ONVIF™

XML Schema Version and Extension Handling White Paper

Version 1.2

December, 2015

1 Background

1.1 Purpose

Version *and* extensions handling for XML schemas are *rather complex issues* and care has to be taken when writing standards based on XML schemas.

This white paper analyzes the XML schema version and extension handling and gives recommendations for versions and extensions of ONVIF defined schemas. With the advent of XML 1.1 schema extension has become much easier so that this version of the document is much more compact than previous ones.

2 Extension requirements and limitations

ONVIF takes the following compatibility considerations and assumes the following limitations on 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.
- ONVIF does 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 Guidelines for vendor specific extensions

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 from 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.
3) If it is not possible to declare the proprietary extensions using new commands or attributes, the proprietary extensions shall be declared adding a unique mandatory extension element before the extension point and then add all new proprietary element in this extensions element. For details on this kind of proprietary extensions see also

Vendor extensions shall use neither the target namespace nor any namespace owned by ONVIF. This requirement prohibits any namespace clashes with future extensions by ONVIF.

4 Recommended schema definition principles

Each complex data type declaration in the schema shall have an extension points. The extension point shall use an xs:any. An extension point allows ONVIF to extend the functionality in future releases of the specification. Additionally the extension point allows vendors to add vendor specific data types. Due to possible conflicts vendors are strongly discouraged to make use of this mechanism and preferably use own methods or attributes which are much easier to co-exist.

ONVIF describes in an XML comment behind the any which extension order shall be applied. Vendor extensions must use vendor allocated namespaces and the vendor extensions should occur after the ONVIF extensions to ease parsing of ONVIF extensions. Due to the fact that ONVIF historically defined that vendor extensions occur first, both orders may occur.

As ONVIF is now referring to XML schema 1.1 any sequence shall be followed by an xs:any as follows:

```
<xs:any namespace=”##any” procesContents=”lax” minOccurs= “0”
maxOccurs=”unbounded”/>.
```

Additionally an

```
<xs:anyAttribute/>
```

shall be added after each sequence declaration.

ONVIF uses the same extension point for both future ONVIF extensions as well as vendor proprietary extensions. Both can coexist in the same extension point by using well-defined namespaces. It must be clearly defined which extension point comes first so that the party defining the first extension can leave room for the other one.

Unfortunately, most of the tools do not understand the namespace distinction needed to share a single extension point. This results in the fact that the parser of the receiving end decodes the secondary extension into the first any element and the application software has to retrieve the information from first any element instance via an editing DOM parsing.

The following examples illustrate the principles. The examples utilize two different ONVIF types from the preliminary schema (without extension points), called type 1 and type 2.

5 Examples

5.1 *Complex type with extension points*

The type Features is defined in the following way in order to ensure both ONVIF and vendor extensibility:

```
<xs:complexType name="Features">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
       maxOccurs="unbounded"/> <!-- ONVIF first Vendor second -->
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>
```

5.2 *Example 1, element extension by ONVIF*

If ONVIF wants to add an element it shall simply precede the any element with the newer content as shown in the following example:

```
<xs:complexType name="Features">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Age" type="xs:int"/>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
       maxOccurs="unbounded"/> <!-- ONVIF first Vendor second -->
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>
```

5.3 *Example 2, element extension by vendor*

The Vendor has to use the following extension technique to add the similar parameter Age

```
<xs:complexType name="Features">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:any namespace="##targetnamespace" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="vv1:Age"/>
    <xs:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute/>
</xs:complexType>
```

With vv1 being the namespace prefix of the vendor. As the change to the schema is more complex, the implementer has to apply the changes carefully.

Note that for extension points that defined as shown below the roles of extending are swapped between vendor and ONVIF.

```
<xs:any namespace=”##any” procesContents=”lax” minOccurs= “0”
maxOccurs=”unbounded”/> <!-- Vendor first, ONVIF second -->
```

5.4 *Example 3, attribute extension by vendor*

The Vendor has to use the following extension technique to add the similar parameter Age

```
<xs:complexType name="Features">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
       maxOccurs="unbounded"/> <!-- ONVIF first Vendor second -->
  </xs:sequence>
  <xs:attribute ref="vv1:Age"/>
  <xs:anyAttribute/>
</xs:complexType>
```

With vv1 being the namespace prefix of the vendor. The example shows that this kind of extension is straightforward for a vendor.

