Principles of XML design: When the order of XML elements matters

When to be strict and when to be lax as you decide how to order child elements

Uche Ogbuji

When multiple XML elements occur within another element, does element order matter? Whether it's the order in which the parser reports elements to applications, or the question of whether or not to mandate specific order in schema patterns, things are not always as simple as they may seem. In this article, Uche Ogbuji covers design and processing considerations related to the order of XML elements.

View more content in this series

Throughout this series, Principles of XML design, I have shown how to name and organize XML elements. A subtle but important consideration I haven't yet covered is whether or not to assign significance to the order of children of XML elements. For example, do you see the documents in Listing 1 and Listing 2 as the same?

Listing 1. An example XML document

```xml
<?xml version='1.0' encoding='utf-8'?>
<memo>
  <title>
    With Usura Hath no Man a House of Good Stone
  </title>
  <date>2005-04-15</date>
  <from>Ezra Pound</from>
  <to>Employees</to>
  <body>
    It appears the art world requires a reminder of the fact that the best art is created for the enjoyment of the first buyer, and not as mere investment. As I've said before, none of the work of Duccio, Piero Della Francesca, Pietro Lombardo, Fra Angelico, Zuan Bellini or such others would have been of any value if guided by usurious motives.
  </body>
</memo>
```
Listing 2. An example XML document with different element order

```xml
<?xml version='1.0' encoding='utf-8'?>
<memo>
  <date>2005-04-15</date>
  <to>Employees</to>
  <title>With Usura Hath no Man a House of Good Stone</title>
  <body>It appears the art world requires a reminder of the fact that the best art is created for the enjoyment of the first buyer, and not as mere investment. As I've said before, none of the work of Duccio, Piero Della Francesca, Pietro Lombardo, Fra Angelico, Zuan Bellini or such others would have been of any value if guided by usurious motives.
  </body>
  <from>Ezra Pound</from>
</memo>
```

The only difference between these documents is the order in which the children of the `memo` element appear. All these elements are collectively siblings, and this article is entirely concerned with the significance of the order of sibling elements. Notice that some of the discussion will also be pertinent to cases where you have text, comments, and processing instructions as siblings of elements, but this discussion focuses solely on elements.

**Lawyer among the specifications**

The first thing to be aware of, and which might surprise you, is that the XML 1.0 specification itself does not guarantee element order in the sections on well-formedness (the sections on validity are more relevant to the discussion later in this article). The XML 1.0 well-formedness definition specifically states that attributes are unordered, but says nothing about elements. This means that technically speaking, a conforming XML parser might decide to report the child elements of `memo` in Listing 1 in any order. You might expect them to be reported in the order they appear in the actual XML text (in this case, the same as what is called document order):

1. title
2. date
3. from
4. to
5. body

But an XML parser is actually free to report them in alphabetical order:

1. body
2. date
3. from
4. title
5. to

I know of no XML parser that does not report sibling elements in document order, just for the practical reason that it’s easiest and most efficient to report parts of the XML document as they are encountered while parsing. But it’s good for you to be aware of the possibility of such odd
arrangement. I use the term **parse order** for the order in which elements are reported by a parser. As you'll see, there is at least one other important aspect to element order.

Having said all the above, I admit that almost no one uses XML 1.0 in complete isolation. People usually work with technologies that build on XML; most of these technologies do specify some ordering rules for elements, and the order imposed is almost universally document order. The XML Information Set (InfoSet -- see Related topics), the core XML data model defined by the W3C, characterizes element children as:

> An **ordered list of child information items**, in **document order**. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items, one for each element, processing instruction, reference to an unprocessed external entity, data character, and comment appearing immediately within the current element. If the element is empty, this list has no members.

I've emphasized the key portion in boldface. Many general-purpose XML processing specifications, such as Canonical XML, derive from the InfoSet and thus inherit this rule for sibling order. Others, such as XPath (and thus XSLT) and DOM, define their own data models with similar rules for siblings.

**Schema constraints of element order**

When designing an XML vocabulary, you can be more precise about rules for the sibling order that is permitted in valid documents. For example, if you wrote a RELAX NG schema for the memo document in Listing 1, you could use a pattern such as that in Listing 3 (in RELAX NG compact syntax). The commas between the sibling element subpatterns indicate that they are required to appear in the given order.

**Listing 3. RELAX NG pattern for an element with ordered children**

```xml
<element memo {
    <element title { text },
    <element date { text },
    <element from { text },
    <element to { text },
    <element body { text }
}
```

Listing 1 is valid against this schema, but Listing 2 is not.

Listing 4 is a similar pattern, but without mandating any order. The ampersand (&) characters between the sibling element subpatterns indicate that any order is acceptable.

