What Can You Do With XML Today?

By Jay Ven Eman, Chief Executive Officer, Access Innovations, Inc.

Interest in Extensible Markup Language (XML) rivals the press coverage the World Wide Web received at the turn of the Millennium. Is it justified? Rather than answer directly, let us take a brief survey of what XML is, why it was developed, and highlight some current XML initiatives. Whether or not the hype is justified, NFAIS members will inevitably become involved in your own XML initiatives.

The Author: Jay Ven Eman joined Access Innovations, Inc. in December, 1978. As Chief Executive Officer, he has contributed to every aspect of the business. He has overseen Access' database production services where he was responsible for the design and conversion of large, legacy databases for a variety of government and private organizations. He writes, conducts workshops, consults, and helps build large scale, complex, information rich databases. Jay works closely with the professional staff, developing and presenting workshops and seminars on database development, legacy file conversions, SGML and HTML, and other related topics.

An alternative question to the title, "What can't you do with XML?" I use it to brew my coffee in the morning. Doesn't everyone? To prove my point, the following is a "well-formed" XML document. ("Well-formed" will be defined later in the article.)

```
<?xml version="1.0" standalone="yes"
encoding="ISO-8859-1"?>
<StartDay Attitude="Iffy">
<Sunrise Time="6:22" AM="Yes"/>
<Coffee Prepare="Or die!" Brand="Starbuck's"
Type="Colombian" Roast="Dark"/>
<Water Volume="24" UnitMeasure="Ounces">Add
cold water.</Water>
<Beans Grind="perc" Type="Java">Grind coffee
beans. Faster, please!!</Beans>
<Grounds>Dump grounds into coffee
machine.</Grounds>
<Heat Temperature="152 F">Turn on burner</Heat>
<Brew>Wait, impatiently!!</Brew>
<Dispense MugSize="24"
UnitMeasure="Ounces">Pour,
        finally.</Dispense>
</StartDay>
```

This XML document instance contains sufficient information to drive the coffee making process. Given the intrinsic nature of XML, our coffee-making document instance could be used by the butler (should we be so lucky) or by a Java applet or perl script to send processing instructions to a wired or wireless coffeepot. If XML can brew coffee, it can drive commerce; it can drive R & D; it can drive the information industry; it can drive the uninitiated out of business.

What is XML? To understand XML, you must understand meta data. Meta data is "data about data." It is an abstraction, layered above the abstraction that is language. Meta data can be characterized as natural or added. To illustrate, consider the following text string, "Is MLB a sport, entertainment, or business." You, the reader, can guess with a degree of accuracy that this is an article title about Major League Baseball (MLB). Presented out of context, even people are only guessing. Computers have no clue, in or

out of context. There are no software systems that can reliably recognize it in a meaningful way.

For this example, it is a newspaper article title. To it we will add subject terms from a controlled vocabulary, identify the author, the date, and add an abstract. As a "well-formed" XML document instance, it is rendered:

```
<?xml version="1.0" standalone="yes" encoding="ISO-8859-1"?>
<DOC Date=5/21/02 Doctype="Newspaper">
<TI> "Is MLB a sport, entertainment, or business?" </TI>
<Byline> Smith </Byline>
<ST> Sports </ST>
<ST> Entertainment </ST>
<ST> Business </ST>
<AB> Text of abstract...</AB>
<Text> Start of article ...</Text>
</DOC>
```

In this example, what are the meta data? What is natural and what is added? Natural meta data is information that enhances our understanding of the document and parts thereof, and can be found in the source information. The date, the author's name, and the title are "natural" meta data. They are an abstraction layer apart from the "data" and add significantly to our understanding of the "data."

The subject terms, document type, and abstract are "added" meta data. This information also adds to our understanding, but it had to be added by an editor and/or software. The tags are "added" and are meta data. Meta data can be the element tag, the attribute tag, or the values labeled by element and attribute tags. It is the collection of meta data that allows computer software to reliably deal with the data. Meta data facilitates networking across computers, platforms, software, companies, cities, countries.

Generalized markup experimentation began in the late 60's and 70's at places like IBM and within the secondary publishing industry. For a brief period in the 80's, the file loading format Dialog Information Services, Dialog format b, became a *de facto* markup standard for secondary publishers.

What is the "data" in this example? The text, tables, charts, figures, and graphs that are contained within the open <Text> and close </Text> tags.

