SPOTLIGHT

Extending Your Markup:
An XML Tutorial

André Bergholz • *Stanford University*

XML introduces a family of languages to provide a more semantic management of information than HTML.

By now, no doubt, you've heard the acronym XML. You've probably also heard that XML (a) is simple and (b) will solve all your problems. Sounds like magic, doesn't it? But then you look a little deeper and encounter more three-letter acronyms, like DTD, XSL, RDF, and DOM. You begin to doubt XML's simplicity (and you never believed XML could solve all your problems in the first place). In this short tutorial I present what I think are the essential concepts of XML, and hopefully will convince you that despite the hype, XML is important for presentation, exchange, and management of information.

More Meaningful Markup

The Standard Generalized Markup Language (SGML) is a rather complicated language that lets you define structure for documents. The Hypertext Markup Language is an application of SGML that has a fixed set of markups. HTML is primarily used for layout on the Web. It tells nothing about the content of the data.

Both SGML and HTML heavily influenced the development of the Extensible Markup Language (XML), a semantic language that lets you meaningfully annotate text. Meaningful annotation is, in essence, what XML is all about. Figure 1a shows a bibliography entry in HTML; Figure 1b shows that same entry written in XML. Because the structure of the information in Figure 1b is more explicit, it is much easier for humans to read and computers to process.

XML Syntax

Syntactically, XML documents look like HTML documents. A *well-formed* XML document—one that conforms to the XML syntax—starts with a *prolog* and contains exactly one *element*. Additionally, an arbitrary number of comments and processing instructions (which are not explained in this tutorial) can be included. The prolog looks something like this:

```
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
```

It tells you that your document follows XML version 1.0, is stand-alone (that is, not accompanied by a document type definition, or DTD), and uses Unicode Transformation Format 8-bit (UTF-8) encoding. If the document was accompanied by a DTD, the DTD declaration would also be part of the prolog.

The single element can be viewed as the root of the document. Elements can be nested, and attributes can be attached to them. Attribute values must be in quotes, and tags must be balanced. Empty element tags must either end with a /> or be explicitly closed.

Figure 2 expands on Figure 1b, adding attributes and a comment. The id attribute assigns an identifier to an element, and the idref attribute reuses the identifier.

For a complete description of the XML syntax, see the W3C XML Recommendation (http://www.w3.org/TR/1998/REC-xml-19980210).

Defining Structure

DTDs, which are also used in SGML, define the structure of XML documents. It's easiest to think of a DTD as a context-free grammar. In particular, DTDs let users specify the set of tags, the order of tags, and the attributes associated with each. A well-formed XML document that conforms to its DTD is called *valid*. Figure 3 shows a simple DTD for the bibliography example in Figure 2.

A DTD is declared in the XML document's prolog using the !DOCTYPE tag. The DTD can be included within the XML document, or it can be contained in a separate file. If the DTD is in a separate file, say document.dtd, the XML document includes the statement:

```
<!DOCTYPE Document SYSTEM "document.dtd">
```

You can also refer to an external DTD through a URI.

 1089-7801/00/$10.00 ©2000 IEEE

DTD elements

Elements can be either *nonterminal* or *terminal*. Nonterminal elements (BIB and BOOK in Figure 3) contain subelements, which can be grouped as *sequences* or *choices*. A sequence defines the order in which subelements must appear. A choice gives a list of alternatives for subelements. Sequences and choices can contain each other.

In Figure 3, both BIB and BOOK are sequences. A BOOK element has to have at least one AUTHOR (indicated by +), followed by a TITLE and, optionally, by a PUBLISHER and a YEAR (indicated by ?). A choice is indicated by the logical operator |. The example below shows a choice in which a SECTION can either be a TITLE followed by at least one PARAGRAPH, or a TITLE followed by zero or more PARAGRAPHs (indicated by the wildcard *) and at least one SUBSECTION.

```
<!ELEMENT SECTION ((TITLE, (PARAGRAPH+)) |
   (TITLE, (PARAGRAPH*), (SUBSECTION+)))>
```

Terminal elements can be declared as parsed character data (#PCDATA, as in Figure 3) or as EMPTY. Elements can also be declared as ANY. An element declared as ANY is a terminal element in the grammar, but it can contain subelements of any declared type, as well as character data.

