Driving network video through global standardization

ONVIF XML Schema Version and Extension Handling

White Paper

Author: Christian Gehrmann, Axis Communications AB

1 Background

1.1 Purpose

Version *and* extensions handling for XML schemas are *rather complex issues* and unfortunately there is no easy path to take when writing standards based on XML schemas. One needs to make a trade-off between backward and forward compatibility requirements and complexity of the schema. The main problem comes from the XML determinism rule, which we briefly explain below. A rather good overview of the issues and potential strategies are given in [1].

In this white paper we analyze the XML schema version and extension handling problem and give recommendations for versions and extensions of ONVIF defined schemas.

1.2 XML determinism rule

Defining extensions and new versions in a compatible way would have been much easier if the XML 1.0 specification [2] did not define the following:

- "... it is required that content models in element type declarations be deterministic. This requirement is for compatibility with SGML (which calls deterministic content models "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors."

The determinism rule or the so called "Unique Particle Attribution Constraint" [3] means that at each point, when matching an instance document against a schema, the schema processor has at most one possible choice. That implies for instance that a schema declaration with wildcard "##any" where minOccurs does not equal maxOccurs not is allowed before an element declaration. Similar an element declaration where minOccurs does not equal maxOccurs before a wildcard ##any declaration is not either allowed.

2 Extension and versioning requirements and limitations

The XML schema limitations can be avoided by using involved schema specifying rules as analyzed in [1]. However, this results in schemas that are very complex and without clean structure. Hence, we assume limitations on the backward and forward compatibility requirements on the ONVIF schema:

- Each version of the schema shall allow for proprietary extensions elements for relevant data

types such that this version will work with all future ONVIF versions in the same namespace.

- We *do not* require that schema Vn with proprietary extensions will interoperate with schema Vn-x (x = any pos. integer) with proprietary extensions from the same or different vendor or with schema Vn+y (y = any pos. integer) with proprietary extensions from the same or different vendor in all cases (see proprietary extension option 3 below).
- We *do not* require that schema Vn with proprietary extensions from vendor A will interoperate with schema Vn from Vendor B in all cases (see proprietary extension option 3 below).
- Each new ONVIF version of the schema, Vn, shall be backward and forward compatible with all other ONVIF versions in the same namespace, i.e., version Vn-x and Vn+y.

3 Recommended version one schema definition principles

The following extension handling principle will be used for the ONVIF 1.0 schema:

- Each complex type declaration in the current preliminary schema is evaluated for inclusion of extension points. If we have consensus that allowing extensions of the element (either proprietary or in future version of the specification) the following apply (otherwise the element is kept without extension points):
    - If in the last element in the sequence, minOccurs = maxOccurs the following extension declaration is added: <xs:any namespace="##any" procesContents="lax" minOccurs= "0" maxOccurs="unbounded". The <xs:anyAttribute/> is also added after the sequence declaration.
    - If in the last element in the sequence, minOccurs *does not equals* maxOccurs, an element with the name "Extension" and of a type unique type with minOccurs="0" and maxOccurs="1" is added at the end of the sequence. The extension element type then needs to be declared as namespace unique type and it shall be a complex type with a sequence of elements declared as an element with namespace="##any", processContents="lax" and with minOccurs="0" and maxOccus="unbounded". The <xs:anyAttribute/> shall be added after the sequence declaration in the base element type declaration.

Proprietary extensions for the first version of the standard shall then be handled as follows:

1) ONVIF strongly recommends adding proprietary extensions through defining new web services commands when suitable and *not through extensions* of *the* existing schema elements. This option allows full backward and forward interoperability with extensions form different vendors.
2) If new commands not are the best way to add proprietary extensions, ONVIF recommends vendors whenever possible to declare proprietary element extensions as *attribute declarations* instead of adding new schema elements in existing types. New proprietary attributes *shall be declared* in a namespace *different from the target namespace.* This option allows full backward and forward interoperability.
3) If it is not possible to declare the proprietary extensions using new commands or attributes, the proprietary extensions *shall be declared* adding a *type unique mandatory extension element before the extension point* and then add all new proprietary element in this extensions element. The new elements shall be declared in a namespace *different from the target namespace*. Each proprietary extension element *must have* minOccus="0" and the proprietary extension element declarations in the schema *must be followed* by an xs:any element declaration in the target namespace with minOccurs="0" and maxOccurs="unbounded".

