWork licensePartIdRef="AClockworkOrange"/>
    -<sx:exerciseLimit>
        <sx:stateReference varRef="stateX"/>
      </sx:exerciseLimit>
    </grant>
  </grant>
```

5.1.3 The ExerciseLimit Condition

ndicates a maximum number of times that the right may be exercised.

Schema representation of `ExerciseLimit`

```
-<xsd:complexType name="ExerciseLimit">
  -<xsd:complexContent>
      <xsd:extension base="sx:StatefulCondition"/>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `exerciseLimit`

```
-<xsd:element name="exerciseLimit" type="sx:ExerciseLimit" substitutionGroup="r:condition">
  </xsd:element>
```

Using the exerciseLimit element

Sometimes `grant`s have an upper bound on the number of times the associated rights may be exercised. This can be a certain number of times a document may be printed or the number of times a piece of music may be played. The element `exerciseLimit` captures these cases. The element has type `ExerciseLimit`, which is based on `StatefulCondition`. Typically, all of the information needed to evaluate an `exerciseLimit` condition is obtained by querying the state designated by `stateReference`, including the initial value of the limit associated with the `grant`.

nteraction with the State

When a right conditioned by `exerciseLimit` is exercised, the state designated by the `stateReference` in `exerciseLimit` is queried for its value. The state returns `count` containing an `integer` value `c`. Upon exercise of the right, the value of the state is updated to return `count` containing `c - 1` hereon.

The `exerciseLimit` condition is satisfied if and only if `c` is greater than zero and the value of the state is updated as described above.

In the following example, Alice may `print` the book, but only for a certain number of times.

Example use of `exerciseLimit`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:exerciseLimit>
    -<sx:stateReference>
      -<uddi>
        -<serviceKey>
           <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
         </serviceKey>
       </uddi>
     </sx:stateReference>
   </sx:exerciseLimit>
 </grant>
```

6.1.4 The SeekApproval Condition

Indicates that the specified service must be contacted and its approval gained before exercising the associated right.

Schema representation of `SeekApproval`

```
-<xsd:complexType name="SeekApproval">
  -<xsd:complexContent>
     <xsd:extension base="sx:StatefulCondition"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `seekApproval`

```
-<xsd:element name="seekApproval" type="sx:SeekApproval" substitutionGroup="r:condition">
 </xsd:element>
```

Using the seekApproval element

A `grant` may have rights that may be exercised only upon specific approval from a service. The element `seekApproval` of type `SeekApproval` is used to model these cases. `seekApproval` is based on the type `StatefulCondition`. The exercise of such a right typically involves querying the appropriate state for approval. The means to connect to the state is made available with the child element `stateReference`.

Interaction with the State

When a right conditioned by `seekApproval` is exercised, the state designated by the `stateReference` in `seekApproval` is queried for its value. The state returns `approval` containing a `boolean` value `b`.

The `seekApproval` condition is satisfied if and only if `b` is true.

In the following example, Alice may `print` the book, but only upon prior approval from the designated state reference.

Example use of `seekApproval`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:seekApproval>
    -<sx:stateReference>
      -<uddi>
        -<serviceKey>
           <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
         </serviceKey>
       </uddi>
     </sx:stateReference>
   </sx:seekApproval>
 </grant>
```

6.1.5 The TrackReport Condition

Indicates that exercising a right must be reported to a designated tracking service.

Schema representation of `TrackReport`

```
-<xsd:complexType name="TrackReport">
  -<xsd:complexContent>
    -<xsd:extension base="sx:StatefulCondition">
      -<xsd:sequence minOccurs="0">
```

```
        -<xsd:element name="communicationFailurePolicy" default="required" minOccurs="0">
          -<xsd:simpleType>
            -<xsd:restriction base="xsd:NMTOKEN">
               <xsd:pattern value="lax"/>
               <xsd:pattern value="required"/>
             </xsd:restriction>
           </xsd:simpleType>
         </xsd:element>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `trackReport`

```
-<xsd:element name="trackReport" type="sx:TrackReport" substitutionGroup="r:condition">
 </xsd:element>
```

.1.5.1 *TrackReport/communicationFailurePolicy*

This locally defined element is used to indicate what should happen if communication with the designated state should fail. With a setting of "lax" ommunication failures may be ignored and the right may be exercised, whereas a setting of "required" would prevent exercises. The default is required".

sing the trackReport element

The `trackReport` condition requires that the exercise of the associated right is reported to a designated state. The `trackReport` element is ased on the `TrackReport` type, which is an extension of `StatefulCondition`. The means to connect to the state is made available with the hild element `stateReference`. A child element, `communicationFailurePolicy`, allows the setting of policy regarding the exercise should the ommunication with the state fail.

nteraction with the State

hen a right conditioned by `trackReport` is exercised, the state designated by the `stateReference` is notified of the exercise. The state esponds in some fashion to confirm the notification.

xcept when the `communicationFailurePolicy` is set to `lax`, the `trackReport` condition is satisfied if and only if the state is successfully notified f the exercise of the right. Note that `trackReport` does not define how the state records the report. This may be done with a `boolean` value or an `integer` value.

n the following example, Alice has the right to `print` a book, but the exercise of that right must be reported.

Example use of `trackReport`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:trackReport>
    -<sx:stateReference licensePartId="stateReferenceAlias">
      -<uddi>
        -<serviceKey>
           <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
         </serviceKey>
       </uddi>
     </sx:stateReference>
     <sx:communicationFailurePolicy>required</sx:communicationFailurePolicy>
   </sx:trackReport>
 </grant>
```

.1.6 The TrackQuery Condition

epresents a condition on the tracking state updated by TrackReport. For example, this condition can be used to predicate the granting of one ight on the successful exercise of another.

Schema representation of `TrackQuery`

```
-<xsd:complexType name="TrackQuery">
  -<xsd:complexContent>
    -<xsd:extension base="sx:StatefulCondition">
      -<xsd:sequence minOccurs="0">
        -<xsd:element name="notLessThan" type="xsd:integer" minOccurs="0">
         </xsd:element>
        -<xsd:element name="notMoreThan" type="xsd:integer" minOccurs="0">
         </xsd:element>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `trackQuery`

```
-<xsd:element name="trackQuery" type="sx:TrackQuery" substitutionGroup="r:condition">
 </xsd:element>
```

Using the trackQuery element

The `trackQuery` condition represents a condition on a specific value that a state holds. The `trackQuery` element is based on the `TrackQuery` type, which is an extension of `StatefulCondition`. The means to connect to the state is made available with the child element `stateReference`. For the `trackQuery` condition to be satisfied the state must have a value within the range is specified by two child elements `notMoreThan` and `notLessThan`.

Interaction with the State

When a `grant` conditioned by `trackQuery` is exercised, the state designated by the `stateReference` is queried for its value. The state returns `count` containing an `integer` `v`.

