XML Schema Versioning Policies Version 1.0

Proposed Endorsed Note 2016-09-06

Working group
Standards and Processes
This version
http://www.ivoa.net/documents/schemaVersioning/20160906
Latest version
http://www.ivoa.net/documents/schemaVersioning
Previous versions
This is the first public release
Author(s)
Paul Harrison, Markus Demleitner, Brian Major, Pat Dowler
Editor(s)
Paul Harrison
Version Control
Revision 3489, 2016-07-19 05:08:19 -0700 (Tue, 19 Jul 2016)
https://volute.g-vo.org/svn/trunk/projects/grid/notes/schema-versioning/schemaVersioning.tex

Abstract

This note describes the recommended practice for the evolution of IVOA standard XML schemata that are associated with IVOA standards. The criteria for deciding what might be considered major and minor changes and the policies for dealing with each case are described.

Status of This Document

This is an IVOA Proposed Endorsed Note for review by IVOA members and other interested parties. It is appropriate to reference this document only as a Proposed Endorsed Note that is under review and may change before it is endorsed or may not be endorsed.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/Documents/.

Contents

Acknowledgments

The content of this note is derived from discussions that occurred in a splinter session at the June 2015 IVOA interoperability meeting in Sesto, Italy.

1 Introduction

Many of the standard protocols and data models developed by the International Virtual Observatory Association (IVOA) have used XML (?) for message or object serialization. The structure of these XML files has usually been constrained using the XML schema definition language (?), or XSD for short. The particular schema that has been associated with a standard is defined by the "target namespace" (hereafter refered to as simply "the namespace") of the schema – the namespace identifier itself is a URI that typically has a form that contains the version number of the standard. There exist in many programming languages XML parsers that can use the XSD schema automatically to do a strong check whether an instance of an XML document conforms to the given schema. If the XML document does conform to the structure defined by the schema then it is known as "valid" and even a small deviation from the specified structure will mean that the XML instance is "invalid". This strong check of validity is extremely useful in the context of interoperating services and clients within the VO as it guarantees that both sides of an interaction will agree on the structure of the document and hence its interpretation. To maintain this behaviour, the conventional XSD practice is to give the schema a new identity (namespace) if any changes are made to it so that both client and server can agree on the exact version of a schema that they are using for checking validity of an XML instance.

Once a VO service has been standardised there will typically be a growing number of clients that are coded against the particular version of the schema – any changes to the definition of the schema that is subsequently used by a new version of the service to create instance documents will result in immediate classification as "invalid" by the clients that have not themselves being updated to use the new schema. As explained above, in general this property is desirable for guaranteeing interoperability, but it does limit the ability even to correct errors in the original schema definition without causing disruption to the deployed clients.

A related problem occurs because, to namespace-aware XML parsers, the element name is a tuple of the namespace URI and the tag content. For instance, in the XML document

```
<doc xmlns="http://example.com/1.0"/>
```

the fully qualified name of the element includes the namespace and is conventionally written as `{http://example.com/1.0}doc`, and clients will expect this full form, whether or not they perform schema validation. As soon as the namespace changes, any client expecting this element name will no longer understand anything in the document. Rather typically, however, the client would still be able to make sense of the document as far as it is relevant to the client if it ignored the namespace part of the element name. In consequence, many clients started discarding the namespace part entirely, which then leads to new interoperability problems, for instance, when elements from a different namespace are embedded within such a document. Hence it is desirable to limit namespace changes to cases when legacy clients would definitely break.

This note describes the circumstances in which it is permissible to make changes to a schema and not change its namespace.

2 Schema Versioning

It is the case that two XML instances are formally regarded by XML conventions as not equivalent if the only textual difference between them is that the namespace declaration is not the same. We can use the observation that simply removing the namespace definition from both instances would make the instances both textually the same and equivalent in the strict XML sense to inform the class of changes can be to the schema definition without necessarily needing to change the namespace, and how a client should treat such changes.