DTD attributes

Elements can have zero or more attributes, which are declared using the !ATTLIST tag. Unlike element definitions, attribute definitions do not impose order on when the attributes occur. Furthermore, if several attributes are declared for the same element type, they can be declared using multiple !ATTLIST tags. Attributes can be optional (#IMPLIED), required (#REQUIRED), or fixed (#FIXED). Optional attributes can, and fixed attributes must, have a default value:

```
<!ATTLIST PROJECT url CDATA
"http://www.myserver.net/index.html">
```

Attributes can have different data types. Character data (CDATA) is the most common. With types id, idref, and idrefs, element identifiers can be declared and referenced. Every value of a referencing attribute must be a value of another attribute of type id. Attributes can also be enumerated as user-defined types:

```
<!ATTLIST PUBLICATION format (html | pdf | ps)
#REQUIRED>
```

XML Examples (Figures 1–3)

```
<UL>
<LI>Aho, A. V., Sethi, R., Ullman, J. D.: <EM>Compilers: Principles,
Techniques, and Tools </EM>, Addison-Wesley, 1985
</UL>
```

(a)

```
<BOOK>
   <AUTHOR> Aho, A. V. </AUTHOR>
   <AUTHOR> Sethi, R. </AUTHOR>
   <AUTHOR> Ullman, J. D. </AUTHOR>
   <TITLE> Compilers: Principles, Techniques, and Tools </TITLE>
   <PUBLISHER> Addison-Wesley </PUBLISHER>
   <YEAR> 1985 </YEAR>
</BOOK>
```

(b)

Figure 1. A bibliography entry (a) in HTML and (b) in XML. The HTML description is layout oriented, while the XML description is structure oriented.

```
<?xml version="1.0" standalone="yes">

<!– This is an example bibliography. –>
<BIB>
   <BOOK nickname="Dragon book">
      <AUTHOR id="aho"> Aho, A. V. </AUTHOR>
      <AUTHOR id="sethi"> Sethi, R. </AUTHOR>
      <AUTHOR id="ullman"> Ullman, J. D. </AUTHOR>
      <TITLE> Compilers: Principles, Techniques, and Tools </TITLE>
      <PUBLISHER> Addison-Wesley </PUBLISHER>
      <YEAR> 1985 </YEAR>
   </BOOK>

   <BOOK>
      <AUTHOR idref="ullman"/>
      <TITLE> Principles of Database and Knowledge-Base Systems, Vol.
1 </TITLE>
   </BOOK>
...
</BIB>
```

Figure 2. A short bibliography in XML. This example demonstrates the use of attributes as well as referencing using the id and idref attributes.

```
<!DOCTYPE bib [
   <!ELEMENT BIB (BOOK+)>
   <!ELEMENT BOOK (AUTHOR+, TITLE, PUBLISHER?, YEAR?)>
   <!ATTLIST BOOK
      isbn CDATA #IMPLIED
      nickname CDATA #IMPLIED>
   <!ELEMENT AUTHOR (#PCDATA)>
   <!ATTLIST AUTHOR
      id ID #IMPLIED
      idref IDREF #IMPLIED>
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT PUBLISHER (#PCDATA)>
   <!ELEMENT YEAR (#PCDATA)>
]>
```

Figure 3. A DTD for the bibliography example. The DTD defines a grammar for documents.

XML Resources on the Web

A lot of material on XML is available online. Of course, your first stop should always be the W3C XML page (http://w3c.org/XML/). The XML-FAQ page (http://www.ucc.ie/xml/) provides a simple introduction, and the XML Cover Pages (http://www.oasis-open.org/cover/sgml-xml.html) offer a comprehensive online reference on every topic covered in this tutorial.

Anders Møller and Michael Schwartzbach have compiled an excellent XML tutorial (http://www.brics.dk/~amoeller/XML/). Ron Bourret has a collection of insightful pages, most notably on DTDs, namespaces, and XML Schema (http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/bourret.htm).

Three (interesting) sections of the XML Bible[1] can be found at http://metalab.unc.edu/xml/books/bible/, while a somewhat severe criticism of the XML hype can be found at http://www.interlog.com/ gray/markup-abuse.html. The xml-dev mailing list (http://xml.org/xml-dev/) keeps you up-to-date on current developments and lets you share your own ideas ("Ask not what XML can do for you, ask what you can do for XML!").

Of course, software that supports XML is essential. Because existing standards continue to develop and new ones constantly arise, many tools support only parts of some specifications. An excellent overview of existing systems and tools can be found at http://xmlsoftware.com.