We illustrate the principles above with some example. The examples utilize two different ONVIF types from the *preliminary* schema (without extension points), called type 1 and type 2.

Type 1:

```
<xs:complexType name="SecurityCapabilities">
  <xs:sequence>
    <xs:element name="TLS1.1" type="xs:boolean"/>
    <xs:element name="TLS1.2" type="xs:boolean"/>
    <xs:element name="OnboardKeyGeneration" type="xs:boolean"/>
    <xs:element name="AccessPolicyConfig" type="xs:boolean"/>
    <xs:element name="X.509Token" type="xs:boolean"/>
    <xs:element name="SAMLToken" type="xs:boolean"/>
    <xs:element name="KerberosToken" type="xs:boolean"/>
    <xs:element name="RELToken" type="xs:boolean"/>
  </xs:sequence>
```

```
</xs:complexType>
```

Type 2:

```
<xs:complexType name="SystemCapabilities">
  <xs:sequence>
    <xs:element name="DiscoveryResolve" type="xs:boolean"/>
    <xs:element name="DiscoveryBye" type="xs:boolean"/>
    <xs:element name="RemoteDiscovery" type="xs:boolean"/>
    <xs:element name="SystemBackup" type="xs:boolean"/>
    <xs:element name="SystemLogging" type="xs:boolean"/>
    <xs:element name="FirmwareUpgrade" type="xs:boolean"/>
    <xs:element name="SupportedVersions" type="tt:OnvifVersion"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

3.1 Example 1, extension declaration after minOccurs = maxOccurs element

Type 1 is extended in the following way:

```
<xs:complexType name="SecurityCapabilities">
  <xs:sequence>
    <xs:element name="TLS1.1" type="xs:boolean"/>
    <xs:element name="TLS1.2" type="xs:boolean"/>
    <xs:element name="OnboardKeyGeneration" type="xs:boolean"/>
    <xs:element name="AccessPolicyConfig" type="xs:boolean"/>
    <xs:element name="X.509Token" type="xs:boolean"/>
    <xs:element name="SAMLToken" type="xs:boolean"/>
    <xs:element name="KerberosToken" type="xs:boolean"/>
    <xs:element name="RELToken" type="xs:boolean"/>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute/>
```

```
</xs:complexType>
```

3.2 Example 2, extension declaration after element where minOccurs does not equal maxOccurs

Type 2 is extended in the following way:

```
<xs:complexType name="SystemCapabilities">
  <xs:sequence>
    <xs:element name="DiscoveryResolve" type="xs:boolean"/>
    <xs:element name="DiscoveryBye" type="xs:boolean"/>
    <xs:element name="RemoteDiscovery" type="xs:boolean"/>
    <xs:element name="SystemBackup" type="xs:boolean"/>
    <xs:element name="SystemLogging" type="xs:boolean"/>
    <xs:element name="FirmwareUpgrade" type="xs:boolean"/>
    <xs:element name="SupportedVersions" type="tt:OnvifVersion"
      maxOccurs="unbounded"/>
    <xs:element name="Extension" type="tt:SystemCapabilitiesExtensionType"
      minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="SystemCapabilitiesExtensionType">
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

