Encyclopedia of Information Systems and Technology

Phillip A. Laplante, Vladimir Stantchev, Jeff Voas, Anne-Marie Smith, Qiuming Zhu, Morris Chang, Irena Bojanova, David Alan Grier, Jia Zhang, Thomas Costello, San Murugesan, Jim Goldman, Joanna DeFranco

XML and Other Metadata Languages

Publication details
William Hugh Murray
Published online on: 29 Dec 2015

How to cite :- William Hugh Murray. 29 Dec 2015, XML and Other Metadata Languages from: Encyclopedia of Information Systems and Technology CRC Press
Accessed on: 18 Aug 2023

PLEASE SCROLL DOWN FOR DOCUMENT

Full terms and conditions of use: https://www.routledgehandbooks.com/legal-notices/terms

This Document PDF may be used for research, teaching and private study purposes. Any substantial or systematic reproductions, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The publisher shall not be liable for an loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.
XML and Other Metadata Languages

William Hugh Murray
Executive Consultant, TruSecure Corporation, New Canaan, Connecticut, U.S.A.

Abstract
HyperText Markup Language (HTML) and similar metadata languages have given us levels of interoperability that were not dreamed of a decade ago. As the number of interoperable systems on the Internet has risen linearly, the value to the users has risen exponentially. EXtensible Markup Language (XML) promises us another order-of-magnitude increase in that interoperability. Not only will it help create interoperability between clients and servers on the Internet, but it will also improve interoperability among arbitrary objects and processes wherever located. By conserving and communicating the meaning and intent of data, it will increase its utility and value. Not since the advent of Common Business-Oriented Language (COBOL) has there been a tool with such promise; this promise is far more likely to be realized and may be realized on a grand scale.

When the author was a beardless boy, he worked as a punched-card machine operator. These were primitive information processing machines in which the information was stored in the form of holes punched in paper cards. Although paper was relatively cheap by historical standards, by modern standards it was very expensive storage. For example, a gigabyte of storage in punched paper would fill the average room from floor to ceiling, wall to wall, and corner to corner. It was dear in another sense, that is, there was a limit to the size of a record. A “unit record” was limited to 80 characters when recorded in Hollerith code. This code in this media could be read serially at about 10–15 characters per second. In parallel, it might be read at 8–12 thousand characters per minute.

As a consequence, application designers often used very dense encoding. For example, the year in a date was often stored as a single digit and two digits when the application permitted it. This was the origin of the famous Y2K problem. As the Y2K problem resolved, it was often thought of as a programming logic problem. That is, the program would not process years stored as four digits and might interpret 2000 as being earlier than 1999 rather than later. However, it was also a quality of data problem. When the year was encoded as one or two digits, information was often permanently lost. In fixing the problem, one often had to guess what the real data was.

The meaning of a character in a punched-card record was determined by its position in the record. For example, an account number might be recorded in columns 1 to 8 of the card. Punched-card operators of large stable applications could often understand the records from that application by looking at the color of the card and determine what information was stored in which columns by looking at the face of the card where the fields were delineated and identified. When dealing with small or novel applications, one often had to refer to a “card layout” recorded on a separate piece of paper and stored in a binder on the shelf. Because this piece of paper was essential in understanding the data, its loss could result in the loss of the ability to comprehend the data.

The name of the file was often encoded in the color of the card, and the name of the field in its position in the card. The codebook might have been printed on the face of the card or it might have been stored separately. In any case, it was available to the operators, but not to the machine. That is, the data about the data was not machinereadable and could not be used by it.

This positional encoding of the meaning of information and separate recording of its identity on a piece of paper carried over into early computer programming. Therefore, when starting to resolve the Y2K problem, one could not rely on the machine to identify where instances of the problem might appear, but had to refer to sources external to the programs and the data.