Reference

1. E.R. Harold, *XML Bible*, IDG Book Worldwide, Foster City, Calif., 1999.

Popular applications of DTDs include XHTML (http://www.w3.org/TR/xhtml1/) and the Chemical Markup Language (CML, http://www.xmlcml.org/).

Extending Capabilities

Extensions to XML include namespaces as well as more powerful addressing and linking abilities. These extensions let you link more than one source document to more than one target document. Furthermore, with XML it is not necessary to place an anchor in the target document.

Namespaces

Using namespaces avoids name clashes (that is, situations where the same tag name is used in different contexts). For instance, a namespace can identify whether an address is a postal address, an e-mail address, or an IP address. Tag names within a namespace must be unique. In an XML document, namespaces are declared using the xmlns attribute.

```
<BIB xmlns:mybib="http://www.myserver.net/">
```

This namespace can be referred to using the prefix mybib. That means that mybib:AUTHOR refers to the AUTHOR in the namespace. The URI http://www.myserver.net/ identifies the namespace. The URI is purely an identifier—it might not point to anything. The prefix definition :mybib can also be skipped, in which case the namespace identified by http://www.myserver.net/ becomes the default namespace for the document.

Namespaces can be defined in any element. Their scope is the element in which they are defined, plus all its descendants. To avoid confusion, you should define all namespaces within the root element and use unique prefixes.

Unfortunately, namespaces and DTDs do not work well together. In a DTD, the element definition <!ELEMENT mybib:BIB...> is just that: an element mybib:BIB is defined. It does not mean that an element BIB is defined within the namespace that the prefix mybib is mapped to. It does not even mean that the colon separates a prefix from a local name. Thus, depending on your application, you must make your document both valid with respect to some DTD and conform to the XML namespaces recommendation.

Addressing and linking

In HTML, URLs point only to a document; it is not possible, for example, to address the fifth item in the second list in a document directly. Moreover, HTML links are one-way, and external link definitions are not possible. XML extends HTML's linking capabilities with three supporting languages.

- *Xlink* (http://www.w3.org/TR/xlink/), which describes how two documents can be linked;
- *XPointer*, which enables addressing individual parts of an XML document; and
- *XPath,* which is used by XPointer to describe location paths.

A location path consists of location steps, which in turn consist of an *axis*, a *node test,* and a *predicate.* The expression below consists of two location steps, evaluated left to right, one starting with child and the other with attribute. It describes the id attributes of the first two AUTHORs within some externally defined initial context.

```
child::AUTHOR[position()<3]/attribute::id
```

The axis (the part before the ::) helps you navigate through the document. Possible axis values include child, attribute, parent, and following-siblings. The node test can be a tag or attribute name (AUTHOR in the example) or the wildcard *. With functions like text() or comment(), other subelements can be addressed. The predicate in the square brackets defines the filter.

XPointer uses XPath to define fragment identifiers. A fragment identifier can be the value of an attribute of type id, a sequence of numbers (for example, /1/4/2/10), or a sequence of XPointer expressions (xpointer(…)) as in the example:

```
http://www.myserver.net/#xpointer(//BOOK/AU
THOR[position()=1])
```

In the XPointer expression above (in parentheses), // is an abbreviation for the descendent-or-self axis whereas AUTHOR comes without an axis: child() is the default. XPointer also defines the initial context for the XPath expression. The most important part of the initial context is the context node, which is the root node of the document specified by the fragment identifier.

With XLink you can link documents together. Links can be either simple or extended. A simple link is very similar to an HTML link.

XLink uses its own namespace, identified by the URI http://www.w3.org/1999/xlink, and typically associated with the prefix xlink. The attributes in Figure 4 are all part of the XLink specification.

- href lets you specify a URI together with a fragment identifier.
- role indicates the purpose of the linked document (or, more accurately, the linked element) in the linking document.
- show specifies what is to be done with the linked document when it is loaded. For example, show="embed" dictates that the linked document is to be embedded in the current document. Other values for show are replace, new, and undefined.
- actuate specifies when the action indicated by show should occur. Values for actuate are onLoad, indicating that the linked document is to be embedded while the current document is loading; onRequest, indicating that it is to be embedded when requested; or undefined.

Extended links connect more than two documents (generally called resources)—that is, more than one source document or more than one target document. The LINKS element in Figure 5 is a linking element of type extended and consists of the LINKS element's children. A local resource has the type resource, whereas a remote resource confusingly has the type locator. Again, the role attribute is used to describe the purpose of the linked document.