3.3 Example 3, proprietary extension declaration using attributes.

A schema with proprietary extensions for type 2 using attributes is declared as shown in the following example:

```
<xs:schema ... xmlns:vendor = "http://www.vendor.com/2009/schema ...>
...
   <xs:complexType name="SystemCapabilities">
     <xs:sequence>
       <xs:element name="DiscoveryResolve" type="xs:boolean"/>
       <xs:element name="DiscoveryBye" type="xs:boolean"/>
       <xs:element name="RemoteDiscovery" type="xs:boolean"/>
       <xs:element name="SystemBackup" type="xs:boolean"/>
       <xs:element name="SystemLogging" type="xs:boolean"/>
       <xs:element name="FirmwareUpgrade" type="xs:boolean"/>
       <xs:element name="SupportedVersions" type="tt:OnvifVersion"
         maxOccurs="unbounded"/>
       <xs:element name="Extension" type=”tt:SystemCapabilitiesExtensionType"
         minOccurs="0"/>
     </xs:sequence>
     <xs:attribute ref="vendor:VersionAttribute" />
     <xs:attribute ref="vendor:LocalStorage" />
     <xs:attribute ref="vendor:HerculesIVEngine" />
     <xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="SystemCapabilitiesExtensionType">
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
...
</xs:schema>

<xs:schema ... targetNamespace = "http://www.vendor.com/2009/schema ...>
```

```
...
  <xs:simpleType name="VendorVersion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Version 1"/>
      <xs:enumeration value="Version 2"/>
      <xs:enumeration value="Version 3"/>
      <xs:enumeration value="Version 4"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:attribute name="VersionAttribute" type="VendorVersion"/>
  <xs:attribute name="LocalStorage" type="xs:boolean" />
  <xs:attribute name="HerculesIVEngine" type="xs:boolean" />

...
</xs:schema>
```

3.4 Example 4, proprietary extension using element declaration for type 1

The example below shows how to do a proprietary extension to the ONVIF 1.0 schema (for an element of type 1) when extension option 3) is used. This option should *only be used* when option 1) or option 2) cannot be used as this option *does not* interoperate with similar extensions from other vendors.

```
<xs:schema ... xmlns:vendor = "http://www.vendor.com/2009/schema ...>
...

<xs:complexType name="SecurityCapabilities">
  <xs:sequence>
    <xs:element name="TLS1.1" type="xs:boolean"/>
    <xs:element name="TLS1.2" type="xs:boolean"/>
    <xs:element name="OnboardKeyGeneration" type="xs:boolean"/>
    <xs:element name="AccessPolicyConfig" type="xs:boolean"/>
    <xs:element name="X.509Token" type="xs:boolean"/>
```

```
    <xs:element name="SAMLToken" type="xs:boolean"/>
    <xs:element name="KerberosToken" type="xs:boolean"/>
    <xs:element name="RELToken" type="xs:boolean"/>
    <xs:element ref="vendor:SecurityLevelElement" minOccurs="0">
    <xs:element ref="vendor:IPSEC" minOccurs="0">
    <xs:element ref="vendor:SRTP" minOccurs="0">
    <xs:any namespace="##targetnamespace" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>
...
</xs:schema>

<xs:schema ... targetNamespace = "http://www.vendor.com/2009/schema ...>

...
  <xs:simpleType name="SecurityLevel">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Low"/>
      <xs:enumeration value="Mid"/>
      <xs:enumeration value="High"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="SecurityLevelElement" type="SecurityLevel" />
  <xs.element name="IPSEC" type="xs:boolean" />
  <xs:element name="SRTP" type="xs:boolean" />

...
</xs:schema>
```

3.5 Example 5, proprietary extension using element declaration for type 2

The example below shows how to do a proprietary extension to the ONVIF 1.0 schema (for an element of type 2) when extension option 3) is used. This option should *only be used* when option 1) or option 2) cannot be used as this option *does not* interoperate with similar extensions from other vendors.

```
<xs:complexType name="SystemCapabilities">
  <xs:sequence>
    <xs:element name="DiscoveryResolve" type="xs:boolean"/>
    <xs:element name="DiscoveryBye" type="xs:boolean"/>
    <xs:element name="RemoteDiscovery" type="xs:boolean"/>
    <xs:element name="SystemBackup" type="xs:boolean"/>
    <xs:element name="SystemLogging" type="xs:boolean"/>
    <xs:element name="FirmwareUpgrade" type="xs:boolean"/>
    <xs:element name="SupportedVersions" type="tt:OnvifVersion"
      maxOccurs="unbounded"/>
    <xs:element name="Extension" type=”tt:SystemCapabilitiesExtensionType"
      minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="SystemCapabilitiesExtensionType">
  <xs:sequence>
    <xs:element ref="vendor:VersionElement" minOccurs="0">
    <xs:element ref="vendor:LocalStorage" minOccurs="0">
    <xs:element ref="vendor:HerculesIVEngine" minOccurs="0">
    <xs:any namespace="##targetnamespace" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:schema ... targetNamespace = "http://www.vendor.com/2009/schema ...>

...
  <xs:simpleType name="VendorVersion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Version 1"/>
      <xs:enumeration value="Version 2"/>
      <xs:enumeration value="Version 3"/>
      <xs:enumeration value="Version 4"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="VersionElement" type="VendorVersion" />
  <xs.element name="LocalStorage" type="xs:boolean" />
  <xs:element name="HerculesIVEngine" type="xs:boolean" />

...
<xs:schema>
```

