

The W3C's Object-Oriented Descriptions for XML



XML Schema

O'REILLY®

Eric van der Vlist

XML Schema

Eric van der Vlist

Documenting Schemas

The issue of documenting schemas—or any machine readable language—goes beyond simple additions of comments. The real challenge is to create schemas that are readable both directly by looking at their source code and by documentation extraction tools.

Style Matters

Writing schemas is much like writing programs. Two pieces of code may both work, but one is more readable and maintainable than the other. Readability is good.

Keep It Simple

Although W3C XML Schema has been carefully specified so that schema processors can find their way through the most complex and intricate combinations of its many features, the same can't be expected of the average human reader. I must confess that I, for one, am getting rapidly lost in the meanders of medium complexity schemas, such as the famous schema for schema.

“Keep it simple” is a useful principle. Although W3C XML Schema gives you a huge number of features, you don't need to use all of them in every schema. Each of them incurs a price in terms of readability of your schema.

Some of the rules for simplicity that we have used for some time with programming languages apply here, such as the conflicting rules for brevity (“If a function is more than one page long, split it”), and directness (“A function should be called more than once”). There are, of course, others such as “Put the code and the documentation in the same place,” “If you can't say it in English, you can't say it in C/C++ (or Java, C#, Perl, Python, etc.),” and “Don't solve problems that don't exist.”

Translated for the XML design world, these four could read as “If a declaration is more than one page long, split it,” “A declaration should be referred to at least one time” (we will see next that there might be exceptions), “If you can't say it in

English, you can't say it in XML," and, of course, "Don't solve problems that don't exist."

Think Globally

When I started working with W3C XML Schema, I used to think that the Russian doll design was the simplest, since it's so close to the structure of the instance documents. Having written the W3C XML Schema reference manual, I am convinced that flat structures in which all the elements are global are much simpler to document and just as simple to write!

The Russian doll design relies on an analogy between object-oriented programming and markup. This is somewhat misleading: there is no such thing as a private or local object in an XML document (except maybe if you encode or encrypt some fragments); when you open an XML document, its whole content is exposed, and everything is public and needs to be documented with the same level of accuracy. To describe a concept, give it a name. W3C XML Schema enforces the attribution of unique names only for global elements. Although different content models are often presented as an advantage over the DTDs, defining them under the same element name is very confusing when reading an instance document. The most convenient way to create a reference manual for an XML vocabulary is through a dictionary of elements. Reusing the same element name for different purposes creates multiple entries that are confusing and difficult to read (like the entries for common words, such as "place" in an English dictionary); the example of W3C XML Schema and its very different meanings for `xs:extension` is enlightening. Therefore, the second piece of advice is to define the elements as global when possible. Note that this advice doesn't apply to unqualified attributes, which cannot be defined as global.

When It's Similar, Show It

The third and last piece of advice contradicts the first one, and a trade-off needs to be found between these two. The first two bits of advice lead to what I call flat schemas. These are similar to our very first example in Chapter 1, in which all the elements and attributes are global with local type definitions. This style is easy to read but doesn't highlight the similarities between elements such as the fact that authors and characters can be considered persons and share some properties. When strong similarities exist between different elements, using one of the techniques already discussed (either a complex type derivation or elements and attributes group composition) can enhance the readability of the schema.

The third bit of advice states you should use W3C XML Schema features to highlight the strong similarities when they are present.

The W3C XML Schema Annotation Element

The recommended way to add comments and documentation in a W3C XML Schema is through the `xs:annotation` element. This element can be added within pretty much all the W3C XML Schema elements (in fact, it can be added within all the schema elements except `xs:annotation` and its child elements, `xs:documentation` and `xs:appinfo`). It generally appears as the first child element (except for the `xs:schema` element in which `xs:annotation` can appear anywhere).