Elements of type arc describe traversals between resources. The example in Figure 6 (next page) introduces two arcs, one from the resource with the role home to the resource with the role amazon, and another to the resource with the role barnes_and_noble. By using the same role for multiple resources, arcs can have multiple sources and targets. In the current specification, the application decides how to handle arcs.

XLink also allows you to define and use link databases in external documents.

XML Examples (Figures 4–5)

```
<AUTHOR xmlns:xlink=http://www.w3.org/1999/xlink
   xlink:type="simple"
   xlink:href="http://www-cs-faculty.stanford.edu/ knuth/"
   xlink:role="don_~knuth_homepage"
   xlink:show="embed"
   xlink:actuate="onLoad">
Donald Knuth </AUTHOR>
```

Figure 4. A simple Xlink. Simple XLinks are similar to HTML links, but they also let you define the behavior of the link.

```
<BOOK>
   <AUTHOR> Ullman, J. D. </AUTHOR>
   <TITLE> Principles of Database and Knowledge-Base Systems, Vol. 1
</TITLE>
   <LINKS xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="extended"
      xlink:title="My Book World">
      <MYPAGE xlink:type="resource"
         xlink:role="home">
         My Page on Ullman's Book
      </MYPAGE>
      <AMAZON xlink:type="locator"
         xlink:href="http://www.amazon.com/exec/obidos/..."
         xlink:role="amazon"/>
      <BARNES_AND_NOBLE xlink:type="locator"
         xlink:href="http://shop.barnesandnoble.com/booksearch/..."
         xlink:role="barnes_and_noble"/>
   </LINKS>
</BOOK>
```

Figure 5. An extended Xlink. Extended XLinks let you define collections of links.

Stylesheets and More

The Extensible Stylesheet Language (XSL) is actually two languages: a transformation lan-

XML Examples (Figures 6–7)

```
<BOOK>
...
   <LINKS xmlns:xlink=http://www.w3.org/1999/xlink
   ...
      <ARC xlink:type="arc"
         xlink:from="home"
         xlink:to="amazon"
         xlink:show="replace"
         xlink:actuate="onRequest"/>
      <ARC xlink:type="arc"
         xlink:from="home"
         xlink:to="barnes_and_noble"
         xlink:show="replace"
         xlink:actuate="onRequest"/>
   </LINKS>
</BOOK>
```

Figure 6. Arcs in an extended Xlink. With arcs you can define how to traverse extended XLinks.

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
   <HTML><xsl:apply-templates/></HTML>
</xsl:template>

<xsl:template match="BIB">
   <UL><xsl:apply-templates/></UL>
</xsl:template>

<xsl:template match="BOOK">
   <LI><xsl:apply-templates/></LI>
</xsl:template>

<xsl:template match="AUTHOR">
   <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="TITLE">
   <EM><xsl:value-of select="."/></EM>
</xsl:template>