Generalized data markup has four objectives: structure, content, context, and format. Markup can articulate and even dictate structure. Structure in a text object includes such concepts as document beginning and end, that the document must have a chapter and at least one or more subchapters and the subchapter(s) must follow the chapter, and so on. Your goal with structure is to render the hierarchical and organizational design of the data and the parts thereof.

Content markup identifies what the data means such as a given text string represents the title of a sidebar and not the title of the article. Author, title, date, and phone number are tags that identify the content characteristics of data.

Context markup labels content so that additional meaning can be derived from a content markup tag such as "title." For example, a document type label or tag would add significant information about the title. In our example about Major League Baseball, a context sensitive tag, document type, identifying "title" as coming from a newspaper, would have different connotations from document type equals a legal journal.

Format markup allows you to determine how the document instance will be displayed. Generalized markup designers are moving to a portable display markup. A portable display markup uses a style sheet that accompanies the document instance. This makes it much easier to change how a document is displayed. A document instance can have any number of style sheets. Markup indicating the title should be displayed using Times New Roman font, 44 point, bold, and centered at the top of page one could be embedded in the document instance at the point in the document where the title begins. By placing these instructions in a style sheet the document instance is freed from the burden of carrying format information. Style changes become easier.

Generalized markup experimentation began in the late 60's and 70's at places like IBM and within the secondary publishing industry. For a brief period in the 80's, the file loading format Dialog Information Services, Dialog format b, became a *de facto* markup standard for secondary publishers. As generalized markup developments moved from the labs to the standards arena and started to become a meta language, three basic parts to generalized markup language (GML) emerged.

One, GML needs a declaration at the start of a document. In both XML examples above, the opening line of each denotes the markup language to be invoked, XML version 1.0. There are many GML's in use and within any given GML there are many implementations. Providing a GML declaration gives the document user, either machine or human, the information needed to begin decoding the markup.

Question: Are the angle brackets, "<" and ">", required?

Answer: They are not required by the SGML standard, but they *are* a part of the XML recommended implementation.

Two, GML relies on a document type definition (DTD) or schema. Using the rules, syntax, etc., of the meta language, the DTD is the set of instructions needed to markup an actual document. The DTD describes the tags, their meaning, and how to use them. It is the DTD that says the publication date in the document will be encoded using <pubdate> and </pubdate>. The DTD is necessary because the labels or tags are arbitrary. Publication date can be just as well defined and marked up as <publicationdate> or <PD>. Many, many design issues and processing instructions are articulated in a DTD or schema. The two examples above are much easier to process by machine or human if accompanied by a DTD.

Three, GML requires a "document instance." GML can be used to encode any digital object, but textual oriented digital files, documents, are currently very common, so "document" is used in conjunction with "instance" to refer to all GML objects. The document instance is your newspaper article about MLB with the markup encoding it. When you have a document instance, you have an article ready for use by a GML tool such as a WEB browser.

Continuing on the history of GML, the publishing industry really pushed strongly for the development of a GML. Standard Generalized Markup Language (SGML) resulted. It is an ISO Standard, ISO 8879:1988 (www.iso.org.) SGML helped publishers and others to wrench the data from proprietary editorial and photocomposition systems. The push came about as publishers recognized a new revenue stream in digital content. Extracting their data from proprietary markup, markup used only for format, was nearly impossible. As publishers implemented SGML, photocomposition vendors developed import and export routines to handle it.

SGML is a meta language. It does not specify how to markup any given document. It allows for application and platform independence. It is portable. Sharing and re-packaging of information is possible.

SGML is very complex and expensive to implement and maintain. The software used to support SGML is expensive and complex. It has no mainstream browser support. Because of its complexity, it is not WEB friendly.

Extensible Markup Language (XML) was developed to overcome these limitations and help foster the burgeoning Internet. SGML was used as the foundation or starting point in the development, but it is incorrect to label XML as a sibling of SGML. Like SGML, XML is a meta language. It allows implementers the flexibility to articulate their own markup strategy. HTML is a specific application and resulting implementation of SGML. HTML is a published, support DTD. HTML could be labeled a sibling of SGML, but "application" or "implementation" is a better descriptor of SGML and XML DTDs.