The `trackQuery` condition is satisfied if and only if `v` lies between the values contained in `notMoreThan` and `notLessThan`.

In the following example, Alice can `print` the book only when the designated state value is between five and ten.

Example use of `trackQuery`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:trackQuery>
    -<sx:stateReference licensePartId="stateReferenceAlias">
      -<uddi>
        -<serviceKey>
           <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
         </serviceKey>
       </uddi>
     </sx:stateReference>
     <sx:notLessThan>10</sx:notLessThan>
     <sx:notMoreThan>5</sx:notMoreThan>
   </sx:trackQuery>
 </grant>
```

`TrackQuery` is commonly used together with `trackReport`, to model exercise of rights predicated on the exercise of other rights. The following example demonstrates one such usage. The idea is that Alice may listen to a piece of music as many times as she pleases provided she has listened to some commercial. The `grant` related to the commercial has a `trackReport` condition. When Alice attempts to listen to the piece of music, the `trackQuery` condition herein allows exercise of the right only when the state value tracked by the `trackReport` condition has a value greater than zero.

Example use of `trackQuery` with `trackReport`

```
-<license>
  -<inventory>
    -<keyHolder licensePartId="Alice">
      -<info>
        -<dsig:KeyValue>
          -<dsig:RSAKeyValue>
             <dsig:Modulus>n5gzmvv4/...</dsig:Modulus>
             <dsig:Exponent>AQABAA==</dsig:Exponent>
           </dsig:RSAKeyValue>
         </dsig:KeyValue>
       </info>
     </keyHolder>
    -<digitalResource licensePartId="Commercial">
      -<cx:metadata>
        -<xml>
          -<cx:simpleDigitalWorkMetadata>
             <cx:title>Toyota Ad</cx:title>
           </cx:simpleDigitalWorkMetadata>
         </xml>
       </cx:metadata>
     </digitalResource>
    -<digitalResource licensePartId="music">
      -<dsig:Reference URI="http://www.server.com/downloads/anInterestingSong.mp3">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</dsig:DigestValue>
       </dsig:Reference>
     </digitalResource>
   </inventory>
  -<grant>
    -<keyHolder licensePartIdRef="Alice">
     </keyHolder>
     <cx:play/>
     <cx:digitalWork licensePartIdRef="Commercial"/>
    -<sx:trackReport>
      -<sx:stateReference licensePartId="AdState">
        -<uddi>
          -<serviceKey>
             <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
           </serviceKey>
         </uddi>
       </sx:stateReference>
       <sx:communicationFailurePolicy>lax</sx:communicationFailurePolicy>
     </sx:trackReport>
   </grant>
  -<grant>
    -<keyHolder licensePartIdRef="Alice">
     </keyHolder>
     <cx:play/>
     <cx:digitalWork licensePartIdRef="Music"/>
   -<sx:trackQuery>
     -<sx:stateReference licensePartId="AdState">
       -<uddi>
         -<serviceKey>
            <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
          </serviceKey>
        </uddi>
      </sx:stateReference>
      <sx:notLessThan>1</sx:notLessThan>
    </sx:trackQuery>
  </grant>
 -<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/2001/11/xrml2core#license"/>
          </dsig:Transforms>
          <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>PB4QbKOQCo941tTExbj1/Q==</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>zIRYaxl5E...</dsig:SignatureValue>
     -<dsig:KeyInfo>
       -<dsig:KeyValue>
         -<dsig:RSAKeyValue>
            <dsig:Modulus>g8NRYMG30...</dsig:Modulus>
            <dsig:Exponent>AQABAA==</dsig:Exponent>
          </dsig:RSAKeyValue>
        </dsig:KeyValue>
      </dsig:KeyInfo>
    </dsig:Signature>
   -<details>
      <timeOfIssue>2001-01-27T15:30:00</timeOfIssue>
     -<validityInterval>
        <notBefore>2000-02-03T17:26:00</notBefore>
        <notAfter>3000-02-03T17:26:00</notAfter>
      </validityInterval>
    </details>
  </issuer>
</license>
```

5.1.7 The ValidityIntervalFloating Condition

Represents an interval of time that begins with the first exercise of a right.

Schema representation of `ValidityIntervalFloating`

```
-<xsd:complexType name="ValidityIntervalFloating">
  -<xsd:complexContent>
     <xsd:extension base="sx:StatefulCondition"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `ValidityIntervalFloating`

```
-<xsd:element name="validityIntervalFloating" type="sx:ValidityIntervalFloating" substitutionGroup="r:condition">
 </xsd:element>
```

Using the validityIntervalFloating element

`validityIntervalFloating` is a temporal condition that is used to indicate for what length of time a right may be exercised. For example, we may speak of a `grant` that provides a right that can be exercised for one week. The element `validityIntervalFloating` is based on the type `ValidityIntervalFloating`, which is based on `StatefulCondition`. The semantic of the condition expressed is that the interval of exercise of he right to which a `validityIntervalFloating` is applied must lie wholly within some contiguous duration of time.

nteraction with the state

When a right conditioned by `validityIntervalFloating`is exercised, the state designated by the `stateReference` is queried for its value. The state returns EITHER a) `validUntil` holding a `dateTime` value `v` OR b) `validFor` holding a `duration` value `d`. It the state returns `d`, and the ight is currently to be exercised, then it is updated to return validUntil from hereon with a value equal to `c + d`, where `c` is the current time.

The `validityIntervalFloating` condition is satisfied if and only if the interval of exercise of the right lies wholly within the interval `{from: c, to: v}` or within the interval `{from: c, to: c+ d}`

n the following example, Alice can exercise the right to `print` the book only when time has not run out of the state clock.

Example use of `validityIntervalFloating`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:validityIntervalFloating>
    -<sx:stateReference licensePartId="stateReferenceAlias">
      -<uddi>
        -<serviceKey>
        <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
      </serviceKey>
    </uddi>
  </sx:stateReference>
 </sx:validityIntervalFloating>
</grant>
```

5.1.8 The ValidityTimeMetered Condition

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.

Schema representation of `ValidityTimeMetered`

```
-<xsd:complexType name="ValidityTimeMetered">
  -<xsd:complexContent>
    -<xsd:extension base="sx:StatefulCondition">
      -<xsd:sequence minOccurs="0">
        -<xsd:element name="quantum" type="xsd:duration" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `validityTimeMetered`

```
-<xsd:element name="validityTimeMetered" type="sx:ValidityTimeMetered" substitutionGroup="r:condition">
  </xsd:element>
```

5.1.8.1 validityTimeMetered/quantum

A locally defined element of type `integer`. It serves to indicate the period with which the metered time is measured. For example, a quantum of one hour would meter every usage to the closest hour.