METADATA

In modern parlance, this data about the data is called metadata. Metadata is used to permit communication about the data to take place between programs that do not otherwise know about each other. Database schemas, style sheets, tagged languages, and even the data definition section of Common Business-Oriented Language (COBOL) are all examples of metadata. Because storage is now both fast and cheap, modern practice calls for the storage of this metadata with the data that it
describes. In many applications and protocols, the metadata is transmitted with the data. A good example is electronic data interchange (EDI), in which fields carry their meaning or intended use in tags.

Good practice says that one never stores or moves the data without the metadata. Preferred security practice says that the metadata should be tightly bound to the data, as in a database, so as to resist unintended change and to make any change obvious. In object-oriented computing, the data, its meaning, and all of the operations that can be performed upon it may be bound into a single object. This object resists both arbitrary changes and misunderstanding.

Tagged Languages

One form of metadata is the tag. A tag is a specially formatted field that contains information about the data. It is associated with the data to which it refers by position, that is, the tag precedes the data. Optionally but often, the tag refers to everything after it and before a corresponding end tag.

XML is a tagged language. In this regard, it is similar to HTML, EDI, and SGML. A tag is a variable that carries information about the data with which it is contextually associated. A tag is metadata. To a limited degree, tags are reserved words. Only limited reservation is required because, as in these other tagged languages, tags are distinguished from data by some convention. For example, tags can be distinguished by bracketing them with the left and right pointing arrows, <tagname>, or beginning them with the colon, :tagname. Each tag has an associated end tag that is similarly distinguished, for example, by beginning the end tag with the left pointing arrow followed by a slash, </tagname> or the colon followed by the letter “e,” :etagname. The use of end tags eliminates the need for a length attribute for the data. Tags are often nested. For example, the tags for name and address may appear inside a tag for name and address.

A tagged language is a set of tag definitions. Such a set, language, dialect, or schema is defined in a Document Type Definition object. This schema can be encapsulated in the object that it describes, or it can be associated with it by reference, context, or default. These language definitions can be, and usually are, nested. This provides maximum functionality and flexibility but may cause confusion.

The concept of “markup” comes from editing and publishing. The author submits a document to the editor who “marks up” the text to communicate with both the author and the printer or composer. One early tagged language was the Generalized Markup Language, perhaps the prototypical markup language. However, the concept of markup suggests something that is done in a separate step to add value or information to the original. Many of the tagged languages called markup languages are really not markup languages in that special sense.

As with most languages, tagged languages provide for special usage. They provide for special vocabularies that may be meaningful only in a special context. For example, the meaning of the word “security” is different when used in financial services than when it is used in information technology. Similarly, EDI uses a number of different vocabularies, including X12, EDIFACT, TRADACOMS, which are applicable only in their intended applications.

The eXtensible Markup Language

XML is a language for describing data elements. It describes the attributes of the data and identifies its intended meaning and use. It consists of a set of tags that are associated with each data element and a description that decodes the tag. Keep in mind the analogies of a database schema and a record layout. Also keep in mind the limitations of these languages. Think of the analogy of HTML; as HTML says this is how to display or print it, XML says these are its attributes and this is what it means. XML is not magic.

XML is an open language. That is why it is called extensible. Of course, all programming languages are extensible to some degree or another. The dynamic HTML bears only a family resemblance to the HTML of a decade ago. Browsers are dynamically extensible through the use of plug-ins and the Dynamic Object Model (DOM). Modern HTML is dynamically extensible, extensible on-the-fly. The capabilities of the interpreter are dynamically extended through the use of plug-ins, applets, and similar mechanisms.

The owner of the object in which XML is used is permitted to define arbitrary tags of their own choice and embed their definition in the object. The meaning and attributes of a new tag are described in old tags. XML is a dialect of the Standard Generalized Markup Language, developed by IBM and adopted as an ISO standard. XML is the parent of a number of dialects, including cXML (Commerce XML), VXML (Voice XML), and even MSXML (Microsoft XML). There can be dialects for industries, applications, and even services. However, the value of any dialect is a function of the number of parties that speak it.