</xsl:stylesheet>
```

Figure 7. A simple XSL stylesheet for transforming the XML bibliography into an HTML unordered list.

guage (called XSL transformations, or XSLT) and a formatting language (XSL formatting objects).

XSLT allows you to transform XML into HTML, thus bypassing the formatting language. It also lets you restructure XML documents so that different kinds of XML representations can be mapped onto one another. This makes XSLT very useful for electronic commerce and electronic data interchange.

Figure 7 shows XSLT code for turning the bibliography entry in Figure 1b into an HTML representation like the one in Figure 1a. XSLT uses the namespace identified by http://www.w3.org/1999/XSL/Transform (Internet Explorer 5.0, which supports XSLT, references http://www.w3.org/TR/WD-xsl).

The stylesheet element contains a collection of template elements and can be included in the document it is to be applied to. The match attribute of a template element addresses the document structures the template can be applied to. Arbitrary XPath expressions can be incorporated into valid values of the match attribute.

The template element indicates the output to be produced. In Figure 7, the root template produces an <HTML>...</HTML> frame, the template for BIB elements produces an unordered list frame (<UL>...</UL>), and the template for BOOK elements produces a list entry frame (<LI>...</LI>). These templates all contain an <xsl:apply-templates/> element that causes the XSLT processor to apply applicable templates on the subelements of the current element recursively.

The templates for AUTHOR and TITLE elements contain an <xsl:value-of select="."/> subelement, which causes the value of the current element (indicated by the path expression ".") to be output. Incidentally, the value of the select attribute can be an arbitrary XPath expression. In the example, the value of the TITLE element is emphasized in the generated HTML output.

For a more detailed discussion of XSL, see http://www.w3.org/Style/XSL/.

XML Schema

Although DTDs were the first proposal to provide for a standardized data exchange between users, they have disadvantages. Their expressive power seems limited, and their syntax is not XML. Several approaches address these disadvantages by defining a schema language (rather than a grammar) for XML documents:

- document definition markup language (DDML), formerly known as XSchema,
- document content description (DCD),
- schema for object-oriented XML (SOX), and
- XML-Data (replaced by DCD).

The W3C's XML Schema activity takes these four proposals into consideration.

XML Schema is well-formed XML that allows the user to define datatypes. The example in Figure 8 (on page 80) describes a datatype for BOOK elements described by a DTD in Figure 3.

Again, XML Schema uses its own namespace.

The complexType element indicates a complex datatype associated with the nonterminal element BOOK and consisting of other elements and attributes.

XML Schema supports a variety of atomic datatypes, such as string, decimal, and date. Datatypes that have been defined to retain compatibility with DTDs, such as id and idref, should only be used for attributes. The user can simulate the +, *, and ? of DTDs or regular expressions using the minOccurs and maxOccurs attributes.

Figure 9 shows the definition of a SECTION element written (a) as a DTD and (b) in XML Schema. XML Schema also distinguishes between sequences and choices, which are instances of the group element. Groups can be defined outside of a type and referenced using the ref attribute.

XML Schema also supports inheritance as part of the more general concept of derivation. A type definition can be accompanied by a base (the name of the base type) and a derivedBy (the kind of derivation) attribute. Possible values for the derivedBy attribute include extension and restriction. As a further extension, a list element lets you define a list of elements of some base type.

The ultimate goal of XML Schema can only be to replace DTDs. Currently, its chances of doing so remain unclear. For more information about XML Schema, see http://www.w3.org/XML/Schema.html.

Other Developments to Watch

To sum it up, the XML wave really introduces a family of languages, which are dedicated to a more semantic management of information.

In addition to the languages discussed here, there are other important developments that affect XML activity. Two such developments are the resource description framework (RDF), which integrates a variety of Web-based metadata activities; and the document object model (DOM), which provides an interface to allow programs to access and update the content, structure, and style of documents dynamically. A future issue of *IEEE Internet Computing* will include a tutorial on RDF. ■

Acknowledgments

Katerina Cai, Stefan Decker, Felix Naumann, Dieter Scheffner, and Jörg Schenk made numerous comments and proofread the manuscript. I am currently supported by the German Academic Exchange Service (DAAD program HSP III).

André Bergholz is a visiting postdoc in the Stanford University database group. He received a diploma in 1996 and a PhD in 2000, both from Humboldt University Berlin. His research interests include the management of semistructured data and the application of database technology in computational biology.

Readers can contact the author at bergholz@db.stanford.edu.

XML Examples (Figure 8–9)

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<xsd:element name="BOOK" type="BOOK_TYPE"/>

<xsd:complexType name="BOOK_TYPE" >
   <xsd:element name="AUTHOR" type="xsd:string"
      minOccurs="1" maxOccurs="unbounded"/>
   <xsd:element name="TITLE" type="xsd:string"/>
   <xsd:element name="PUBLISHER" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
   <xsd:element name="YEAR" type="xsd:decimal"
      minOccurs="0" maxOccurs="1"/>
   <xsd:attribute name="isbn" type="xsd:string"/>
   <xsd:attribute name="nickname" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

Figure 8. The XML Schema for the bibliography in Figure 2. Like a DTD, the XML Schema defines a grammar for the document. However, XML Schema uses XML syntax and is more expressive.

```
<!ELEMENT SECTION ((TITLE, (PARAGRAPH+)) |
   (TITLE, (PARAGRAPH*), (SUBSECTION+)))>
```

(a)

```
<xsd:element name="SECTION" type="SECTION_TYPE"/>

<xsd:complexType name=" SECTION_TYPE">
   <xsd:choice>
      <xsd:group ref="plainSection"/>
      <xsd:group ref="nestedSection"/>
   </xsd:choice>
</xsd:complexType>

<xsd:group name="plainSection">
   <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="PARAGRAPH" type="xsd:string"
         maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:group>

<xsd:group name="nestedSection">
   <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="PARAGRAPH" type="xsd:string"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="SUBSECTION" type="SUBSECTION_TYPE"
         maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:group>
```

(b)

Figure 9. Sequences and choices in (a) a DTD and (b) XML Schema.