XML Schema Versioning

Issue

What is the Best Practice for versioning XML schemas?

Introduction

It is clear that XML schemas will evolve over time and it is important to capture the schema's version. This write-up summarizes two cases for schema changes and some options for schema versioning. It then provides some 'best practice' guidelines for XML schema versioning.

Schema Changes – Two Cases

Consider two cases for changes to XML schemas:

Case 1. The new schema changes the interpretation of some element. For example, a construct that was valid and meaningful for the previous schema does not validate against the new schema.

Case 2. The new schema extends the namespace (e.g., by adding new elements), but does not invalidate previously valid documents.

Versioning Approaches

Some options for identifying a new a schema version are to:

1. Change the (internal) schema version attribute.
2. Create a schemaVersion attribute on the root element.
3. Change the schema's targetNamespace.
4. Change the name/location of the schema.

Option 1: Change the (internal) schema version attribute.

In this approach one would simply change the number in the optional version attribute at the start of the XML schema. For example, in the code below one could change version="1.0" to version="1.1"

```
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="1.0">
```

Advantages:

- Easy. Part of the schema specification.
- Instance documents would not have to change if they remain valid with the new version of the schema (case 2 above).
- The schema contains information that informs applications that it has changed. An application could interrogate the version attribute, recognize that this is a new version of the schema, and take appropriate action.

Disadvantages:

- The validator ignores the version attribute. Therefore, it is not an enforceable constraint.

Option 2: Create a schemaVersion attribute on the root element.

With this approach an attribute is included on the element that introduces the namespace. In the examples below, this attribute is named 'schemaVersion'. This option could be used in two ways.

Usage A: First, like option 1, this attribute could be used to capture the schema version. In this case, one could make the attribute required and the value fixed. Then each instance that used this schema would have to set the value of the attribute to the value used in the schema. This makes schemaVersion a constraint that is enforceable by the validator. With the example schema below, the instance would have to include a schemaVersion attribute with a value of 1.0 for the instance to validate.

```
<xs:schema xmlns="http://www.exampleSchema"
           targetNamespace="http://www.exampleSchema"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
          <xs:element name="Example">
               <xs:complexType>
                         ....
                    <xs:attribute name="schemaVersion" type="xs:decimal" use="required" fixed="1.0"/>
               </xs:complexType>
           </xs:element>
```

Advantages:

- The schemaVersion attribute is an enforceable constraint. Instances would not validate without the same version number.

Disadvantages:

- The schemaVersion number in the instance must match exactly. This does not allow an instance to indicate that it is valid using multiple versions of a schema.

Usage B: The second approach uses the schemaVersion attribute in an entirely different way. It no longer captures the version of the schema within the schema (i.e., it is not a fixed value). Rather, it is used in the instance to declare the version (or versions) of the schema with which the instance is compatible. This approach would have to be done in conjunction with option 1 (or an alternative indicator in the schema file to identify its version).

The schemaVersion attribute's value could be a list or a convention could be used to define how this attribute is used. For example, if the convention was that the schemaVersion attribute declares the latest schema version with which the instance is compatible, then the example instance below states that the instance should be valid with schema version 1.2 or earlier.

With this approach, an application could compare the schema version (captured in the schema file) with the version to which the instance reports that it is compatible.

Sample Schema (declares it's version as 1.3)

```
<xs:schema xmlns="http://www.exampleSchema"
        targetNamespace="http://www.exampleSchema"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified"
        version="1.3">
       <xs:element name="Example">
            <xs:complexType>
                    ....
                <xs:attribute name="schemaVersion" type="xs:decimal" use="required"/>
            </xs:complexType>
        </xs:element>
```

Sample Instance (declares it is compatible with version 1.2
(or 1.2 and other versions depending upon the convention used))

```
<Example schemaVersion="1.2"
        xmlns="http://www.example"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example MyLocation\Example.xsd">
```

Advantages:

- Instance documents may not have to change if they remain valid with the new schema version (case 2).
- Like option 1, an application would receive an indication that the schema has changed.
- Could provide an alternative to schemaLocation as a means to point to the correct schema version. This could be desirable where the business practice requires the use of a schema in a controlled repository, rather than an arbitrary location.