XML is a global language. That is to say, it has global schemas that go across enterprises, industries, and even national boundaries. These schemas represent broad prior agreement between users and applications on the meaning and use of data. The scope of the vocabulary of XML can be contrasted to that of programming languages such as COBOL where the data description is usually limited to an enterprise and often to a single
program, where the base set of verbs is common across enterprises but there are no common nouns.

XML implements the concept of namespaces. That is, it provides for more than one agreement between a name and its meaning. The intended namespace is indicated by the name of the space, followed by a colon in front of the tagname (\texttt{<ns:tagname>}). There can be broad agreement on a relatively small vocabulary with many special vocabularies used only in a limited context.

XML is a declarative language. It makes flat statements. These statements are interpreted; they are not procedural. It says what is rather than what to do. However, one must keep in mind that tagnames can encapsulate arbitrary definitions that are the equivalent of arbitrary procedures.

XML is an interpreted language. Like BASIC, Java, and HTML, it is interpreted by an application. However, to provide for consistency and to make XML-aware applications easier to build, most will use a standard parser and a standard definition or schema.

It is recursive. The XML schema, the object that defines XML, is written in XML. It can include definitions by reference. For example, it can reference definition by uniform resource locator (URL). Indeed, because it increases the probability that the intended definition of the tag will be found, this style of use is not only common, but also frequently recommended. Of course, from the perspective of the owner of the data, this is safe; it ensures the owner that the tags will be interpreted using the definitions that the owner intended.

From the perspective of the recipient of the data, it may simply be one more level of indirection (i.e., sleight of hand) to worry about. The good thing about this is that URLs begin with a domain name. (Keep in mind that,

Table 1 The E-Wallet: An example

| A good example of the use of metadata in communication is the E-wallet application. Its owner uses the e-wallet to store and use electronic credentials. These include things such as name and address, user IDs and passwords, credit card numbers, etc. Because all of this information is sensitive to disclosure, it is usually stored in a database. The database can hide the data and associate it with its metadata, its intended meaning and use. Alternatively, the data could be stored in a flat file using tags for the metadata and file encryption to hide the data in storage when not in use. The user employs the E-wallet application to present the credentials in useful ways. For example, suppose that the user has decided to make a purchase from an online merchant. After making a selection, the user presses the checkout button on the screen and is presented with the checkout screen. This screen asks for the name and billing address, name and shipping address, and charge information. The user invokes the E-wallet application to complete this screen. The E-wallet presents the data stored in it and the user clicks and drags it to the appropriate fields on the checkout screen. The user knows what information to put in what places on the screen because the fields are labeled. These labels are put on the screen using HTML. While they are visible to the user, they are not visible to the E-wallet application. Therefore, the user must do the mapping between the fields in the E-wallet and those on the checkout screen. Although this process is flexible, it is also time-consuming. Although it ultimately produces the intended results, it relies on feedback and some intermediate error correction. When the screen is completed to the user’s satisfaction, the user presses the Submit button. At this point, the screen is returned to the merchant where the merchant’s computer verifies it further and might initiate another round of error correction. If, in addition to labeling the fields on the screen with HTML, the merchant also labeled them with XML, then an XML-aware E-wallet could automatically complete part of the checkout screen for the user. If the checkout screen requests billing information, the E-wallet will look to see if it has the information to complete that section. In the likely case that it has more than one choice, it will present the choices to the user and the user will choose one. When the screen is completed to the user’s satisfaction, the user will press the Submit button. When the screen is received to the merchant, the data is suitably labeled with his XML so that his XML-aware applications and those of his trading partners (e.g., his credit card transaction service) can validate the data. The use of XML has not changed the application or its appearance to the user. It has not changed the data in the application or its meaning. It has simply facilitated the communication between XML-aware applications. It has made the communication between the applications more automatic. Data is stored where it is supposed to be, controlled as it is supposed to be, and communicated as it is supposed to be. The applications behave more automatically and the opportunity for error is reduced. Notice that the applications of some merchants, most notably Amazon, achieve the same degree of automation. However, they do it at the cost of replicating the data and storing it in the wrong place, that is, user data is stored on the merchant system. This can and has led to compromises of that data. While one might argue that the data is better protected on the merchant’s server than on the customer’s client, the aggregation of data across multiple users is also a more attractive target.

