# XML Matters #7

## Comparing W3C XML Schemas and Document Type Definitions (DTDs)

David Mertz, Ph.D. (mertz@gnosis.cx)
Idempotentate, Gnosis Software, Inc.
March 2001

Many developers expect that XML schemas will soon supplant DTDs for specifying XML document types. David Mertz is skeptical that schemas will replace DTDs, though he believes that XML schemas are an invaluable tool in a developer's arsenal. This installment of the "XML Matters" column steps up to the challenge of comparing schemas and DTDs and clarifying just what is going on in the XML schema world.

While there are a number of instances where W3C XML Schemas excel, there remain, nonetheless, a number of areas where DTDs are better. Developers are continually left with tough choices (which is not unusual in the XML world). Let's begin the journey of sorting through some of those choices.

### The state of affairs

Much of the point of using XML as a data representation format is the possibility of specifying structural requirements for documents: rules for exactly what types of content and subelements may occur within elements (and in what order, cardinality, etc.). In traditional SGML circles, the representation of document rules has been as DTDs -- and indeed the formal specification of the *W3C XML 1.0 Recommendation* explicitly provides for DTDs. However, there are some things that DTDs cannot accomplish that are fairly common constraints; the main limitation of DTDs is the poverty in their expression of data types (you can specify that an element must contain PCDATA, but not that it must contain, for example, a nonNegativeInteger). As a side matter, DTDs do not make the specification of subelement cardinality easy (you can compactly specify "one or more" of a subelement, but specifying "between seven and twelve" is, while possible, excessively verbose, or even outright contorted).

In answer to various limitations of DTDs, some XML users have called for alternative ways of specifying document rules. It has always been possible to programmatically examine conditions in XML documents, but the ability to impose the more rigid standard that, "a document not meeting a set of formal rules is *invalid*," essentially, is often preferable. W3C XML Schemas are one major answer to these calls (but not the only schema option out there). Steven Holzner, in *Inside XML* has a characterization of XML schemas that is worth repeating:

> Over time, many people have complained to the W3C about the complexity of DTDs and have asked for something simpler. W3C listened, assigned a committee to work on the problem, and came up with a solution that is much more complex than DTDs ever were (p.199).

Holzner continues -- and most all XML programmers will agree (myself included) -- that despite their complexity, W3C XML Schemas provide a lot of important capabilities and are worth using for many classes of validation rules.

At least two fundamental and conceptual wrinkles remain for any "schemas everywhere" goal. The first issue is that the *W3C XML Schema Candidate Recommendation*, which just ended its review period on December 15, 2000, does not include any provision for entities; by extension, this includes parametric entities. The second issue is that despite their enhanced expressiveness, there are still many document rules that you cannot express in XML schemas (some proposals offer to utilize XSLT to enhance validation expressiveness, but other means are also possible and in use). In other words, schemas cannot quite do everything DTDs have long been able to, while on the other hand, schemas also cannot express a whole set of further rules one might wish to impose on documents. At a more pragmatic level, tools for working with XML schemas are less mature than those for working with DTDs (especially regarding validation, which is the core issue).

The whole state of XML document validation rules remains messy. Unfortunately, I am not able to prognosticate how everything will eventually shake out. (For a summary of when DTDs probably make sense to use, see the sidebar "When to use DTDs.") In the meantime, let's look at some specifics of what DTDs and XML schemas are capable of expressing.

**Rich typing**
The place where W3C XML Schemas really shine is in expressing type constraints on attribute values and element contents. This is where DTDs are weakest. Beyond providing an extremely rich set of built-in simpleTypes, XML schemas allow you to derive new simpleTypes using a regular-expression-like syntax. The built-ins include those you would expect if you have worked with programming languages: string, int, float, unsignedLong, byte, and so on; but they also include some types that most programming languages lack natively: timeInstant (that is, date/time), recurringDate (day-of-year), uriReference, language, nonNegativeInteger.