Using the validityTimeMetered element

The `validityTimeMetered` condition that is used to indicate for what non-contiguous length of time a right may be exercised. Unlike the `validityIntervalFloating` condition, the length in question only takes into account the periods of time of actual exercise. For example, a user may be granted the right to play music clips from a catalog for a cumulative period of one hour; the user may exercise this right by playing, say, twelve five-minute clips over a week, or over a month. The element `validityTimeMetered` has type `ValidityTimeMetered` , which is an extension of StatefulCondition. The means to connect to the state is made with the child element `stateReference`.

nteraction with the state

Upon exercise of a right conditioned by `validityTimeMetered`, beginning with the moment of exercise, at regular intervals of quantum `q` units of ime, the state designated by the `stateReference` is queried for its value. Suppose at interval `i` the state returns validFor containing a duration value d_i. Then, the state is updated to return $d_i - q$ hereon. This continues until the value of d_i is lesser than `q`, or when the interval of exercise of the right is complete.

The `validityTimeMetered` condition is said to be satisfied at interval `i` if and only if d_i is greater than `q`.

n the following example, Alice can exercise the right to read the book only when time has not run out on the designated state clock.

Example use of `validityTimeMetered`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <cx:print/>
    <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:validityTimeMetered>
    -<sx:stateReference licensePartId="stateReferenceAlias">
      -<uddi>
        -<serviceKey>
            <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
          </serviceKey>
        </uddi>
      </sx:stateReference>
      <sx:quantum>P1D</sx:quantum>
    </sx:validityTimeMetered>
  </grant>
```

5.1.9 The ValidityTimePeriodic Condition

ndicates 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".

Schema representation of `ValidityTimePeriodic`

```
-<xsd:complexType name="ValidityTimePeriodic">
  -<xsd:complexContent>
    -<xsd:extension base="r:Condition">
      -<xsd:sequence minOccurs="0">
        -<xsd:element name="start" type="xsd:dateTime">
          </xsd:element>
        -<xsd:element name="period" type="xsd:duration">
          </xsd:element>
        -<xsd:element name="phase" type="xsd:duration" minOccurs="0">
          </xsd:element>
        -<xsd:element name="duration" type="xsd:duration">
          </xsd:element>
        -<xsd:element name="periodCount" type="xsd:nonNegativeInteger" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `validityTimePeriodic`

```
-<xsd:element name="validityTimePeriodic" type="sx:ValidityTimePeriodic" substitutionGroup="r:condition">
  </xsd:element>
```

6.1.9.1 *ValidityTimePeriodic/start*

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

6.1.9.2 *ValidityTimePeriodic/period*

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

6.1.9.3 *ValidityTimePeriodic/phase*

A locally defined element of type `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.

6.1.9.4 *ValidityTimePeriodic/duration*

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

6.1.9.5 *ValidityTimePeriodic/periodCount*

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

Using the validityTimePeriodic element

Sometimes `grant`s may predicate rights to be exercisable on a periodic basis, such as "every weekend after Jan 1 2001 for a total of ten times". That is following some starting date, a time window of exercise periodically recurs. Such conditions are expressed using the `validityTimePeriodic` element. The element `validityTimePeriodic` has type `ValidityTimePeriodic`, which is an extension of Condition. The child element start marks the starting date for this condition to become relevant. The child element period indicates how often this time window should recur. The length of the duration is indicated by the element duration. An optional element phase is used to mark latency from the beginning of the period.

Formally, let `s` be the start time, `p` the period, `h` the phase, `d` the duration and `c` the count. Then the interval of exercise of the right must fall in one of the intervals described by the expression `{from:s + i*p + h, to: s + i*p + h + d}` where the value of `i` ranges from 0 to `c - 1` if `h` is positive and ranges from `1` to `c` if `h` is negative. If `c` is unspecified then `i` is unbounded.

In the following example, starting January 1st 2001, Alice may `print` the book only on the first two days of the second week of every month. Furthermore, this right lapses after twelve months.

Example of `validityTimePeriodic`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <cx:print/>
    <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:validityTimePeriodic>
      <sx:start>2001-01-01T00:00:00</sx:start>
      <sx:period>P1M</sx:period>
      <sx:phase>P7D</sx:phase>
      <sx:duration>P2D</sx:duration>
      <sx:periodCount>12</sx:periodCount>
    </sx:validityTimePeriodic>
  </grant>
```

The negative phase can be used to capture certain interesting periods. For example, on Memorial day, people who are veterans may watch the movie "Pearl Harbor" for free. Memorial day is the last Monday of May. So we combine three `validityTimePeriodic` conditions. The first one captures the last week of every month, the second one captures the month of May and the final one captures all Mondays beginning Jan 1 2001.

Example of `validityTimePeriodic`

```
-<grant>
  -<keyHolder licensePartIdRef="VeteransGroup">
    </keyHolder>
    <cx:play/>
    <cx:digitalWork licensePartIdRef="PearlHarbor"/>
  -<allConditions>
    -<sx:validityTimePeriodic>
```

```
        <sx:start>2001-01-01T00:00:00</sx:start>
        <sx:period>P1M</sx:period>
        <sx:phase>-P7D</sx:phase>
        <sx:duration>P7D</sx:duration>
      </sx:validityTimePeriodic>
     -<sx:validityTimePeriodic>
        <sx:start>2001-05-01T00:00:00</sx:start>
        <sx:period>P1Y</sx:period>
        <sx:duration>P1M</sx:duration>
      </sx:validityTimePeriodic>
     -<sx:validityTimePeriodic>
        <sx:start>2001-01-01T00:00:00</sx:start>
        <sx:period>P7D</sx:period>
        <sx:duration>P1D</sx:duration>
      </sx:validityTimePeriodic>
    </allConditions>
  </grant>
```

Astute readers will note that Memorial Day can actually be characterized without the usage of phase.

Example of `validityTimePeriodic`

```
-<grant>
  -<keyHolder licensePartIdRef="VeteransGroup">
    </keyHolder>
    <cx:play/>
    <cx:digitalWork licensePartIdRef="PearlHarbor"/>
  -<allConditions>
     -<sx:validityTimePeriodic>
        <sx:start>2001-05-25T00:00:00</sx:start>
        <sx:period>P1Y</sx:period>
        <sx:duration>P7D</sx:duration>
      </sx:validityTimePeriodic>
     -<sx:validityTimePeriodic>
        <sx:start>2001-01-01T00:00:00</sx:start>
        <sx:period>P7D</sx:period>
        <sx:duration>P1D</sx:duration>
      </sx:validityTimePeriodic>
    </allConditions>
  </grant>
```

.1.10 The Fee Condition

ndicates that a fee must be paid before a right is exercised.

Schema representation of `Fee`

```
-<xsd:complexType name="Fee">
  -<xsd:complexContent>
     -<xsd:extension base="r:Condition">
       -<xsd:sequence minOccurs="0">
          <xsd:element ref="sx:paymentAbstract"/>
         -<xsd:element name="min" minOccurs="0">
           -<xsd:complexType>
             -<xsd:sequence>
                <xsd:element ref="sx:paymentAbstract"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
         -<xsd:element name="max" minOccurs="0">
           -<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:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `fee`

```
-<xsd:element name="fee" type="sx:Fee" substitutionGroup="r:condition">
  </xsd:element>
```

.1.10.3 Fee/paymentAbstract

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

.1.10.2 Fee/min

This locally defined optional element contains a `paymentAbstract` element. It 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.