Just as there are multiple browsers, there will be multiple E-wallet applications. As the requirement for the browser is that it recognizes HTML, the requirement for the E-wallet is to speak the same dialect of XML as the merchant’s application. To make sure that it speaks the same dialect of XML as the merchant, the E-wallet may speak multiple XML dialects, similar to the way that browser applications speak multiple encryption algorithms. Notice that the merchant’s application could request information from the user’s E-wallet that it does not display on the screen and which the user does not intend to provide. The user relies on the behavior of his application, the E-wallet, to send only what he authorizes.

As the merchant’s application might attempt to exploit the E-wallet or its data, the user might attempt to alter the tags sent by the merchant in an attempt to dupe the merchant. The merchant relies on his application to protect him from such duping.
While domain names are very reliable, they can be spoofed.) While it is possible, even usual, for the meaning of the metadata to be stored in a separate object, local definition may override the global definition.

It supports “typed” data, that is, data types on which only a specified set of operations is legal. However, as with all properties of XML-defined data, it is the application, not the language itself that prevents arbitrary operations on the data. For example:

```xml
<simpleType name="nameType">
  <restriction base="string">
    <maxLength value="32"/>
  </restriction>
</simpleType>
```

sets the maximum length of “nameType” equal to 32. Similar metadata could impose other restrictions or define other attributes such as character set, case, set or range of valid values, decimal placement, or any other attribute or restriction.

XML and other tagged metadata languages are not tightly bound to the data. That is to say, anyone who is privileged to change the data may be privileged to change the metadata. Anyone who is privileged to change the tag can separate it from the data. This loose binding can be contrasted with a database in which changing the metadata requires a different set of privileges than changing the data itself (see Table 1).

### XML Capabilities and Limitations

Every tool has both capabilities, things that it can do, and limitations. The limitations may be inherent in the very concept of the tool (e.g., screwdrivers are not useful for driving nails) or they may be implementation induced (e.g., the handle of the screwdriver is not sufficiently bound to the bit). The tool may not be suitable for the application (e.g., the screwdriver is too large or too small for the screw). One does not use Howitzers to kill flies. This section discusses the capabilities, uses, misuses, abuses, and limitations of XML and similar metadata languages.

XML is metadata. It is data about data. Its role is similar to that of the schema in a database. Its fundamental role is to carry the identity, meaning, and intent of the data. It is neither a security tool nor is it intrinsically a vulnerability. From a security point of view, its intrinsic role is to support communication and reduce error. The potentially hostile or threatening aspects of XML are not those unique to it, but rather those that it shares with other languages and metadata, tagged and otherwise; a language that usually communicates truth can be used to lie.

People have been using and living with HTML for almost a decade. As XML is defined in XML, so is HTML 4.0, the vocabulary known as XHTML. (Recursion is often confusing and sometimes even scary.) People have been using EDI tags for almost a generation. Although they are now a subset of XML, all of our experience with them is still valid.

Perhaps the aspect of XML that is the source of most security concerns is that it is used with “push” technology, that is, the tags that describe the data come with the data. Moreover, the schema for interpreting the data may also be included. All of this often happens without much knowledge or intent on the part of the recipient or user. However, the meaning will be interpreted on the receiving system. Although it causes concern, it is as it should be. Only the sender of the data knows the intended meaning.

The fundamental responsibility for security in XML rests with the interpreter. As the browser hides the file system from HTML, the application must hide it from XML. As the browser decides how the HTML tag is to be rendered, so the application decides on the meaning of the XML tag. However, in doing so, it may rely on a called parser to help it deal with the tags. To the extent that the application relies on the parser, it must be sure that the one that it is using is the one that it expects.