**Listing 4. RELAX NG pattern for an element with children that aren't ordered**

```xml
<element memo {
    <element title { text } &
    <element date { text } &
    <element from { text } &
    <element to { text } &
    <element body { text }
```
Both Listing 1 and Listing 2 are valid against this schema.

**Decisions | decisions**

The question is: When do you use the commas, and when do you use the pipes? I call this aspect of ordering schema order, which can either be ordered or unordered. My main rule of thumb for element schema order is: **Use ordered patterns unless you have specific reason not to.** The reasons for this prescription are actually a bit philosophical, but they come from experience in XML design, and observing the effects of both cases. In the end, I think it's well proven that it's better not to give users and downstream systems unnecessary choices. If you don't set an order, then they generally have to come up with one, and that opens up some room for confusion.

One problem with this position is that it runs a little bit afoul of Postel's Law -- "Be conservative in what you do, be liberal in what you accept from others" -- which suggests that you should have guidelines for the order you use in patterns in documents you control, but that you should not be too eager to reject documents that use different ordering. Respect for this principle might be one reason not to follow my prescription above, especially if most of the documents you're dealing with will not be created or modified by people or systems under your control.

**Information value of order**

An important distinction to remember is that if you choose ordered patterns, then the parse order does not provide any useful information, whereas if you choose unordered patterns, the parse order may provide useful information. As an example, if you use the pattern in Listing 3, you always know what order the elements will appear in valid documents, whereas if you use the pattern in Listing 4, the order can be used to tell the application something about the elements.

Suppose you have an application that stores many memo documents using the pattern in Listing 4, and it includes a search engine for them. The search engine application might return result documents so that the field that the user searched upon always appears first. So if the user searched for all documents with "Usura" in the title, then one of the results would be a document like that in Listing 1 (where the title element is the first sibling); and if the user searched for all documents dated "2005-04-15", then one of the results would be a document like that in Listing 2 (where the date element is the first sibling). Both represent the same memo instance in this application, but the ordering of the elements now conveys something meaningful about the document, specifically what form of search criteria was used to retrieve it. The element order thus becomes useful metadata. If you think you will make use of such conventions, you will want to use unordered patterns.

**Processing considerations (and documents versus data)**

Some uses of XML are more connected to database management than to documents and prose. This is sometimes called records-oriented XML. In such XML, using ordered patterns everywhere can be a problem. For example, if you manage data in hash tables or other unordered data structures in the application domain, you might face additional work re-assembling elements into
the order set by a schema. In records-oriented XML, you should probably use unordered patterns unless you have specific reason for ordering them (for example, when a specific order is already specified in the application domain model). Listing 5 is an example of records-oriented XML.

**Listing 5. An example of records-oriented XML**

```xml
<label>
  <occupation>Poet</occupation>
  <name>Ezra Pound</name>
  <address>
    <street>45 Usura Place</street>
    <city>Hailey</city>
    <state>ID</state>
  </address>
</label>
```

In this example, you don't need relative ordering between the *occupation*, *name*, and *address* elements. But, if you define a strict order, processing software will have to keep track of this. For example, you couldn't just extract the information from relational data in the usual arbitrary order and place it directly into output documents. You would have to build schema order information into the application, which is otherwise unnecessary. However, the application might itself define a meaningful ordering between *street*, *city*, and *state*, so you cause no additional interference if you mandate this order in the schema.

On the other hand, if you use W3C XML Schema (WXS), you will probably come across several types of unordered patterns that you cannot express due to language limitations. Most of these limitations do not apply to RELAX NG, but if your schema language of choice is WXS, you might find that requiring order by default is the easiest way to ensure WXS friendliness, whether your vocabulary is records-oriented or not.

**Wrap-up**

If you have been following this series, you've probably realized that in reality few design considerations are trivial in XML. Before you decide whether the order of information is significant in your schemata, and how your applications will process valid instance documents, consider the nature of the XML and the implications of either decision. Such implications can be subtle, but they can have surprisingly far-reaching effects.
Related topics

- Don't miss the earlier articles in this series on XML design:
  - "When to use elements versus attributes" (developerWorks, March 2004)
  - "Use XML namespaces with care" (developerWorks, April 2004)
  - "Element structures for names and addresses" (developerWorks, August 2004)
  - "Considering container elements" (developerWorks, September 2004)
- Jon Postel was an extraordinary contributor to the building of the Internet. Check out his famous "law" in IETF RFC 793.
- Visit the XML-DEV mailing list, the site of many discussions on the significance of XML ordering. One of the most useful was started and summarized by Eric van der Vlist.
- Familiarize yourself with the core XML data model, XML Information Set (Second Edition) (W3C Recommendation, February 2004). In this article, the author quotes from the section defining the information model for elements.
- Find more XML resources on the developerWorks XML zone, including Uche Ogbuji's Thinking XML column.
- Find out how you can become an IBM Certified Developer.