

Principles of XML design

When to use elements versus attributes

Exploring the oldest question in XML design



Uche Ogbuji

Published on March 04, 2004



Several frequently pondered questions of DTD design in SGML have followed the legacy to its offshoot, XML. Regardless of what XML schema language you use, you might find yourself asking:

- When do I use elements and when do I use attributes for presenting bits of information?
- When do I require an order for elements, and when do I just allow arbitrary order?
- When do I use wrapper elements around sequences of similar elements?

In my experience working with users of XML, the first question is by far the most common. In some cases the answer is pretty unambiguous:

- If the information in question could be itself marked up with elements, put it in an element.
- If the information is suitable for attribute form, but could

end up as multiple attributes of the same name on the same element, use child elements instead.

- If the information is required to be in a standard DTD-like attribute type such as `ID`, `IDREF`, or `ENTITY`, use an attribute.
- If the information should not be normalized for white space, use elements. (XML processors normalize attributes in ways that can change the raw text of the attribute value.)

Unfortunately, design decisions don't often fall into such neat categories. The question remains how to make the right choice in the gray areas. The usual answer is *No single answer is right, use your best judgment*. But this is not very helpful for those trying to find their feet with XML. True, even experts do not always agree on what to do in certain situations, but this is no reason not to offer basic guidelines for choosing between XML and attributes.

First I want to comment on two guidelines that I have heard and *do not* recommend. I have heard *Just make everything an element*. The reasons given range from *Attributes just complicate things* to *Attributes can stunt extensibility*. But if you do not use attributes, you are leaving out a very important aspect of XML's power, and you're probably better off using some delimited text format.

I have also heard *If it is the sort of material you would expect to display in a browser, use element content*. The problem with this guideline is that it encourages people to think of XML content design in terms of presentation, two considerations that should not be mixed. I present a very similar guideline in this article, but I express it in terms of the intent of the content, rather than in terms of presentation.

In the rest of this article, I present a set of guidelines that I do recommend when choosing between elements and attributes.

Recommended guidelines

I have divided these guidelines into a set of principles that I think frame the choice between elements and attributes overall. None of the guidelines are meant to be absolute; use them as rules of thumb and feel free to break the rules whenever your particular needs require it.

Principle of core content

If you consider the information in question to be part of the essential material that is being expressed or communicated in the XML, put it in an element. For human-readable documents this generally means the core content that is being communicated to the reader. For machine-oriented records formats this generally means the data that comes directly from the problem domain. If you consider the information to be peripheral or incidental to the main communication, or purely intended to help applications process the main communication, use attributes. This avoids cluttering up the core content with auxiliary material. For machine-oriented records formats, this generally means application-specific notations on the main data from the problem-domain.

As an example, I have seen many XML formats, usually home-grown in businesses, where document titles were placed in an attribute. I think a title is such a fundamental part of the communication of a document that it should always be in element content. On the other hand, I have often seen cases where internal product identifiers were thrown as elements into descriptive records of the product. In some of these cases, attributes were more appropriate because the specific internal product code would not be of primary interest to most readers or processors of the document, especially when the ID was of a very long or inscrutable format.

You might have heard the principle data goes in elements, metadata in attributes. The above two paragraphs really express the same principle, but in more deliberate and less fuzzy language.

Principle of structured information

If the information is expressed in a structured form, especially if the structure may be extensible, use elements. On the other hand: If the information is expressed as an atomic token, use attributes. Elements are the extensible engine for expressing structure in XML. Almost all XML processing tools are designed around this fact, and if you break down structured information properly into elements, you'll find that your processing tools complement your design, and that you thereby gain productivity and maintainability. Attributes are designed for expressing simple properties of the information represented in an element. If you work against the basic architecture of XML by shoehorning structured information into attributes

Famous exceptions

I've been careful to point out that my design guidelines are flexible and that you will certainly find reason to make exceptions. You'll also find the need to balance the impetus from different principles. In many examples of standard XML vocabularies, conscious design decisions were made that contradict principles I post in this article. Here is a sampling.

- SVG vector paths are expressed as a series of space-delimited numbers in a single attribute, which contradicts the *Principle of structured information*. This decision was made because SVG had an urgent need for terseness, but it has been criticized because of how it can complicate the processing of SVG formats.
- Many formats that allow Cascading Stylesheet (CSS) styles in attributes (such as XHTML 1.0 and XSL-FO) allow you to create complex descriptions of style, mixing multiple facets and units of measure. This contradicts the *Principle of structured information*. Interestingly, this is no longer a concern in XHTML 2.0 because with that version you can only specify a class token that is mapped to a separate CSS. You no longer have in-line style attributes.

you may gain some specious terseness and convenience, but you will probably pay in maintenance costs.

Dates are a good example: A date has fixed structure and generally acts as a single token, so it makes sense as an attribute (preferably expressed in ISO-8601). Representing personal names, on the other hand, is a case where I've seen this principle surprise designers. I see names in attributes a lot, but I have always argued that personal names should be in element content. A personal name has surprisingly variable structure (in some cultures you can cause confusion or offense by omitting honorifics or assuming an order of parts of names). A personal name is also rarely an atomic token. As an example, sometimes you may want to search or sort by a forename and sometimes by a surname. I should point out that it is just as problematic to shoehorn a full name into the content of a single element as it is to put it in an attribute. Thus:

```
<customer>
  <name>Gabriel Okara</name>
  <occupation>Poet</occupation>
</customer>
```

is not much better than:

```
<customer name="Gabriel Okara">
  <occupation>Poet</occupation>
</customer>
```

I hope to expand on the treatment of people's names in markup in a future article.