2.1 Minor changes

In general the class of changes that might be considered minor are those which allow legacy clients (i.e. without rewriting) to keep functioning with content produced against the new schema. Specific structual changes to the schema that allow this goal to be achieved include:

- Not removing concepts (i.e. elements or attributes) from the old schema.
- Ensuring that any new concepts are optional.

Even with the restrictive conditions above it is still necessary that any consumer of XML instance documents takes the approach that it does not do strict schema validation against the version of the schema that it knows about, but rather ignores everything that it does not understand. This approach is allowable because any new concepts are optional even for consumers of the XML instance that are aware of the latest version of the schema, and so clients cannot use this information for fundamental changes in the behaviour of an IVOA protocol. In other words the IVOA protocol remains backwards compatible and would only warrant a "point change" in the standard version.

Conversely if the conditions above cannot be met by an evolution of an IVOA standard then it is a indication that the standard is undergoing a "major change" and backwards compatibility may be broken. Such a change would involve both a change in the first part of the standard document's version number as well as a change in the version used in the schema namespace.

2.1.1 Determining if the changes are indeed minor

Although the conditions outlined above for minor changes should generally be strictly adhered to in the design of minor extensions to standards, there are occasions where a new version of a standard might try to correct an error made in a previous version. For example a certain construct could have been schema valid which should not have been allowed in a correctly authored schema that expressed the intentions of the standard. In this case it is likely that the correction would break the first of the above conditions, but hopefully clients would not be written to expect the unintentionally valid construct and would not be unduely impacted by the change. It would be up to the IVOA Technical Coordination Group (TGC) to determine the likely impact of such a change and whether the violation of the constraints for a minor change would be allowable in such circumstances.

2.2 Indicating the version number

When allowing minor changes to the schema without changing the namespace it becomes imperative that the the minor version number of the schema is communicated via a means other than the namespace. This is true for both instance documents and schema documents.

2.2.1 Version numbers in instance documents

It is important that clients are able to know which version of the schema was used to create an XML instance. A common use case envisioned is that in service responses, this information can be used to discover a service's ability to correctly respond to newly specified parts of the protocols.

Hence, in service responses, root elements must have a *version* attribute exactly giving the value of the corresponding XSD file's *version* attribute.

When global elements from a different schema are included within other XML files, each of these global elements should have a version attribute of its own. Where the discovery of extended capabilities is not a likely use case, such version attributes may be left out.