.1.10.3 Fee/max

This locally defined optional element contains a `paymentAbstract` element. It 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 value of max increases.

6.1.10.4 Fee/to

This locally defined element is of type of `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.

6.1.10.5 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 `any` element.

Schema representation of `AccountPayable`

```
 -<xsd:complexType name="AccountPayable">
   -<xsd:choice>
     -<xsd:element name="paymentService" type="r:ServiceReference">
       </xsd:element>
     -<xsd:element name="aba">
       -<xsd:complexType>
         -<xsd:sequence>
           -<xsd:element name="institution">
             -<xsd:simpleType>
               -<xsd:restriction base="xsd:integer">
                   <xsd:totalDigits value="9"/>
                 </xsd:restriction>
               </xsd:simpleType>
             </xsd:element>
           -<xsd:element name="account" type="xsd:integer">
             </xsd:element>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
     -<xsd:any namespace="##other" processContents="lax">
       </xsd:any>
     </xsd:choice>
   </xsd:complexType>
```

6.1.10.5.1 AccountPayable/paymentService

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

6.1.10.5.2 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.

Using the fee condition

A `grant` can predicate that a `fee` be paid before a right can be exercised. 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.

The `fee` element is of type `Fee`, which is an extension of Condition. A child element `to` of type `AccountPayable` is used to characterize both the payment 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 if the right is exercised at all. When both a maximum and minimum price specifications are given, the maximum price specification dominates.

Suppose that the `fee`s in payment, `min` and `max` independently amount to `p`, `min` and `max` respectively due for the exercise of the associated right as defined by the respective payment extension. Then, `x` is `min` if `p <= min < max`; `x` is `p` if `min < p< max`; `x` is `max` if `min < max < p`. Then the `fee` condition is satisfied if and only if the amount `x` is paid to the entity identified by `to`

Not all forms of payment extensions are directly comparable. For the `fee` condition to be sensibly evaluated, it is necessary that the payment extensions of the `paymentAbstract` elements in payment, min and max are comparable. In the standard extension, the payment extensions either contain state references or they do not. Stateful payment extensions are not comparable with stateless ones. So it is required that the payment extensions in payment, min and max are either all stateful or all stateless. The corresponding `fee` conditions are respectively termed as stateful and stateless.

6.1.11 The Territory Condition

Indicates a geographic or virtual space within which the associated right may be exercised.

Schema representation of `Territory`

```
 -<xsd:complexType name="Territory">
   -<xsd:complexContent>
     -<xsd:extension base="r:Condition">
```

```
      -<xsd:choice minOccurs="0" maxOccurs="unbounded">
        -<xsd:element name="location">
          -<xsd:complexType>
            -<xsd:sequence>
              -<xsd:element name="region" type="sx:RegionCode" minOccurs="0">
               </xsd:element>
              -<xsd:element name="country" type="sx:CountryCode" minOccurs="0">
               </xsd:element>
              -<xsd:element name="state" type="xsd:string" minOccurs="0">
               </xsd:element>
              -<xsd:element name="city" type="xsd:string" minOccurs="0">
               </xsd:element>
              -<xsd:element name="postalCode" type="xsd:string" minOccurs="0">
               </xsd:element>
              -<xsd:element name="street" type="xsd:string" minOccurs="0">
               </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        -<xsd:element name="domain">
          -<xsd:complexType>
            -<xsd:sequence>
              -<xsd:element name="url" type="xsd:anyURI">
               </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `territory`

```
-<xsd:element name="territory" type="sx:Territory" substitutionGroup="r:condition">
  </xsd:element>
```

5.1.11.1 Territory/location

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

5.1.11.1.1 Territory/location/region

The locally defined element region, as defined by its type `RegionCode`, is a three-letter ISO 3166 region code.

5.1.11.1.2 Territory/location/country

The locally defined element country, as defined by its type `CountryCode`, is a two-letter ISO 3166 country code.

5.1.11.1.3 Territory/location/state

The locally defined element state is a two-letter code for US states.

5.1.11.1.4 Territory/location/city

The locally defined element is of type `string` and designates a city.

5.1.11.1.5 Territory/location/postalCode

The locally defined element `postalCode` is of type `string` and designates a postal code (e.g. a zip code).

5.1.11.1.6 Territory/location/street

The locally defined element street is of type `string` and designates a street.

5.1.11.2 Territory/domain

The locally defined element defines inline an element `url` to designate a digital location.

5.1.11.2.1 Territory/domain/url

The locally defined element `url`, has type `anyUri` and can be any uri.

Using the territory condition

Grants may sometimes predicate rights to be exercisable only in certain locations. These locations may correspond to physical or geographical regions, or they may correspond to virtual or digital locations. The element `territory` is used to specify such conditions. `territory` has type `Territory`, which is an extension of `Condition`. The child element location is used to indicate physical locations, and the child element domain indicates the digital locations. Since the right may be exercisable in more than one location, physical or digital, the content model for territory allows for an unbounded number of children.

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

In the following example, Alice may `print` the book only if she is in the USA or from a device in the www.xrml.org domain.

Example of the `territory` condition

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:territory>
    -<sx:location>
       <sx:country>US</sx:country>
     </sx:location>
    -<sx:domain>
       <sx:url> www.xrml.org </sx:url>
     </sx:domain>
   </sx:territory>
 </grant>
```

5.1.12 State Interaction Elements

The following elements are used by various elements of type `StatefulCondition` (or derivations thereof) to describe their interaction with the state. Specifically, the state exchanges data using these elements.

5.1.12.1 approval

This element of type `boolean` is used by `seekApproval`.

5.1.12.2 count

This element of type `integer` is used by `ExerciseLimit`, `trackQuery` and `paymentPerUse`.

5.1.12.3 paid

This element of type `boolean` is used by `paymentFlat`.

5.1.12.4 validFor

This element of type `duration` is used by `validityIntervalFloating` and `validityTimeMetered`.

5.1.12.5 validUntil

This element of type `dateTime` is used by `validityIntervalFloating` and `paymentPerInterval`.

5.2 Payment and its Extensions

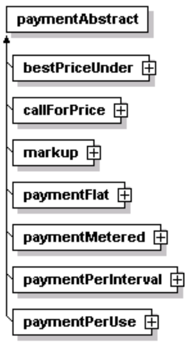
5.2.1 paymentAbstract

The head of a substitution group chain for the PaymentAbstract type.

The notion of payment is left abstract in XRML2.0. The element `paymentAbstract` has type `PaymentAbstract`. The type `PaymentAbstract` does not define any specific sub-elements.