4 Next versions

In next versions of the specification the introduced extensions points can be used to extend the specification as wanted. The target namespace will be reserved for the ONVIF organization and any proprietary extension needs to be kept in the points already reserved for those. If a vendor wants vendor specific backward or forward compatible extensions, he/she needs to define own extensions elements. However, this is *not recommended* as we will not have interoperability with proprietary extensions from *other vendors.* It is instead recommended to use new commands or attributes for the proprietary functions as previously discussed.

Next, we give some further examples in order to illustrate the backward-forward compatibility version changes in the *second version* of the specification. Similar techniques shall be used for all future specification changes.

4.1 Example 6, second version extension declaration after minOccurs = maxOccurs element

Type 1 is extended in the second specification version in the following way:

```
<xs:complexType name="SecurityCapabilities">
  <xs:sequence>
    <xs:element name="TLS1.1" type="xs:boolean"/>
    <xs:element name="TLS1.2" type="xs:boolean"/>
    <xs:element name="OnboardKeyGeneration" type="xs:boolean"/>
    <xs:element name="AccessPolicyConfig" type="xs:boolean"/>
    <xs:element name="X.509Token" type="xs:boolean"/>
    <xs:element name="SAMLToken" type="xs:boolean"/>
    <xs:element name="KerberosToken" type="xs:boolean"/>
    <xs:element name="RELToken" type="xs:boolean"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name ="FurtherSecurityCapabilities"
      type="tt:SecurityCapabilitiesONVIF2" minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="SecurityCapabilitiesONVIF2">
  <xs:sequence>
    <xs:element name="SecureSigning" type="xs:boolean" />
    <xs:element name="ONVIFPolicyFormat" type="xs:boolean" />
    <xs:any namespace="##targetnamespace" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

4.2 Example 2, second version extension declaration after minOccurs does not equal

maxOccurs element

Type 2 is extended in the second specification version in the following way:

```
<xs:complexType name="SystemCapabilities">
  <xs:sequence>
    <xs:element name="DiscoveryResolve" type="xs:boolean"/>
    <xs:element name="DiscoveryBye" type="xs:boolean"/>
    <xs:element name="RemoteDiscovery" type="xs:boolean"/>
    <xs:element name="SystemBackup" type="xs:boolean"/>
    <xs:element name="SystemLogging" type="xs:boolean"/>
    <xs:element name="FirmwareUpgrade" type="xs:boolean"/>
    <xs:element name="SupportedVersions" type="tt:OnvifVersion"
      maxOccurs="unbounded"/>
    <xs:element name="Extension" type="tt:SystemCapabilitiesExtensionType"
      minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="SystemCapabilitiesExtensionType">
  <xs:sequence>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name ="FurtherSystemCapabilities"
      type="tt:SystemCapabilitiesONVIF2" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SystemCapabilitiesONVIF2">
  <xs:sequence>
    <xs:element name="IOInput" type="xs:boolean" />
    <xs:element name="USB" type="xs:boolean" />
    <xs:any namespace="##targetnamespace" processContents="lax" minOccurs="0"
```

```
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

5 References

[1] D. Occard, “Extensibility, XML Vocabularies, and XML Schema”, O’Reilly XML.com, October 2004, http://www.xml.com/pub/a/2004/10/27/extend.html

[2] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008, http://www.w3.org/TR/REC-xml/

[3] XML Schema Part 1: Structures, W3C Recommendation 2 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#non-ambig