For example, in a DTD one might have a declaration such as the one in Listing 1:

**Listing 1: DTD "item" element definition**

```
<!ELEMENT item (prodName+,USPrice,shipDate?)
<!ATTLIST item partNum CDATA>
<!ELEMENT prodName (#PCDATA)>
<!ELEMENT USPrice (#PCDATA)>
<!ELEMENT shipDate (#PCDATA)>
```

In W3C XML Schema, one can be more specific (modified slightly from the W3C Schema primer):

**Listing 2: XML schema "item" element definition**

```
<xsd:element name="item">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="prodName" type="xsd:string" maxOccurs="5"/>
         <xsd:element name="USPrice"  type="xsd:decimal"/>
         <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="partNum" type="SKU"/>
   </xsd:complexType>
</xsd:element>

<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Two striking, if superficial, features stand out in these element definitions. One is that the schema is itself a well-formed XML instance with its tags using the xsd namespace (actually, so is the DTD, but it has only processing instructions, no content as such); the second (and consequence of the first) is that the schema is far more verbose than the DTD.

Beyond the syntactic niceties, you can see that the schema example does several things that are impossible with DTDs. The type of prodName is basically the same between the definitions, but the specifications of USPrice and shipDate in the schema are as types decimal and date. As a text file, an XML instance with these elements contains some ASCII (or Unicode) characters inside the elements; however, a validator that is schema-aware can demand much more specific formatting of the characters inside decimal and date elements (and likewise other types). Much more interesting is the attribute partNum, which is of a derived specialized type. The type SKU is not a built-in type, but rather a sequence of characters following the given pattern in the "SKU" declaration (specifically, it must have three digits: a dash, and two capital letters, in that order). It is also possible to use SKU for an element type; it is just a coincidence that it defines an attribute in this case.

In the DTD version of the element definition, all these interesting (and potentially rather complicated, if specialized) types must simply get called PCDATA, with no further say as to what that character data looks like (CDATA in the case of attributes).

**When to use DTDs**
DTDs are still your best choice when:

- A compact representation of your document rules is important to you.
- You want downstream users to be able to override and specialize types via parametric internal sets.
- Your document rules primarily concern nesting of elements, not semantic constraints on contents (as in prose markup).
- The tools you are used to using support DTD better than schemas.

In richly typing element/attribute values, schemas shade subtly from describing the syntax of an XML instance to describing its semantics. Parsing purists might take issue with my characterization: "built-in schema types are defined syntactically, and patterns built on those built-in are thusly also formally syntactic." But in practical terms, when you declare that a given element must be a date, what you really want is, well, for the element to contain a date. Expressing semantic information is not a bad thing, of course, but one might argue that it is better to confine that to an application level as such, rather than a format declaration. After all, there are semantic features -- even simple ones -- that elude schemas but might be just as important in an application as what schemas express. For example, sure a "stock-keeping unit" must look like "999-AA"; but maybe you also ship out widgets only in baker's dozens. Divisibility on an integer by 13 is not expressible in XML schemas (and therefore you still can't give *widgetquantity* the needed constraints at that level). The point here is that even with the extra capabilities of schemas (over DTDs), one still might need to do post-validation at an application level to determine if an XML document is *functionally* valid.

## Occurrence constraints

As well as powerful type declaration, XML schemas improve upon the DTD's ability to declare the cardinality of subelement patterns. However, DTDs have always had a more clumsy way of expressing every occurrence constraint (cardinality) than XML schemas.

In DTDs, one of the symbols: ?, *, and +, which specify, respectively, "zero or one," "zero or more," "one or more," quantifies cardinality. That is, except for the question mark's ability to say: "it is there or it isn't," nothing in the DTD syntax seems to limit the number of occurrences of a given pattern (whether a single subtag, or a nested sequence of them). So expressing the 1-5 occurrences of prodName in the above example schema seems to be a problem. Likewise, without having the XML schema attribute minOccurs, we seem unable to express the requirement that something occurs some specific number of times (other than "at least once"). Actually, DTDs' minimum quantifiers are good enough, if inelegant at times. The following constraints are equivalent:

**Listing 3: XML schema syntax for "seven to twelve" donuts**