In its place more concrete forms of payment such as `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 work toward the `fee` condition being satisfied. Since payment extensions themselves are not conditions, we introduce a notion of fulfillment. For a `fee` condition to be satisfied, the payment extension must be fulfilled. Each payment extension describes the semantic of its fulfillment.

Payment Extensions



Schema representation of `PaymentAbstract`

```
-<xsd:complexType name="PaymentAbstract" abstract="true">
 </xsd:complexType>
```

Schema representation of `paymentAbstract`

```
 -<xsd:element name="paymentAbstract" type="sx:PaymentAbstract">
  </xsd:element>
```

6.2.2 Cash

A fixed amount of money in a designated currency.

Schema representation of Cash

```
 -<xsd:complexType name="Cash">
   -<xsd:simpleContent>
     -<xsd:extension base="xsd:decimal">
        <xsd:attribute name="currency" type="sx:CurrencyCode" default="USD"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
```

6.2.3 PaymentFlat

Specifies a payment due upon exercising a right when the value in paymentRecord is False. When this fee is paid, the payment record should be updated to True.

Schema representation of PaymentFlat

```
 -<xsd:complexType name="PaymentFlat">
   -<xsd:complexContent>
     -<xsd:extension base="sx:PaymentAbstract">
       -<xsd:sequence>
         -<xsd:element name="rate" type="sx:Cash">
          </xsd:element>
         -<xsd:element name="paymentRecord">
           -<xsd:complexType>
             -<xsd:sequence>
                <xsd:element ref="sx:stateReference"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of paymentFlat

```
 -<xsd:element name="paymentFlat" type="sx:PaymentFlat" substitutionGroup="sx:paymentAbstract">
  </xsd:element>
```

6.2.3.1 PaymentFlat/rate

This locally defined element of type `Cash` indicates the amount to be paid to exercise the right.

6.2.3.2 PaymentFlat/paymentRecord

This locally defined element contains a `stateReference` that holds a value. This value is true if payment has been made, and false otherwise.

Using the paymentFlat element

Some rights may be exercised upon the payment of a one-time `fee`. For example, purchasing a song. The `paymentFlat` element of type `PaymentFlat` is used to describe such payments. A local element rate is used to indicate the amount to be paid and another element `paymentRecord` is used to designate a state which holds the information whether the payment has been made.

Interaction with the state

When a right conditioned upon a `paymentFlat` `fee` is exercised, the state designated by `PaymentFlat`/ `paymentRecord`/`stateReference` is queried for its value. The state returns `paid` holding a `boolean` value `b`. When a payment of amount as designated by `rate` is made, the state is updated to return `paid` set to true.

The `paymentFlat` payment is fulfilled if `b` is true.

In the following example, Alice may listen to "La Bamba" provided she makes a one-time payment of $5.

Example of paymentFlat

```
 -<grant>
   -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <cx:play/>
    <cx:digitalWork licensePartIdRef="LaBamba"/>
   -<sx:fee>
     -<sx:paymentFlat>
        <sx:rate> 5 </sx:rate>
       -<sx:paymentRecord>
         -<sx:stateReference licensePartId="stateReferenceAlias">
           -<uddi>
             -<serviceKey>
                <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
```

```
          </serviceKey>
        </uddi>
      </sx:stateReference>
    </sx:paymentRecord>
  </sx:paymentFlat>
 -<sx:to>
   -<sx:aba>
      <sx:institution>123456789</sx:institution>
      <sx:account>987654321</sx:account>
    </sx:aba>
  </sx:to>
 </sx:fee>
</grant>
```

6.2.4 PaymentMetered

Specifies a payment due for each time interval during which the right is actually exercised.

Schema representation of `PaymentMetered`

```
-<xsd:complexType name="PaymentMetered">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence>
        -<xsd:element name="rate" type="sx:Cash">
          </xsd:element>
        -<xsd:element name="per" type="xsd:duration">
          </xsd:element>
        -<xsd:element name="by" type="xsd:duration">
          </xsd:element>
        -<xsd:element name="phase" type="xsd:duration">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `paymentMetered`

```
-<xsd:element name="paymentMetered" type="sx:PaymentMetered" substitutionGroup="sx:paymentAbstract">
  </xsd:element>
```

6.2.4.1 PaymentMetered/rate

This locally defined element of type `Cash`, together with the element `per`, indicates the charge per period.

6.2.4.2 PaymentMetered/per

This locally defined element of type `duration` indicates the period available for the payment of the `rate`.

6.2.4.3 PaymentMetered/by

This locally defined element of type `duration` indicates the quantum by which time is measured for the computation of the amount.

6.2.4.4 PaymentMetered/phase

This locally defined element of type `duration` is used for rounding purposes. It indicates what portion of time may elapse before someone is billed for the whole duration defined with `by`.

Using the paymentMetered element

Some rights may be exercised upon the payment of a `fee` that is prorated according to the duration of usage. For example, one may have the righ to play a game and pay a `fee` according dictated by the length of time one plays the game. The `paymentMetered` element of type `PaymentMetered` is used to describe such payments. A local element `rate`, with another element `per`, is used to designate the basic cost. Another element `by` is used to describe the granularity with which time is measured. Finally, 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, and a value of the duration described in the `by` element would have the effect of rounding down.

Formally, after normalizing 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*[b/p]*[floor(t/b) + round]` where `roundt%b` is greater than `h` and is zero otherwise.

n the following example, Alice may play a game but is charged a daily rate of 24$/day. This rate is computed on an hourly basis. Furthermore, any usage of over thirty minutes is docked as an hour. So if Alice spends two hours and twenty-nine minutes playing, then she would end up paying $2 n all ($1 after 30 minutes and another dollar after 90 minutes). However, if she spends two hours and thirty-one minutes playing she would end up paying $3 in all (yet another dollar after 150 minutes).

Example of `PaymentMetered`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <cx:play/>
    <cx:digitalWork licensePartIdRef="AlicesGame"/>
  -<sx:fee>
    -<sx:paymentMetered>
       <sx:rate> 24 </sx:rate>
```

```
        <sx:per> P1D </sx:per>
        <sx:by> PT1H </sx:by>
        <sx:phase> PT30M </sx:phase>
      </sx:paymentMetered>
    -<sx:to>
      -<sx:aba>
          <sx:institution>123456789</sx:institution>
          <sx:account>987654321</sx:account>
        </sx:aba>
      </sx:to>
    </sx:fee>
  </grant>
```

5.2.5 PaymentPerInterval

Specifies a payment due for each time interval during which the ability to exercise the right is desired.

Schema representation of `PaymentPerInterval`

```
-<xsd:complexType name="PaymentPerInterval">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence>
        -<xsd:element name="rate" type="sx:Cash">
          </xsd:element>
        -<xsd:element name="per" type="xsd:duration">
          </xsd:element>
        -<xsd:element name="paidThrough">
          -<xsd:complexType>
            -<xsd:sequence>
                <xsd:element ref="sx:stateReference"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of `paymentPerInterval`

```
-<xsd:element name="paymentPerInterval" type="sx:PaymentPerInterval" substitutionGroup="sx:paymentAbstract">
  </xsd:element>
```

5.2.5.1 PaymentPerInterval/rate

This locally defined element of type `Cash`, indicates the amount to paid for the allocation of an interval of time defined by `per`.