Disadvantages:

- Requires extra processing by an application. For example, an application would have to pre-parse the instance to determine what schema version with which it is compatible, and compare this value to the version number stored in the schema file.

Option 3: Change the schema's targetNamespace.

In this approach, the schema's 'targetNamespace' could be changed to designate that a new version of the schema exists. One way to do this is to include a schema version number in the designation of the target namespace as shown in the example below.

```
<xs:schema xmlns="http://www.exampleSchemaV1.0"
        targetNamespace="http://www.exampleSchemaV1.0"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
```

Advantages:

- Applications are notified of a change to the schema (i.e., an application would not recognize the new namespace).
- Requires action to assure that there are no compatibility problems with the new schema. At a minimum, the instance documents that use the schema, and schemas that include the relevant schema, must change to reference the new targetNamespace. This both an advantage and a disadvantage.

Disadvantages:

- With this approach, instance documents will not validate until they are changed to designate the new targetNamepsace. However, one does not want to force all instance documents to change, even if the change to the schema is really minor and would not impact an instance.
- Any schemas that 'include' this schema would have to change because the target namespace of the included components must be the same as the target namespace of the including schema.

Option 4: Change the name/location of the schema.

This approach changes the file name or location of the schema. This mimics the convention that many people use for naming their files so that they know which version is the most current (e.g., append version number or date to end of file name).

Advantages:

Disadvantages:

- As with option 3, this approach forces all instance documents to change, even if the change to the schema would not impact that instance.
- Any schemas that import the modified schema would have to change since the import statement provides the name and location of the imported schema.
- Unlike the previous options, with this approach an application receives no hint that the meaning of various element/attribute names has changed.
- The schemaLocation attribute in the instance document is optional and is not authoritative even if it is present. It is a hint to help the processor to locate the schema. Therefore, relying on this attribute is not a good practice (with the current reading of the specification).

XML Schema Versioning Best Practices

[1] Capture the schema version somewhere in the XML schema.

[2] Identify in the instance document, what version/versions of the schema with which the instance is compatible.

[3] Make previous versions of an XML schema available.

This allows applications to use previous versions. It also allows users to migrate to new versions of the schema as compatibility is assured.

One way to do this is to have applications pre-parse the instance and choose the appropriate schema based on the version number. For example, one could have the schemaLocation URI point to a document that includes a list of the locations of the available versions of the schema. A tool could then be used to obtain the correct version of the schema. The disadvantage of this approach is that this pre-parsing requires two passes at the XML instance (one to get the correct version of the schema and one to validate).

[4] When an XML schema is only extended, (e.g., new elements, attributes, extensions to an enumerated list, etc.) one should strive to not invalidate existing instance documents.

For example, if one is adding new elements or attributes, one could consider making them optional where this makes sense.

Also, one could come up with a convention for schema versioning to indicate whether the schema changed significantly (case 1) or was only extended (case 2). For example, for case 1 a version could increment by one (e.g., v1.0 to v2.0) whereas for case 2 a version could increment by less than one (e.g., v1.2 to v1.3).

In this case, a possible approach would be to do the following with respect to the schema:

a. Change the schema version number within the schema (e.g., option 1).
b. Record the changes in the schema in a change history.
c. Make the new and previous versions of the schema available (therefore, one would want to change the file name/location as well).

[5] Where the new schema changes the interpretation of some element (e.g., a construct that was valid and meaningful for the previous schema does not validate against the new schema), one should change the target namespace.

In this case, the changes with respect to the schema are the same as with [4], with one addition:

d. Change the target namespace.

In this case there are also required changes with respect to the instances that use this schema.

e. Update the instances to reflect the new target namespace.
f. Confirm that there are no compatibility problems with the new schema.
g. Change the attribute that identifies the version/versions of the schema with which the instance is valid.
h. Update the schema name/location if appropriate.