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Web mail: An example</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Web mail” turns normal two-tier client/server e-mail into a three-tier client/server application. Perhaps the most well-known example is Microsoft’s Hotmail. However, other portals such as Excite and Yahoo! have their own implementations. Many Internet service providers have an implementation that permits their mail users to access their post office from an arbitrary machine, from behind a firewall (that permits HTTP but restricts mail), or from a public kiosk. In Web mail, the message is actually decoded and handled on the middle tier. Then the message is displayed to the user on his workstation by his Web browser. In one implementation, the middle tier failed to recognize the tags and simply passed them through to the Web browser. An attacker exploited this capability to use the browser to pop up a window labeled as the Web mail log-on window with prompts for the username and passphrase. Although mature users would not respond to a log-on prompt that they were not expecting, novice users did. Although all applications behaved as intended, the attacker used them to produce a result that duped the user. Web mail enabled the tags to escape the mail environment where they were safe, merely text, into the browser environment in which they were rendered in a misleading way. This exploit illustrates an important characteristic of languages like XML that is easy to overlook when discussing them: they are transparent to the end user. The end user does not even know that they exist, much less what they say, how they carry meaning to the user system, the user’s application, or to users themselves.</td>
<td></td>
</tr>
</tbody>
</table>
While normal practice permits a program to rely on the environment to vouch for the identity of a called program, good security practice may require that the application validates the identity of the parser, even to the extent of checking its digital signature.

Similar to many interpreted languages, XML can call escape mechanisms that permit it to pass instructions to the environment or context in which the user or receiver expects it to be interpreted. This may be the most serious exposure in XML, but it is not unique to XML. Almost all programming or data description languages include such an escape mechanism. These escape mechanisms have the potential to convert what the user thinks of as data into procedure (see Table 2).

While most of the use of such mechanisms will be benign, they have the potential to be used maliciously. The escape mechanisms included in Word, Excel, and Visual Basic have been widely exploited by viruses to get themselves executed, to get access to storage in which to place replicas, and to display misleading information to the user.

**WORLD WIDE WEB SECURITY**

While XML will have many applications other than the World Wide Web, this is the application of both interest and importance. As discussed, XML does little to aggravate the security of the Web. It is true that it can be used to dupe both users and applications. However, the vulnerabilities that are exploited can as easily be exploited using other languages or methods. By making the intent and meaning of the data more explicit, it may facilitate intelligence gathering.

On the other hand, it has the potential to improve communication and reduce errors. XML is being used to extend the capabilities of web clients and servers so as to increase the security of their applications. While these capabilities might be achieved in a variety of other ways, they are being implemented using XML. That they are being implemented using a metadata language demonstrates one value of such languages. These implementations have the potential to bring to security many of the advantages of metadata languages, including interoperability that is both platform and transport independent. However, keep in mind that these definitions are about the use of XML for security rather than about the security of XML.

**Control of Access to XML Objects**

One such application is the control of access to documents or arbitrary objects stored on web servers in a manner that is analogous to the control of access to database objects. In client/server applications, XML can be analogous to an SQL request. That is, it is used to specify the data that is being requested. As the database server limits access to the data that it stores and serves up, the server responding to an XML request can control access to the data that it serves.

In SQL, the fundamental object of request and control is a table. However, most database servers will also provide more granular control. For example, they may provide for discretionary access control over rows, columns, or even cells. Many can exercise control over arbitrary combinations of data called views. Notice that discretionary access control over the data is a feature of the database manager rather than of the language or schema. Notice also that the data is bound to the schema only when it is in a database manager. Once the data is served up by the database manager, trusted paths and processes may be required to preserve its integrity.

In XML, as in HTML, the fundamental object of access control is the document. For this purpose, the document is analogous to the database table. Almost all servers can restrict access to some pages. While this capability is rarely used, many provide discretionary access control to pages, that is, the ability to grant some users access to a page while denying it to others. For example, the Apache Web server permits the manager to grant or restrict access to named documents to specified users, user groups, IP addresses, or address/user pairs. Notice that as a database administrator can exercise more granular access control by naming multiple views of the same data, so too can the administrator of a server exercise more granular control by creating multiple documents.