It is a World Wide Web Consortium (W3C) Recommendation REC-XML-19980210 (www.w3c.org.) No DTD or schema is required, but can be designed and used in conjunction with XML document instances. Content and context tags are possible. XML can make use of style sheets. It does not have all of the features and complexity of SGML and, thus, is easier to support and easier to share XML document instances in a networked world. XML is a family of recommendations and initiatives.

XML document instances can be generated and used without benefit of a DTD or schema. This can happen if the document instance is "well-formed." An XML document instance is "well-formed" if it adheres to XML syntax. The document intance must have an XML declaration. XML syntax requires that each document instance has a root, or document, tag that immediately follows the XML declaration. It can be used only once and must close the document instance. All tags are closed and properly nested.

Both examples above are well-formed. They start with an XML declaration. The root, or document, tag is <StartDay></StartDay> for our coffee document and <DOC></DOC>

for the sports article. Be aware that XML is case sensitive regarding element tags. Using <StartDay> and </Startday> would result in a parsing error. The parser would report that the document was not well-formed because the element, <StartDay>, was not closed, that is, it had no end-tag, </StartDay>. All tags must be closed.

XML syntax requires proper nesting of tags. Here is an example of properly nested and improperly nested tags.

Properly Nested Tags

```
<Brew>Wait, impatiently!!</Brew>
<Dispense MugSize="24" UnitMeasure="Ounces">Pour,
finally.</Dispense>
```

Improperly Nested Tags

```
<Brew>Wait, impatiently!!
<Dispense MugSize="24" UnitMeasure="Ounces">Pour,
finally.</Brew></Dispense>
```

The second markup is not well-formed because <Dispense> opens before the closing tag, </Brew> is encoded. The open and close tags are crossed. The coffee brewing instructions would not parse, resulting in no coffee and that would be disastrous!

Validity is another important XML concept. A document instance is valid when it parses successfully against its accompanying DTD or schema. As mentioned XML does not require the use of a DTD, but when a document instance invokes a DTD or schema, then a parser will validate the document instance against it. If it parses successfully, the document is both well-formed and valid. A document instance can be well-formed, but not valid if it has a DTD, but violates the rules of its DTD. For example, its schema may require an element to be numeric only. If the numeric only element contains an alpha character, it would be invalid. Parsed without invoking its DTD and it could very likely be well-formed. If a document instance is valid, then it is always well-formed.

Early in the development of XML, "namespaces" was introduced to allow for the resolution of ambiguous elements and attributes that inevitably occurs as information zooms around the Internet or even a corporate intranet. Extending our coffee example illustrates a simple case of namespace use.

```
<?xml version="1.0"?>
<Startday
        xmlns:USA="http://coffee.org/usa/measures"
        xmlns:UK="http://coffee.org/uk/measures">
<USA:ounces>24</ounces>
<UK:ounces>24</ounces>
</Startday>
```

In this example, using the namespace syntax disambiguates the "ounces." The first instance of "ounces" has an element value 24, but the namespace syntax tells the parse

to use the definition of "ounces" found at the url pointing to the USA subdirectory. The second "ounces" element points to the subdirectory of UK. At each unique URL, the parse will find a definition for the element, "ounces." It is possible that both schemas define "ounces' similarly, but without the schema or DTD, the element remains ambiguous to the parser.

XML does not require a DTD, but DTD's are very useful. When a DTD accompanies its document instance, the recipient can do much more with the document. The DTD defines the elements, attributes, and entities used within the document instance. This enhances understanding and reduces ambiguity. An XML DTD allows for much of the functionality described above. SGML and XML allow for attributes and entities. Briefly, an attribute is meta
data about an element.

<Beans Grind="perc" Type="Java">

"Grind" and "Type" are attributes and provide additional descriptive and processing information about the element, "Beans. The element "Beans" could appear in the document instance as, <Beans>, but with the addition of the attributes, you gain greater understanding of the element.

Entities are shortcuts similar to keyboard macros. They are used within a DTD and within the document instance. Within a DTD they can be used to summarize a long series of elements, attributes, and other entities, so they can be grouped and reused without significant typing. They are used within the document instance to represent special characters such as the Greek character beta. They can be used to represent a long data or text string that is used many times within the document instance. The parser inserts the correct character or text string on rendering.