Principle of readability

If the information is intended to be read and understood by a person, use elements. In general this guideline places prose in element content. If the information is most readily understood and digested by a machine, use attributes. In general this guideline means that information tokens that are not natural language go in attributes.

In some cases, people can decipher the information being

represented but need a machine to use it properly. URLs are a great example: People have learned to read URLs through exposure in Web browsers and e-mail messages, but a URL is usually not much use without the computer to retrieve the referenced resource. Some database identifiers are also quite readable (although established database management best practice discourages using IDs that could have business meaning), but such IDs are usually props for machine processing. For these reasons I recommend putting URLs and IDs in attributes.

Principle of element/attribute binding

Use an element if you need its value to be modified by another attribute. XML establishes a very strong conceptual bond between an attribute and the element in which it appears. An attribute provides some property or modification of that particular element. Processing tools for XML tend to follow this concept and it is almost always a terrible idea to have one attribute modify another. For example, if you are designing a format for a restaurant menu and you include the portion sizes of items on the menu, you may decide that this is not really important to the typical reader of the menu format so you apply the *Principle of core content* and make it an attribute. The first attempt is:

```
<menu>
  <menu-item portion="250 mL">
    <name>Small soft drink</name>
  </menu-item>
  <menu-item portion="500 g">
    <name>Sirloin steak</name>
  </menu-item>
</menu>
```

Following the *Principle of structured information* you decide not to shoehorn the portion measurement and units into a single attribute, but instead of using an element, you opt for:

```
<menu>
  <menu-item portion-size="250" portion-
unit="mL">
    <name>Small soft drink</name>
  </menu-item>
  <menu-item portion-size="500" portion-
unit="g">
```

```
    <name>Sirloin steak</name>
  </menu-item>
</menu>
```

The attribute `portion-unit` now modifies `portion-size`, which as I've mentioned is a bad idea. An attribute on the element `menu-item` should modify that element, and nothing else. The solution is to give in and use an element:

```
<menu>
  <menu-item>
    <portion unit="mL">250</portion>
    <name>Small soft drink</name>
  </menu-item>
  <menu-item>
    <portion unit="g">500</portion>
    <name>Sirloin steak</name>
  </menu-item>
</menu>
```

In this case I applied a mix of the *Principle of core content* and the *Principle of readability* to the decision to put the value in content and the units in an attribute. This is one of those cases that are less cut and dried, and other schemes might be as suitable as mine. The solution also involves contradicting the original decision to put the portion size into an attribute based on the *Principle of core content*. This illustrates that sometimes the principles will lead to conflicting conclusions where you'll have to use your own judgment to decide on each specific matter.

No substitute for study and experience

XML design is a matter for professionals, and if you want to gain value from XML you should be willing to study XML design principles. Many developers accept that programming code benefits from careful design, but in the case of XML they decide it's OK to just do what seems to work. This is a distinction that I have seen lead to real and painful costs down the road. All it takes for you to learn sound XML design is to pay attention to the issues. Examine standard XML vocabularies designed by experts. Take note of your own design decisions and gauge the positive and negative effect each has had on later developments. As you gain

experience, your instinct will become the most important tool in making design decisions, and the care you take will pay certain rewards if you find yourself using XML to any significant extent.

I plan to continue covering XML design principles in future articles such as this one. I'll also continue to touch briefly on issues of XML design in my *Thinking XML* column.

Downloadable resources



Related topics

- Check out the famous Cover Pages, which offer several compilations of information on the topic aggregated on these pages: [SGML/XML: Elements versus attributes, April 1998](#) and [SGML/XML: Using Elements and Attributes](#).
- Read *Effective XML* by Elliotte Rusty Harold (Addison-Wesley, 2003). In my experience, the author has an excellent command of XML design and best practices. You can read some of the sections online, including "[Store metadata in attributes](#)" and "[Make structure explicit through markup](#)", both of which are directly relevant to this article.
- Don't miss the any of articles in this series on XML design:
 - "[Use XML namespaces with care](#)" (*developerWorks*, April 2004).
 - "[Element structures for names and addresses](#)" (*developerWorks*, August 2004).
- [IBM trial software](#): Build your next development project with trial software available for download directly from developerWorks.

Comments

☐ Subscribe me to comment notifications

[Sign in](#) or [register](#) to add and subscribe to comments.

developerWorks

About

Help

Submit content

RFE

Community

Report abuse

Third-party
notice



Join

Faculty

Students

Business Partners

Select a language

English

中文

日本語

Русский

Português (Brasil)

Español

한글



Events



Newsletters



dW

Premium



dW TV



dW

Answers



Downloads



Feeds



dW Blog



dW

Courses

[Contact](#)

[Privacy](#)

[Terms of use](#)

[Accessibility](#)

[Feedback](#)

[Cookie Preferences](#)