5.2.5.2 PaymentPerInterval/per

This locally defined element of type `duration`, indicates the quantum of time allocated with the payment of the amount in `rate`.

5.2.5.3 PaymentPerInterval/paidThrough

This locally defined element contains a `stateReference` whose value is the time through which the amount is paid.

Using the paymentPerInterval element

Some rights may be exercised for a period of time that they are paid for. For example, one may buy up some time to play a game and may keep playing the game until it is paid for. The `paymentPerInterval` element of type `PaymentPerInterval` is used to describe such payments. A local element `rate`, with another element `per`, is used to indicate the cost for a certain duration of time. Another element `paidThrough` is used to record the time until when the right may be exercisable.

Interaction with the state

When a right conditioned upon a `paymentPerInterval` `fee` is exercised, the state designated by `paymentPerInterval`/ `paidThrough`/ `stateReference` is queried for its value. The state returns `validUntil` that contains a `dateTime` value `t`. This marks the time until when the right may be exercised. For every payment currently made in the amount in `rate`, if `t` is in the future then it is increased by the duration designated in the `per` element. Otherwise, the value is set to a time that is `per` duration from the global official time (colloquially, the present time). The state is then updated to return `validUntil` containing the new value of `t`.

The `paymentPerInterval` payment is fulfilled if the interval of exercise of the associated right is bounded by `t`.

In the following example, Alice may `play` her game and is charged a rate of $5/day.

Example of `paymentPerInterval`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
    </keyHolder>
    <cx:play/>
    <cx:digitalWork licensePartIdRef="AlicesGame"/>
  -<sx:fee>
    -<sx:paymentPerInterval>
        <sx:rate> 5 </sx:rate>
        <sx:per> P1D </sx:per>
      -<sx:paidThrough>
      -<sx:stateReference licensePartId="stateReferenceAlias">
        -<uddi>
          -<serviceKey>
              <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
            </serviceKey>
          </uddi>
        </sx:stateReference>
      </sx:paidThrough>
    </sx:paymentPerInterval>
   -<sx:to>
     -<sx:aba>
        <sx:institution>123456789</sx:institution>
        <sx:account>987654321</sx:account>
      </sx:aba>
    </sx:to>
  </sx:fee>
</grant>
```

5.2.6 PaymentPerUse

Specifies a payment due each time a right is exercised.

Schema representation of PaymentPerUse

```
-<xsd:complexType name="PaymentPerUse">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence>
        -<xsd:element name="rate" type="sx:Cash">
          </xsd:element>
        -<xsd:element name="allowPrePay" minOccurs="0">
          -<xsd:complexType>
            -<xsd:sequence>
              -<xsd:element name="initialNumberOfUses" type="xsd:integer" minOccurs="0">
                </xsd:element>
              -<xsd:element name="prePaidUsesRemaining">
                -<xsd:complexType>
                  -<xsd:sequence>
                      <xsd:element ref="sx:stateReference"/>
                    </xsd:sequence>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

Schema representation of paymentPerUse

```
-<xsd:element name="paymentPerUse" type="sx:PaymentPerUse" substitutionGroup="sx:paymentAbstract">
  </xsd:element>
```

5.2.6.1 paymentPerUse/rate

This locally defined element of type `Cash` indicates the amount to be paid for a certain number of uses, if specified by the element `initialNumberOfUses`, otherwise it defines the amount to be paid for each use.

5.2.6.2 paymentPerUse/allowPrePay

This locally defined optional element is used to allow for prepayments.

5.2.6.2.1 paymentPerUse/allowPrePay/initialNumberOfUses

This optional element of type `integer` indicates the number of uses each rate payment buys. If absent, the number of uses is one.

5.2.6.2.2 paymentPerUse/allowPrePay/prepaidUsesRemaining

This locally defined element contains a `stateReference` that indicates the remaining number of uses.

Using the paymentPerUse

Some rights may be exercised upon payment of a `fee` for each use. For example, one may have the right to listen to a piece of music provided a payment is made for each listening. The element `paymentPerUse` of type `PaymentPerUse` is used to describe such payments.

Interaction with the state

When a right conditioned upon a `paymentPerUse` `fee` is exercised, the state designated by `paymentPerUse/allowPrePay/prepaidUsesRemaining` is queried for its value. The state returns `count` containing an `integer` value `c`. Upon exercise of the right, the value of the state is updated to return `count` containing `c - 1` hereon. When payment in the amount defined in the `rate` element is made, then state is updated to increment `count` by the `initialNumberOfUses` value.

The `paymentPerUse` payment is fulfilled if the value of `c` is greater than zero and the state is updated as described above.

In the following example Alice may `print` the book five times provided she pays $5.

Example of `paymentPerUse`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:fee>
    -<sx:paymentPerUse>
       <sx:rate> 5 </sx:rate>
      -<sx:allowPrePay>
         <sx:initialNumberOfUses> 5 </sx:initialNumberOfUses>
        -<sx:prePaidUsesRemaining>
          -<sx:stateReference licensePartId="stateReferenceAlias">
            -<uddi>
              -<serviceKey>
                 <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
               </serviceKey>
             </uddi>
           </sx:stateReference>
         </sx:prePaidUsesRemaining>
       </sx:allowPrePay>
     </sx:paymentPerUse>
    -<sx:to>
      -<sx:aba>
         <sx:institution>123456789</sx:institution>
         <sx:account>987654321</sx:account>
       </sx:aba>
     </sx:to>
   </sx:fee>
 </grant>
```

6.2.7 BestPriceUnder

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.

Schema representation of `BestPriceUnder`

```
-<xsd:complexType name="BestPriceUnder">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence>
         <xsd:element ref="sx:paymentAbstract"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `bestPriceUnder`

```
-<xsd:element name="bestPriceUnder" type="sx:BestPriceUnder" substitutionGroup="sx:paymentAbstract">
 </xsd:element>
```

Using the bestPriceUnder element

`bestPriceUnder` is a kind of payment that can be dynamic and is determined when the account is settled. It is used to accommodate special deals, rebates, and pricing that depends on information that is not available to the trusted repository at the time the usage right is exercised, but without communicating with a dealer before the purchase is authorized. A `bestPriceUnder` specification limits the risk to the user by naming a maximum amount that the exercising of the right will cost. This is the amount that is tentatively charged to the account. However, when the transaction is ultimately reconciled, any excess amount charged will be returned to the user/copy-owner in a separate transaction.