The `xs:annotation` element is a container for the `xs:documentation` and `xs:appinfo` elements that contain additional information. These two elements are dedicated to holding human readable documentation (`xs:documentation`) and machine-processable information (`xs:appinfo`). They accept any text and child elements. (These are the only W3C XML Schema elements that have a mixed content model.) Note, though, that the schema for schema specifies that the processing to apply to the content of these elements is lax. Concretely, this means that although W3C XML Schema elements can be included within these elements, they must be valid per the schema for schema. This mixed content model allows the inclusion of almost any content, such as text:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The author of a book.
    </xs:documentation>
    <xs:documentation xml:lang="fr">
      Designe l'auteur d'un livre.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

It also allows rich content, such as XHTML, which can be assembled to create more user friendly and readable documentation:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p id="author" xmlns="http://www.w3.org/1999/xhtml">
        This element describes the
        <em>
          author
        </em>
        of a
        <a href="#book">
          book
        </a>
      </p>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

It even allows SVG, such as the following, which provides a picture of what an author may look like:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:documentation>
      <svg xmlns="http://www.w3.org/2000/svg">
        <title>
          An author
        </title>
        <ellipse style="stroke:#000000; fill:#e3e000;
          stroke-width:2pt;" id="head" cx="280" cy="250" rx="110"
          ry="130"/>
        <ellipse style="stroke:none; fill:#7f7f7f; " id="leftEye"
          cx="240"cy="225" rx="18" ry="18"/>
        <ellipse style="stroke:none; fill:#7f7f7f; " id="rightEye"
          cx="320"cy="225" rx="18" ry="18"/>
        <path style="fill:none;stroke:#7F7F7F; stroke-width:5pt;"
          id="mouth"d="M 222 280 A 58 48 0 0 0 338 280"/>
      </svg>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

Dublin Core elements are a set of elements widely used on the Web to qualify web pages and supported by a large range of applications. They may be used as general purpose metadata embedded within annotations:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:appinfo xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:creator>
        Eric van der Vlist (mailto:vdv@dyomedeia.com)
      </dc:creator>
      <dc:date>
        2002-02-01
      </dc:date>
      <dc:subject>
        author, person, book
      </dc:subject>
      <dc:description>
        This element describes the author of a book.
      </dc:description>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Annotations are also a good container for application-specific metadata, such as those used by the schema for schema to describe the list of facets and properties of its primitive datatypes:

```
<xs:simpleType name="string" id="string">
  <xs:annotation>
    <xs:appinfo>
```

```

    <hfp:hasFacet name="length"/>
    <hfp:hasFacet name="minLength"/>
    <hfp:hasFacet name="maxLength"/>
    <hfp:hasFacet name="pattern"/>
    <hfp:hasFacet name="enumeration"/>
    <hfp:hasFacet name="whiteSpace"/>
    <hfp:hasProperty name="ordered" value="false"/>
    <hfp:hasProperty name="bounded" value="false"/>
    <hfp:hasProperty name="cardinality" value="countably
    infinite"/>
    <hfp:hasProperty name="numeric" value="false"/>
  </xs:appinfo>
  <xs:documentation
    source="http://www.w3.org/TR/xmlschema-2/#string"/>
</xs:annotation>
<xs:restriction base="xs:anySimpleType">
  <xs:whiteSpace value="preserve" id="string.preserve"/>
</xs:restriction>
</xs:simpleType>