2.2.2 Version numbers in schema files

It is of similar importance to software writers to know exactly the version of the schema that they are using. In the XSD definition there is an appropriate *version* attribute on the top level *<schema>* element that can be used for this purpose.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ivoa.net/xml/UWS/v1.0"
   xmlns:uws="http://www.ivoa.net/xml/UWS/v1.0"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="1.1-PR-20150626"
>
```

This version attribute is not used formally by the schema validation machinery but can be used as desired here to indicate a precise version of the schema given that the namespace is only used to differentiate between major revisions of the schema.

2.2.3 Version numbers in Namespace URIs

In the IVOA, namespace URIs traditionally contained the full version specification, as in `http://www.ivoa.net/xml/UWS/v1.0`. This practice is somewhat paradoxical, as changes in the minor version number should, by the IVOA's document standards (?), not break downwards compatibility, but a change in the namespace URI is, as argued above, incompatible in both directions.

Therefore, future namespace URI must only contain the major version number. The recommended format is to have the major version as the last URI element, for instance,

`http://www.ivoa.net/xml/UWS/2`.

An unfortunate side effect of the proposal made in this note is that legacy namespace URIs become somewhat confusing, as a namespace like `http://www.ivoa.net/xml/UWS/v1.0` can be defined by schema documents with actual versions 1.1, 1.2, etc.

Specifications employing such legacy namespace URIs should warn their readers to this effect. The suggested presentation is an admonition box with a content like this:

> The namespace URI `http://www.ivoa.net/xml/UWS/v1.0` is used for backwards compatibility. The presence of the string `v1.0` does *not* indicate that the document content conforms to version 1.0 of the schema, or can be validated against that schema. It does indicate, however, that documents valid according to version 1.0 of the schema still validate with the current schema file. It also means that any XML reader compliant with the recommentations of in Harrison et al (2016) [this would be a reference to the current version of this note] must be able to process the elements' content if they support any version 1.0 or later of this specification.
>
> The version an instance document actually conforms to can be obtained from the *`version`* attribute on the instance document's root element [or the element containing the namespace declaration, if applicable].

3 Hosting the Schema on the IVOA Web Site

As mentioned in the introduction the schemata are typically associated with a particular version of a standard and so are typically included in an appendix of the standard. In addition they are located on the ivoa website `http://www.ivoa.net/xml` to allow software authors interactively to obtain the latest version directly as a file.

The availability of the schemata via the IVOA web site also allows XML parsers to be instructed to fetch the schema automatically when validating an XML instance. This is done using the *`xsi:schemaLocation`* attribute which pairs a namespace with the URL of a schema that has definitions in that namespace. Although there is no particular necessity to do so, it has been standard practice to use namespaces that have a one-to-one correspondence with a location on the IVOA web site. The advantage of this approach is that the namespace is "owned" by the IVOA and there is therefore much less chance of accidental clashes of namespaces.

Thus a current schema is usually available at two URLs on the IVOA site;

1. at a URL that corresponds to the namespace
   e.g. `http://www.ivoa.net/xml/UWS/v1.0`

2. at a URL that corresponds to the filename of the schema
   e.g. `http://www.ivoa.net/xml/UWS/UWS-v1.0.xsd`

As this note now recommends that there are potentially many minor versions of a schema all in the same namespace the actual file than the first from of the URL above points to must be updated when a new minor version is published. The various minor versions of a particular schema should be available at the "filename" style URLs of the second kind. It should be noted that the actual file pointed to by the "namespace" style URL should only be updated to point to the new minor version when the associated standard has reached the "recommendation" stage.

4 Client use of schema

Using this strategy means that client software that simply uses validation with the parser utilizing this schema location hint will always pick up the latest version of the schema, which should mean that the instance document will always be valid assuming that the server has created a strictly valid document. Note that this still applies even if the server is using an older (minor) version of the schema, as new versions are only allowed to add optional features to the schema.

It is often the case that client software is written to use a local copy of the schema to avoid the overhead of continually downloading the schema. In this case it is possible that the XML instance document will not be valid against the local schema if the server is working against a newer minor version of the schema – as it could have added a new optional element into the XML instance. It is for this reason that the client side of an IVOA service should be forgiving in ignoring elements that it is not expecting. It is possible for the client to recognise that the XML instance is valid against a newer version of the schema by comparing the version attribute of the root element of the XML instance against the version attribute in the local XSD schema document.

5 Summary of Recommendations

This section summarizes the main recommendations contained within this document.

- The namespace URI should contain only the associated standard major number, and therefore does not change when a minor version increment occurs. e.g. `http://www.ivoa.net/xml/ABC/v1`. However, existing namespace URIs which contain a "`.0`" minor increment suffix should not be changed to conform to this rule.

- Include a *version* attribute in the top level element to allow client version discovery – this version number should include the full standard version number including the minor version increment. – 1.1

- Set the *version* attribute of the *<schema>* element in the XSD to be equal to the full standard version including the minor version increment and document status and version – e.g. 1.1-WD-20150607

- Minor version changes should not break clients – i.e. in general they should only add new elements/attributes, not remove formerly valid content – correspondingly, this means that clients should quietly ignore things that they do not know about in the XML – i.e. they should not automatically issue an error if schema validation fails (though of course if they can determine that the validation error is because of a missing required element then they should issue an error).

- In contrast servers must produce strictly valid documents, and service validators must test strict validity against the relevant schema (discovered from the namespace and the version element).

A Changes from Previous Versions

No previous versions yet.