The `bestPriceUnder` element has type `BestPriceUnder`, which is an extension of `PaymentAbstract`. It has a child element `fee` which is used to indicate the charge. The `bestPriceUnder` element is defined to be in payment's substitution group, and can be used wherever payment is expected.

In the following example, Alice may `print` the book and will be charged at most $5 to do it.

Example of `bestPriceUnder`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:fee>
    -<sx:bestPriceUnder>
      -<sx:paymentFlat>
         <sx:rate> 5 </sx:rate>
        -<sx:paymentRecord>
          -<sx:stateReference>
            -<uddi>
              -<serviceKey>
                 <uuid>D04951E4-332C-4693-B7DB-D3D1D1C20844</uuid>
               </serviceKey>
             </uddi>
           </sx:stateReference>
         </sx:paymentRecord>
       </sx:paymentFlat>
     </sx:bestPriceUnder>
    -<sx:to>
      -<sx:aba>
         <sx:institution>123456789</sx:institution>
         <sx:account>987654321</sx:account>
       </sx:aba>
     </sx:to>
   </sx:fee>
 </grant>
```

6.2.8 CallForPrice

dentifies an entity with whom a price must be negotiated before exercising the right.

Schema representation of `CallForPrice`

```
-<xsd:complexType name="CallForPrice">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence minOccurs="0">
        -<xsd:element name="location" type="r:ServiceReference" maxOccurs="unbounded">
         </xsd:element>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `callForPrice`

```
-<xsd:element name="callForPrice" type="sx:CallForPrice" substitutionGroup="sx:paymentAbstract">
 </xsd:element>
```

6.2.8.1 CallForPrice/location

This optional element of type `ServiceReference` indicates the location where the negotiation may be supported. This may simply be a matter of ocating a dealer.

Using the callForPrice element

callForPrice is similar to `bestPriceUnder` in that it is intended to accommodate cases where prices are dynamic. However, unlike `bestPriceUnder`, communication with a dealer to determine the price is required before the purchase is authorized; the transaction cannot be completed if the trusted repository is unable to communicate with the dealer.

`callForPrice`is of type `CallForPrice`, which is an extension of `PaymentAbstract`. It can contain possibly several location child elements. The ocation elements are used to communicate with the dealers.

Example of `callForPrice`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:print/>
   <cx:digitalWork licensePartIdRef="Book"/>
  -<sx:fee>
    -<sx:callForPrice>
      -<sx:location licensePartId="LocationOne">
        -<uddi>
          -<serviceKey>
             <uuid>EE4A90A5-8AC9-4f31-85F7-6619AA573449</uuid>
           </serviceKey>
         </uddi>
       </sx:location>
      -<sx:location licensePartId="LocationTwo">
        -<uddi>
          -<serviceKey>
             <uuid>EE4A90A5-8AC9-4f31-85F7-6619AA5734BA</uuid>
           </serviceKey>
         </uddi>
       </sx:location>
     </sx:callForPrice>
   </sx:fee>
 </grant>
```

6.2.9 Markup

Specifies a fee due each time some other fees are due.

Schema representation of `Markup`

```
-<xsd:complexType name="Markup">
  -<xsd:complexContent>
    -<xsd:extension base="sx:PaymentAbstract">
      -<xsd:sequence>
        -<xsd:element name="rate" type="xsd:float">
         </xsd:element>
        -<xsd:choice maxOccurs="unbounded">
           <xsd:element ref="sx:fee"/>
          -<xsd:element name="feeForResource">
            -<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>
```

Schema representation of `markup`

```
-<xsd:element name="markup" type="sx:Markup" substitutionGroup="sx:paymentAbstract">
 </xsd:element>
```

.2.9.1 rate

This locally defined element is of type float. It indicates the fractional rate at which `markup` is calculated.

.2.9.2 feeForResource

mplies a context in which multiple resources are being used simultaneously. The total price for using the specified resource is calculated, and the amount is paid as defined by any license agreements for that resource. The price is then marked up by the rate specified in this `markup` element, and the markup is paid as specified by the containing `fee` element. If the specified resource is not used in conjunction with exercising this `grant`, this markup is not fulfilled and the containing `fee` is not satisfied.

Using the markup element

`Markup` `fee`s are `fee`s that are computed as a percentage of other `fee`s. For example, a distributor may want to add a flat ten percent overhead for selling copies of a digital work, or a government may want to tax sales of a digital works.

The following example shows a `license` granted to Alice (by some distributor) that allows Alice to issue a `grant` conditioned upon the payment of a marked-up `fee`. This allows Alice the flexibility of setting her own `fee` when issuing the `grant`. The distributor pockets the `markup` of 10%.

Example of `markup`

```
-<grant>
  -<forAll varName="preMarkupFee">
   </forAll>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <issue/>
  -<grant>
    -<keyHolder licensePartIdRef="bookClubMember">
     </keyHolder>
     <cx:play/>
     <cx:digitalWork licensePartIdRef="Book"/>
    -<sx:fee>
      -<sx:markup>
         <sx:rate>.1</sx:rate>
         <sx:fee varRef="preMarkupFee"/>
       </sx:markup>
      -<sx:to>
        -<sx:aba>
           <sx:institution>123456789</sx:institution>
           <sx:account>987654321</sx:account>
         </sx:aba>
       </sx:to>
     </sx:fee>
   </grant>
 </grant>
```

The following example shows two `license`s that may be exercised by a streaming media consumer. The first one is a `grant` to actually play the content for some `fee`, and the second is a `grant` to download the content from some repository for a markup of the `fee` indicated in the first.

Example of `markup` with `feeForResource`

```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:play/>
   <cx:digitalWork licensePartIdRef="AClockworkOrange"/>
  -<sx:fee>
    -<sx:paymentPerUse>
       <sx:rate>5</sx:rate>
     </sx:paymentPerUse>
    -<sx:to>
      -<sx:aba>
         <sx:institution>123456789</sx:institution>
         <sx:account>987654321</sx:account>
       </sx:aba>
     </sx:to>
   </sx:fee>
 </grant>
```

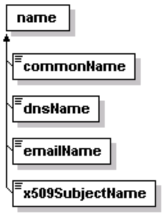
```
-<grant>
  -<keyHolder licensePartIdRef="Alice">
   </keyHolder>
   <cx:read/>
   <cx:digitalWork licensePartIdRef="AClockworkOrange"/>
  -<sx:fee>
    -<sx:markup>
        <sx:rate>0.1</sx:rate>
      -<sx:feeForResource>
          <cx:digitalWork licensePartIdRef="AClockworkOrange"/>
        </sx:feeForResource>
      </sx:markup>
    -<sx:to>
      -<sx:aba>
          <sx:institution>123456789</sx:institution>
          <sx:account>987654321</sx:account>
        </sx:aba>
      </sx:to>
    </sx:fee>
  </grant>
```

6.3 Name Extensions

Together with the `possessProperty`, the resource `Name` and its extensions allow `license`s to straightforwardly express authorized association of names with principals. This is useful for modeling the X.509 certificate like binding of names to principals. The standard extension defines four extensions of the type `Name`.

6.3.1 Name

Name Extensions



A resource indicating a name from some name space.

Schema representation of `Name`