```

The Schema Adjunct Framework (SAF) is a proposal to complement schemas with the information needed to generate applications. It is found at <http://www.extensibility.com/saf/spec>. One of its syntaxes, “schema adornments,” also uses `xs:appinfo`; although this syntax hasn’t been adapted to the W3C XML Schema Recommendation yet, it could be something like this (note that this is my own unofficial adaptation given here just as an example):

```

<xs:element name="author" type="author">
  <xs:annotation>
    <xs:appinfo source="saf:meta-data-item"
      xmlns:sql="http://www.extensibility.com/saf/spec/safsample/sql-map.saf"
      >
      <sql:select>
        select
          <sql:elem>
            name
          </sql:elem>
          ,
          <sql:elem>
            birthdate
          </sql:elem>
          ,
          <sql:attr>
            deathdate
          </sql:attr>
        from tbl_author
      </sql:select>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

SAF also defines a syntax to embed rules written as XPath expressions, which is also the domain of Schematron (which is discussed in Appendix A). Schematron rules can

be embedded in `xs:appinfo` elements and used to test things that W3C XML Schema cannot—like ensuring that the birth of a person took place before their death.

How can this work, given that the XPath 1.0 on which Schematron is built can't interpret dates? This is a fairly dangerous practice, but assuming that the dates use four digits for the years, as well as the same time zone (which we saw how to impose through patterns in Chapter 6), this works because ISO 8601 dates are then following the alphabetical sort order!

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:appinfo xmlns:sch="http://www.ascc.net/xml/schematron">
      <sch:pattern name="Born before dead">
        <sch:rule context="author">
          <sch:assert test="not(dead) or (dead > born)"
            diagnostics="bornAfterDead">
            An author should die after her or his death.
          </sch:assert>
          <sch:diagnostics>
            <sch:diagnostic id="bornAfterDead">
              Error, this author is born after her or his birth!
              Author=
              <sch:value-of select="name"/>
              Birth =
              <sch:value-of select="born"/>
              Death =
              <sch:value-of select="dead"/>
            </sch:diagnostic>
          </sch:diagnostics>
        </sch:rule>
      </sch:pattern>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Although not a common practice, it is also possible to embed code snippets, such as this XSLT template:

```
<xs:element name="book" type="book">
  <xs:annotation>
    <xs:appinfo xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:template match="book">
        <xsl:apply-templates select="title"/>
        <xsl:apply-templates select="isbn"/>
        <p>
          Authors:
        </p>
        <ul>
          <xsl:apply-templates select="author"/>
        </ul>
        <p>
          Characters:
        </p>
      </xs:template>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```

    <ul>
      <xsl:apply-templates select="character"/>
    </ul>
  </xsl:template>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

Instead of embedding resources, you can also link them using XLink. RDDL, a vocabulary aimed at describing namespaces may be diverted from its original goal to provide the glue for expressing these links:

```

<xs:element name="author" type="author">
  <xs:annotation>
    <xs:appinfo xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:rddl="http://www.rddl.org/">
      <rddl:resource id="author-transform"
        xlink:arcrole="http://www.w3.org/1999/xhtml"
        xlink:role="http://www.w3.org/1999/XSL/Transform"
        xlink:title="Author template"
        xlink:href="library.xslt#author">
        <div class="resource">
          <h4>
            XSLT Transformation
          </h4>
          <p>
            This
            <a href="library.xslt#author">
              template
            </a>
            displays the description of an author.
          </p>
        </div>
      </rddl:resource>
      <rddl:resource id="CSS" xlink:title="CSS Stylesheet"
        xlink:role="http://www.isi.edu/in-notes/iana/assignments/media-types/text/css"
        xlink:href="author.css">
        <div class="resource">
          <h4>
            CSS Stylesheet
          </h4>
          <p>
            A
            <a href="author.css">
              CSS stylesheet
            </a>
            defining the styles which may be used to display an author.
          </p>
        </div>
      </rddl:resource>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

RDDL should be pronounced “riddle,” and its specification is available at <http://rddl.org>.



`xs:documentation` and `xs:appinfo` both accept an optional `source` attribute, which is a URI, and can identify the source or the nature of the included information. Since it's designed for human consumption, `xs:documentation` also accepts an optional `xml:lang` attribute.

Foreign Attributes

Although `xs:annotation` is certainly the most flexible way to embed any information within a schema, W3C XML Schema defines a second opening in its vocabulary that may be used either as an alternative or in conjunction with annotations. All the W3C XML Schema elements (except `xs:documentation` and `xs:appinfo`) accept any attribute that has a namespace other than the W3C XML Schema namespace (unprefixed attributes are forbidden). Such attributes may be used to document a schema:

```
<xs:element name="author" type="author" doc:doc="This element
describes the author of a book."
xmlns:doc="http://dyomedea.com/ns/doc"/>
```

This approach is also used by SAF adornments, whose simple form can be embedded in attributes:

```
<xs:element name="author" type="author" sql:table="TBL_AUTHOR"
xmlns:sql="http://www.extensibility.com/saf/spec/safsafsample/sql-map.saf"
/>
```

The huge opening given to attributes is especially interesting with attribute-only vocabularies such as XLink, and simple XLinks can be directly embedded into W3C XML Schema. The following example links the definition of our `author` element directly to a XSLT template:

```
<xs:element name="author" type="author"
xlink:arcrole="http://www.w3.org/1999/xhtml"
xlink:role="http://www.w3.org/1999/XSL/Transform"
xlink:title="Author template" xlink:href="library.xslt#author"
xmlns:xlink="http://www.w3.org/1999/xlink"/>
```

Unfortunately, because of the exception for `xs:appinfo` and `xs:documentation`, which do not accept foreign attributes, metadata cannot be added through attributes in these elements, so the following example is invalid:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:appinfo
      xlink:arcrole="http://www.w3.org/1999/XSL/Transform"
      xlink:role="http://www.w3.org/1999/XSL/Transform"
      xlink:title="Author template" xlink:href="library.xslt#author"
      xmlns:xlink="http://www.w3.org/1999/xlink">
      <div class="resource">
```

```

    <h4>
      XSLT Transformation
    </h4>
    <p>
      This
      <a href="library.xslt#author">
        template
      </a>
      displays the description of an author.
    </p>
  </div>
</xs:appinfo>
<xs:appinfo title="CSS Stylesheet"
  role="http://www.isi.edu/in-notes/iana/assignments/media-types/text/css"
  href="author.css">
  <div class="resource">
    <h4>
      CSS Stylesheet
    </h4>
    <p>
      A
      <a href="author.css">
        CSS stylesheet
      </a>
      defining the styles which may be used to display an
      author.
    </p>
  </div>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

Of course, the usual limitations of attributes apply here: attributes are less extensible than elements and they cannot include structured content.

XML 1.0 Comments

There is a general tendency among recent XML vocabularies to use special elements to add documentation or meta-information rather than general purpose XML 1.0 comments or processing instructions. W3C XML Schema follow this tendency with its `xs:documentation` and `xs:appinfo` elements. The arguments often used to justify this choice are the ease of use by applications and the fact that parsers are not obliged to transmit comments to applications. In practice, most parsers do transmit comments; comments are available to the applications and can be manipulated quite easily, even through XPath expressions. A more solid argument in favor of using elements is their ability to include structured content.

On the other hand, XML 1.0 comments (and processing instructions) are lighter weight and can be included at any location within any element—not only as the first

child element of XML Schema components. With an XML 1.0 comment, our example becomes:

```
<xs:element name="author" type="author">
  <!-- This element describes the author of a book. -->
</xs:element>
```

Which One and What For?

If we recap the three different forms we have seen to include simple comments, we may use `xs:documentation`:

```
<xs:element name="author" type="author">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The author of a book.
    </xs:documentation>
    <xs:documentation xml:lang="fr">
      Designe l'auteur d'un livre.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

Or foreign attributes:

```
<xs:element name="author" type="author" doc:doc="This element
describes the author of a book."
xmlns:doc="http://dyomedea.com/ns/doc"/>
```

Or XML 1.0 comments:

```
<xs:element name="author" type="author">
  <!-- This element describes the author of a book. -->
</xs:element>
```

Which one is more appropriate? There is no simple answer, since it depends on which form is supported by the set of tools you choose, and on what you want to do out of these comments. That being said, does that really matter? The hard work is to include relevant comments in your schemas and to keep them up to date. The format itself is not really that important, and each approach is only one XSLT transformation away from the two other alternatives!

The same applies to all the meta-information that we included in our schemas: they do not break the schema and can be processed by any schema processor but will be ignored by most of them! Adding them is useful only if one has an application in mind, and the choice of the format will be determined by the constraints of this application.

Finally, we must note that we have only presented schema-centric options, in which information is added within W3C XML Schema that conforms to the Recommendation. Other options exist, which may be object-oriented design-centric (generating schemas from UML specifications) or adjunct-centric (describing the schema adjunct

outside of the W3C XML Schema document). Also, some new inventive packaging could be invented, such as formats that would group different resources (schemas, transformations, stylesheets, etc.) for an information item under a single umbrella, and perhaps recombine the traditional “per function” packaging, depending on the composition done by the application.