One limitation of an XML DTD is that it is not a well-formed XML document instance. XML schemas are well-formed and are functionally equivalent to DTD's. However, schemas are more powerful than XML DTD's because their functionality has been extended. A schema can include extensive processing instructions and parameters used by software for any number of reasons. Data can be manipulated, generated, reorganized, and checked for errors. Element and attribute values can be better controlled fostering greater data quality.

Given the greater functionality of XML schemas, processing initiatives have been developed. Two processing initiatives are Digital Object Model (DOM) and Simple API for XML (SAX). DOM has a Level 1 W3C recommendation in circulation. DOM defines a programmatic interface for traversing the document instance hierarchy, and manipulating elements and attributes. DOM places the entire schema structure into memory, parsing it into a tree structure. Using a DOM implementation, programs can treat each node of the tree as an object and perform endless manipulations. SAX is being utilized to overcome the memory requirements of DOM, which can be onerous, for very complex schema applied to long documents. SAX as the name implies is simpler than DOM. Processing initiatives are the interface between your data and your software

environment. DOM would layer between your SML repository, or database, and your photocomposition and fulfillment software.

An example of an XML transformation initiative is Extensible Stylesheet Language (XSL). XSL transform XML into HTML and other presentation formats including other XML. It allows extensive reordering, generating text, and calculations, but does not modify the source data. Because XSL uses XML syntax an XSL is well-formed. It can be sent with its document instance. Even multiple XSL can be sent with a document instance. Other transformation initiatives are underway. Transformations allow multiple rendering, or views, of your data. Your data requirements and the dominant trends in your field should guide the choice of initiatives.

A visit to an XML WEB site like www.xml.org reveals a bewildering array of XML schema, processing, and transformation initiatives. Where do you start? If helpful, go to the faq's and tutorial pages. There is even an article titled, "XML for Dummies." Once you're comfortable with XML jargon, visit the vertical market pages. These represent initiatives in just about evey field of endeavor from Agriculture to Zoology, from profit to nonprofits, from large to small. Stick to what is closest to "home." Such a visit can be overwhelming. A danger of XML is that it could become a "Tower of Babel." Too many competing schema, requiring too much overhead to properly cross-reference using namespaces or anything else.

A look into the "Publishing/Print" page at XML.org references DocBook, Text Encoding Initiative (TEI), and Job Definition Format (JDF) among others. An important initiative here is the ONIX International DTD. This has been adopted by many book wholesalers and retailers including Amazon and Ingram. "ONIX International is the international standard for representing and communicating book industry product information in electronic form, incorporating the core content which has been specified in national initiatives such as BIC Basic and AAP's ONIX Version."

Much of this information has been captured for years using AACRII and communicated in electronic form using MARC. The book industry has additional requirements in creating and facilitating commerce that libraries do not. Libraries have their own needs. The Library of Congress is helping to develop Metadata Object Description Schema (MODS) and Metadata Encoding & Transmission Standard (METS). MODS is an XML schema for a bibliographic element set and will carry "selected data from existing MARC 21 records. METS is an XML schema for "encoding descriptive, administrative, and structural metadata regarding objects within a digital library." MODS deals with just bibliographic information. METS woould be used to encode an entire digital book, for example, and could use MODS for the books bibliographic information. The Library of Congress has recently released both schemas for comment.

What can you do with XML? What can't you do with XML? Data Harmony, Inc. (www.dataharmony.com) developed a complete database management system around XML. The product is called XIS, for XML Intranet Management System. It uses a hierarchy of XML schema to drive the system. Written in Java, it employs a system level XML schema that articulates all of the operations needed for creating a structured textual database. Each database application is defined in a project level schema. The

project level schema contains the processing instructions to markup the document instance. The document instance is stored and processed as an XML object. That is, the XIS database management system stores data in XML, rather than relying on a proprietary markup as do most database management systems. This provides tremendous flexibility and yet, great control over database operations.

Using ONIX, a book buyer could receive an ONIX document instance that contains information about the dimensions and weight of a book they're considering ordering. Because ONIX is a published XML schema, the buyer can have their computer programmed to read the ONIX document instance and calculate the weight and cubic feet an order of one thousand units. This can be very valuable and a great timesaver. Without standard markup, Internet commerce won't happen. Your ability to transmit your commercial databases will be greatly enhanced. It frees your data from legacy hardware and software. It allows you to develop efficiently, rapidly new products and services, generating new revenue streams. Lower operating costs; new revenue streams? Worth thinking about!