However, tags are used to specify more granular objects than documents. This raises the possibility of more granular access control. As a database manager may provide more granular access control than a table, a server may provide more granular access control than a page. If it is going to do this at all, it can do it to the level of any tagged object. While administratively one might prefer large objects, from the perspective of the control mechanism, one tag looks pretty much like any other. Damiani et al.\(^1\) have demonstrated such a mechanism.

**Process-to-Process Authentication**

On the web, particularly in E-commerce applications, it is often necessary for a client process to demonstrate its identity to a server process. These *bona fides* are often obtained from a trusted third party or parties. Such a demonstration may involve the exchange of data in such a way that the credentials cannot be forged or replayed. The protocols for such exchanges are well worked out. These protocols lend themselves to being described in structured data. In XML, such exchanges involve two
schemas: one for the credentials themselves and another for requesting them.

A dialect of XML, authXML, has been proposed for this application. It defines formats for data to assert a claim of identity and for evidence to support that claim.

**Process-to-Process Integrity**

Similarly, in E-commerce applications, it is necessary to be able to digitally sign transactions so as to demonstrate their origin and content. This requires tags for the transaction itself, the signature, and the certificate. S²ML, the Security Services Markup Language, provides a common language for the sharing of security services between companies engaged in B2B and B2C transactions.

**RECOMMENDATIONS**

1. **Identify and tag your own data.** Keep tags with your data. Although useful and used for communication, metadata is primarily for the use of the owners of the data.
2. **Bind your metadata to your data.** Use database managers, access-controlled storage, encryption, trusted applications, trusted systems, and trusted paths.
3. **Verify what you rely on.** This is the fundamental rule of security in the modern networked world. If relying on an object description, then be sure that you are using that description. If you are relying on an object not to have a script hidden in it, then be sure to scan for scripts.
4. **Accept tags only from reliable sources.** Do not place more reliance on tags from a source than you would on any other data from that source. While you might reject data without tags from a source, do not accept data with tags where you might not accept the data without the tags.
5. **Reject data with unexpected tags.** Do not pass the tags on. Do not strip them off and pass the data on.
6. **Include tags in logs and journals.** Not only will this improve the integrity and usability of the logs and journals, but it will improve accountability.
7. **Use the security tags where indicated and useful.**
8. **Communicate these recommendations to application developers and managers in appropriate standards, procedures, and enforcement mechanisms.** Although these measures are essential to the safe use of metadata, their use and control is usually in the hands of those with other priorities.
9. **Focus on the result seen by the end user.** After all is said and done, the security of the application will reside in what the end user understands and does.

**CONCLUSION**

HyperText Markup Language (HTML) and similar metadata languages have given us levels of interoperability that were not dreamed of a decade ago. As the number of interoperable systems on the Internet has risen linearly, the value to the users has risen exponentially. Extensible Markup Language (XML) promises us another order-of-magnitude increase in that interoperability. Not only will it help create interoperability between clients and servers on the Internet, but it will also improve interoperability among arbitrary objects and processes wherever located. By conserving and communicating the meaning and intent of data, it will increase its utility and value. Not since the advent of Common Business-Oriented Language (COBOL) has there been a tool with such promise; this promise is far more likely to be realized and may be realized on a grand scale.

However, as with any new tool, the value of XML will depend, in large part, on one’s skill in using it. As with any idea, its value will depend on one’s understanding of it. As with any new technology, its value may be limited by fear and ignorance.

As with any new tool, one must understand both its capabilities and its limitations. Few things in information technology have caused as many problems as using tools without proper regard for their limitations.

Although the use of XML will often be outside the purview of the information security professional, hardly anyone else will be concerned about its limitations, misuse, or abuse. If the enterprise suffers losses because of limitations, misuse, or abuse, it is likely to hold us accountable. If the fundamental idea should become tarnished because of such limitations, misuse, or abuse, we will all be poorer for it.

**REFERENCE**