```
<xsd:element name="donutorder">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="donut" type="xsd:string"
                      minOccurs="7" maxOccurs="12" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>



<!ELEMENT donut (#PCDATA)>
<!ELEMENT donutorder
          (donut,donut,donut,donut,donut,donut,donut,
           donut?,donut?,donut?,donut?,donut?)
```

Of course, if you get orders by the gross, DTDs start to look *really* inelegant!

**Enumeration**
Both DTDs and W3C XML Schemas allow the use of enumerated types in attributes, but schemas are a great improvement in also allowing enumerated types in element contents. The lack of those, in my opinion, is a genuine shortcoming of DTDs. Furthermore, the Schema approach to enumeration is general and elegant. A specialized simpleType can contain an enumeration *facet.* Such a simpleType is automatically suitable for either an attribute or element value type.

Let us illustrate each syntax:

**Listing 4: XML schema syntax for enumerated attribute**

```
<xsd:simpleType name="shoe_color">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="red"/>
      <xsd:enumeration value="green"/>
      <xsd:enumeration value="blue"/>
      <xsd:enumeration value="yellow"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="person" type="person_type">
   <xsd:attribute name="shoes" type="shoe_color"/>
</xsd:element>



<!ATTLIST person shoes (red | green | blue | yellow)>
```

The DTD attribute declaration appears just as good (maybe better in its conciseness), but if your model puts shoe_color in an element content instead, the DTD falls flat:

**Listing 5: XML schema syntax for enumerated element**

```
<xsd:element name="shoes" type="shoe_color">
```

**Whither**
W3C XML Schemas let XML programmers express a new set of declarative constraints on documents for which DTDs are insufficient. For many programmers, the use of XML instance syntax in schemas also brings a greater measure of consistency to different parts of XML work (others disagree, of course). Schemas are certainly destined to grow in significance and scope as they become more familiar, and as developers enhance more tools to work with them.

One way to get a jump start on schema work is to automate the conversion of existing DTDs to XML schema format. Obviously, automated conversions cannot add the new expressive capabilities of XML schemas themselves; but automation can create good templates from which to specify the specific typing constraints one wishes to impose. The Resources section provides two links to automated DTD-to-schema conversion tools.

**Resources**

- The *W3C Candidate Recommendation 24 October 2000* is the basic standard for W3C XML Schemas.
- The *Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000* can be found at w3.org/TR/REC-xml. "The second edition is not a new version of XML (first published 10 February 1998); it merely incorporates the changes dictated by the first-edition errata."
- To keep matters sufficiently complicated, the W3C's XML Schema is not the only schema options out there. *RELAX* (Regular Expression Language for XML) is now ISO/IEC DIS (Draft International Standard) 22250-1. This standard is most widely used in Japan, but it is not language or culture specific. A good starting place is xml.gr.jp/relax/.
- *The XML Schema Specification in Context* is a nice compact summary of the comparative capabilities of W3C XML Schema (compared to a number of other descriptive formats).
- Check out Yuichi Koike's Conversion Tool from DTDs to XML Schema. (It requires Perl.)
- A nice thick, informative -- but perhaps somewhat rambling -- introduction to most all matters XML is *Inside XML*, Steven Holzner, New Riders, 2001 (ISBN 0-7357-1020-1). This column excerpts a particular pithy and humorous sentence.

**About the author**

David Mertz, in his gnomist aspirations, wishes he had coined the observation that the great thing about standards is that there are so many to choose from. But then, he is also fuzzy on OS design. David may be reached at mertz@gnosis.cx; his life pored over at gnosis.cx/publish/. Suggestions and recommendations on this, past, or future columns are welcome.

e-mail it!

**What do you think of this article?**

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

**Comments?**

Submit feedback