```
 -<xsd:complexType name="Name" abstract="false">
   -<xsd:complexContent>
       <xsd:extension base="r:Resource"/>
     </xsd:complexContent>
   </xsd:complexType>
```

Schema representation of `name`

```
 -<xsd:element name="name" type="sx:Name" substitutionGroup="r:resource">
   </xsd:element>
```

Using the name element and its substitutions

The `name` element is of type `Name`. This is an extension of the abstract type `Resource`. Along with the `possessProperty` right, the `name` element being a `resource` can be used in `license`s to associate a name with one or more `principal`s. Such `license`s allow other `license`s to be issued to, colloquially speaking, `principal`s identified by their name.

The `name` element is conceptually abstract and should not appear in an XrML document except in the form of a variable reference.

The following example shows a `license` which allows anyone who has the name Alice, as authorized by the `grant` in the prerequisite right, can watch "A Clockwork Orange". This example illustrates how an extension of name is used; note the use of `commonName` to identify the authorized `principal`.

Example of `name` extension

```
 -<grant>
   -<forAll varName="personX">
     </forAll>
     <principal varRef="personX"/>
     <cx:play/>
     <cx:digitalWork licensePartIdRef="AClockworkOrange"/>
   -<prerequisiteRight>
       <principal varRef="personX"/>
       <possessProperty/>
       <sx:commonName>Alice</sx:commonName>
     -<trustedIssuer>
       +<keyHolder licensePartId="trustedIssuer">
       </trustedIssuer>
     </prerequisiteRight>
   </grant>
```

6.3.2 EmailName

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

Schema representation of `EmailName`

```
-<xsd:complexType name="EmailName" mixed="true">
  -<xsd:complexContent mixed="true">
     <xsd:extension base="sx:Name"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `emailName`

```
-<xsd:element name="emailName" type="sx:EmailName" substitutionGroup="sx:name">
 </xsd:element>
```

Using the emailName element

The `emailName` element is of type `EmailName`. This is an extension of the type `Name`. Typically it contains a string that designates an internet email address (per rfc822/2822). Like with `name`, `license`s can associate the element `emailName` with `principal`s by using the right `possessProperty`.

5.3.3 DnsName

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

Schema representation of `DnsName`

```
-<xsd:complexType name="DnsName" mixed="true">
  -<xsd:complexContent mixed="true">
     <xsd:extension base="sx:Name"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `dnsName`

```
-<xsd:element name="dnsName" type="sx:DnsName" substitutionGroup="sx:name">
 </xsd:element>
```

Using the dnsName element

The `dnsName` element is of type `DnsName`. This is an extension of the type `Name`. Typically it contains a string that designates a domain name. Like with `name`, `license`s can associate the element `dnsName` with `principal`s by using the right `possessProperty`.

5.3.4 CommonName

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

Schema representation of `CommonName`

```
-<xsd:complexType name="CommonName" mixed="true">
  -<xsd:complexContent mixed="true">
     <xsd:extension base="sx:Name"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `commonName`

```
-<xsd:element name="commonName" type="sx:CommonName" substitutionGroup="sx:name">
 </xsd:element>
```

Using the commonName element

The `commonName` element is of type `CommonName` . This is an extension of the type `Name`. Typically it contains a string that designates a colloquial name. Like with `name`, `license`s can associate the element `commonName` with `principal`s by using the right `possessProperty`.

5.3.5 X509SubjectName

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

Schema representation of `X509SubjectName`

```
-<xsd:complexType name="X509SubjectName" mixed="true">
  -<xsd:complexContent mixed="true">
     <xsd:extension base="sx:Name"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `x509SubjectName`

```
-<xsd:element name="x509SubjectName" type="sx:X509SubjectName" substitutionGroup="sx:name">
 </xsd:element>
```

Using the x509SubjectName element

The `x509SubjectName` element is of type `X509SubjectName`. This is an extension of the type `Name`. Typically it contains a string that designates a subject name from some X509 certificate. Like with `name`, `license`s can associate the element `x509SubjectName` with `principal`s by using the right `possessProperty`.

6.3.6 X509SubjectNamePattern

Schema representation of `X509SubjectNamePattern`

```
-<xsd:complexType name="X509SubjectNamePattern">
  -<xsd:complexContent>
      <xsd:extension base="r:ResourcePatternAbstract"/>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `x509SubjectNamePattern`

```
-<xsd:element name="x509SubjectNamePattern" type="sx:X509SubjectNamePattern" substitutionGroup="r:resourcePatternAbstract">
 </xsd:element>
```

Using the x509SubjectNamePattern

The `x509SubjectNamePattern` is of type `X509SubjectNamePattern`. This is an extension of the type `ResourcePattern`. The `x509SubjectNamePattern` is pattern that matches an `x509SubjectName`. Specifically it matches the root of the `x509SubjectName` tree. This element can be used to enforce constraints similar to the X.509 specification.

6.4 Revocation Extensions

6.4.1 Revocable

Identifies a signature that can be revoked. The signature can be identified literally or by reference. In the latter case, the result of dereferencing the reference must be of type dsig:SignatureType; the signature value being revoked is the one signature therein.

Schema representation of `Revocable`

```
-<xsd:complexType name="Revocable">
  -<xsd:complexContent>
    -<xsd:extension base="r:Resource">
      -<xsd:choice minOccurs="0">
          <xsd:element ref="dsig:SignatureValue"/>
          <xsd:element ref="dsig:Reference"/>
       </xsd:choice>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

Schema representation of `revocable`

```
-<xsd:element name="revocable" type="sx:Revocable" substitutionGroup="r:resource">
 </xsd:element>
```

Using the revocable element

The `revocable` element of type `Revocable`, based on `Resource`, is used with the `revoke` right. Typically, to revoke a signature, a `license` is issued which identifies the `principal` that has the right to revoke the signature specified by the `revocable` element. The `revocable` element identifies the signature either by its literal value or by indirect means such as its cryptographic hash value.

The following example illustrates how Alice has the right to `revoke` a specific signature. Note that the signature is referred to by its cryptographic hash value.

Example of `revocable`

```
-<grant>
  -<keyHolder licensePartId="Alice">
   </keyHolder>
   <revoke/>
  -<sx:revocable>
    -<dsig:Reference>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <dsig:DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</dsig:DigestValue>
     </dsig:Reference>
   </sx:revocable>
  -<validityInterval>
     <notBefore>2001-05-25T00:00:00</notBefore>
     <notAfter>2003-05-25T00:00:00</notAfter>
   </validityInterval>
 </grant>
```

Go to Part IV: Content Extension Schema