6 Extensibility in ItemList data structure

The ItemList data structure is utilized in event payload, rule configuration, analytics module configuration and action configuration. The ItemListDescription defines the schema of acceptable ItemList data structure. The ItemListDescription defines the “name” and “data structure/type” of acceptable elements for event definition, rule configuration definitions, analytics module configuration definitions, and action configuration definitions. The extensibility for ONVIF future versions and vendors allows additional fields in event payload, configuration parameters in rule, analytics module, and actions.

It is recommended that the vendor proprietary items in configurations of rules, analytics modules, and actions should use a prefix in naming these proprietary items to prevent naming clashes with ONVIF current and future versions. For example, the revised rule description (in response to GetSupportedRules) contains additional proprietary items.

```
<tt:RuleDescription Name="tt:LineDetector">
 <tt:Parameters>
  <tt:SimpleItemDescription Name="Direction" Type="tt:Direction"/>
  <tt:ElementItemDescription Name="Segments" Type="tt:Polyline"/>
  <tt:ElementItemDescription Name="vendor:MyMagicParam1" Type="xs:float"/>
 </tt:Parameters>
 ...
</tt:RuleDescription>
```

A vendor may choose to use its company name as a suitable prefix. The following example demonstrates how the ItemList structure contains additional fields (in response to GetRules request);

```
<tt:RuleEngineConfiguration>
 <tt:Rule Name="MyLineDetector" Type="tt:LineDetector">
  <tt:Parameters>
   <tt:SimpleItem Name="Direction" Value="Any"/>
   <tt:ElementItem Name="Segments">
    <tt:Polyline>
     <tt:Point x="10.0" y="50.0"/>
     <tt:Point x="100.0" y="50.0"/>
    </tt:Polyline>
   </tt:ElementItem>
   <tt:SimpleItem Name=”vendor:MyMagicParam1” Value=”0.4”/>
  </tt:Parameters>
 </tt:Rule>
 ...
<tt:RuleEngineConfiguration>
```

It is recommended that the vendor proprietary items shall not be added into Source and Key elements of properties because properties are uniquely identified by the combination of their Topic, Source and Key values. It is recommended that the vendor proprietary items shall be added into Data element. It is recommended that a vendor shall use a prefix to name these additional proprietary items.

```
<tt:MessageDescription IsProperty="true">
 <tt:Source>
  <tt:SimpleItemDescription   Name="VideoSourceConfigurationToken"
                              Type="tt:ReferenceToken"/>
  <tt:SimpleItemDescriptionD  Name="VideoAnalyticsConfigurationToken"
                              Type="tt:ReferenceToken"/>
  <tt:SimpleItemDescription   Name="Rule" Type="xs:string"/>
 </tt:Source>
 <tt:Key>
  <tt:SimpleItemDescription   Name="ObjectId" Type="xs:integer"/>
 </tt:Key>
 <tt:Data>
  <tt:SimpleItemDescription   Name="IsInside" Type="xs:boolean"/>
  <tt:ElementItemDescription  Name="vendor:MyNewItem"
                              Type="vendor:ObjInfoType"/>
 </tt:Data>
```

```
</tt:MessageDescription>
```

ONVIF controls the naming of items in configuration parameters and events, therefore, we recommend vendors to use a prefix when naming vendor proprietary additional items in configuration parameters and events. This approach eliminates any naming clash with vendor extensions and ONVIF future versions.

To support backward and forward compatibility,

- A service requester (client, NVC) shall ignore the additional items in configuration parameters and events when a service requester (client, NVC) is unaware of these additional items. A subscriber shall not fail when a received event does not contain additional items since a publisher is unaware of these additional items.
- A service provider (server, NVT) may ignore the additional items in configuration parameters when a service provider (server, NVT) is unaware of these additional items. A service requester shall not send the additional items to a service provider that is unaware of these additional items.
- A service provider (server, NVT) shall have default values for additional configuration parameters since some service requesters will not provide these parameters.

7 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) 1.1 (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

Annex A. Revision History

Rev.	Date	Editor	Changes
1.0	Nov-2008	Christian Gehrman	Initial version
1.1	Aug-2011	Hans Busch Stefan Andersson Hasan Ozdemir	Added the description for extensibility in ItemList data structure (See 5 Extensibility in ItemList data structure
1.2	Dec-2015	Hans Busch	Completely revised for support of XML schema 1.1