

1 Introduction to XML

XML (Extensible Markup Language) is an evolving and open standard that is a data-storage toolkit, a configurable vehicle for any kind of information. The open source movement has led to an explosion of software, and a consistent interface is a necessity; this is supplied by XML. XML can store and organize almost any kind of information in a form tailored to one's needs. It supports a large number of scripts and symbols since its standard character set is Unicode, which includes all the ASCII characters and many more: Chinese ideograms, Arabic letters in their various forms, a host of mathematical symbols, and much else. XML allows many ways to check a document including syntax rules, internal link checking, comparison to document models, and datatyping. Since it has a clear, simple syntax, XML is easy to parse for humans and programs. Finally, XML is easily combined with stylesheets to create formatted documents in any style.

This part is an overview of what XML involves, how it evolved, its goals, the standards process that controls its development, and the family of specifications of which it is a part. We do not begin systematically describing XML until the next part.

1.1 Aspects of XML

At the most specific level, XML is a convention for containing and managing information with markup. It is part of a family of technologies to do everything from formatting documents to filtering data. Most generally, XML represents a philosophy for information handling that seeks maximum usefulness and flexibility for data by refining it to its purest and most structured form.

The XML specification contains a set of rules for building markup languages. A *markup language* is a set of symbols that can be placed in the text of a document to demarcate and label the parts of the document. Markup tags convey information in several ways. An opening-closing tag pair defines the *boundaries* of a collection of text and markup. *Roles* are indicated by the particular tag names. A tag pair also defines the *position* a piece of text occupies: it comes before some things and after others. Tag pairs can be nested hence indicate *containment*. Finally, *relationships* can be indicated in tags since a piece of text can be linked to a resource somewhere else, as with HTML's `href` attribute.

A *document* is a basic unit of XML information, composed of elements and other markup in an orderly package. It can contain text (such as an article) but could contain, for example, a database of numbers or some abstract structure representing a molecule or equation. One of the most promising applications of XML is as a format for application-to-application data exchange.

A document is composed of *elements*, which are nested. At the top level is the *document element* or *root element*. The following is a document in MathML (Mathematics Markup Language), an application of XML used to encode mathematical formulas. This example encodes the equation

$$F = G \frac{Mm}{r^2}$$

The root element is `math` and the namespace is defined by the URL in the root tag; all the other tags have meaning within this namespace. An `mi` tag indicates an identifier (a

variable or constant), `mo` indicates an operator, and `mn` indicates a number. The other tags are largely self-explanatory. Note that, where two symbols are normally juxtaposed to indicate multiplication, we must explicitly indicate an operator since we define the structure of the data, not just its presentation. While one application might use this input to display the equation, another might use it to solve the equation with a series of values.

```
<?xml version = ?1.0??>
<math xmlns = ?http://www.w3.org/TR/REC-MathML/?>
  <mi>F</mi>
  <mo>=</mo>
  <mi>G</mi>
  <mo>&InvisibleTimes;</mo>
  <mfrac>
    <mrow>
      <mi>M</mi>
      <mo>&InvisibleTimes;</mo>
      <mi>m</mi>
    </mrow>
    <apply>
      <power>
        <mi>r</mi>
        <mn>2</mn>
      </power>
    </apply>
  </mfrac>
</math>
```

The next document is in SVG (Scalable Vector Graphics), an application of XML used to draw resizable line art; this document defines a picture with a text description and three shapes. The `!DOCTYPE` tag identifies the document model that this example must conform to.

```
<?xml version = ?1.0? standalone = ?no??>
<!DOCTYPE svg
  PUBLIC ?-//W3C//DTD SVG 20001102//EN?
  http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd>
<svg>
  <desc>Three shapes</desc>
  <rect fill=?green? x=?1cm? y=?1cm?
    width=?3cm? height=?3cm?/>
  <circle fill=?red? cx=?3cm? cy=?2cm? r=?4cm?/>
  <polygon fill=?blue?
    points=?110,160, 50,300, 180,290?/>
</svg>
```

The above two examples are based on already established markup languages. You can, however, define your own XML-based language. For example, the following uses fabricated element names with the obvious meanings to construct a message. We assume that a graphic tag can have a `fileref` attribute whose value is the URL of a graphics file.

```

<?xml version = "1.0"?>
<message>
  <exclamation>Hello, world!</exclamation>
  <paragraph>XML is <emphasis>fun</emphasis> and
    <emphasis>easy</emphasis> to use.
    <graphic fileref=?smiley_face.pict?/></paragraph>
</message>

```

Note that a document is a *logical* structure and is not the same as a file, which is a *physical* structure. A file is a package of data treated as a contiguous unit by the operating system, an XML document can exist in one file or in many files and uses special markup to integrate the contents of different files to create a single entity.

There are two ways to create a language based on XML. The first is by using *freeform XML*—there are some minimal rules about how to form and use tags, but any tag names can be used and they can appear in any order. A document satisfying the minimal rules of XML is called *well-formed*.

The other, and more interesting, way to create a language based on XML is by *document modeling*, which involves creating a specification that lays out the rules for document structure. Such a specification is a model against which you can compare a given document (a *document instance*) to check that it represents your language. The test is called *validation*, and a document passing the test is called *valid*. The most common way to model documents is with a *document type definition* (DTD)—a set of declarations specifying which tags can be used and what they can contain. At the top of a document is a reference to the DTD, declaring the desire to have the document validated. A more recent document-modeling standard is *XML schema*. Schemas use XML fragments called templates to demonstrate how a document should look. Schemas themselves are a form of XML so can be edited with the same tools used to edit documents, and they introduce more powerful datatype checking. A markup language created using XML rules is called an XML *application* or sometimes a *document type*.

Presentation describes how a document should look when prepared for human viewing. An XML author assigns styles in a separate location, a document called a *stylesheet*. We can design a markup language that, like HTML, mixes style information with “pure” markup. But, instead of incorporating style information in a tag, we would like to keep the style information for each tag in a stylesheet. For example, if we have a tag `<emphasis>`, then, to change emphatic phrases from italic to bold, we need only edit one line in the stylesheet. One can have as many different tags as there are types of information in a document. Keeping style out of the document enhances representation possibilities, and, since any number of stylesheets can be applied to a document, we can create different versions on the fly.

Reading an XML document and doing something with it is called *processing* the document; a program that does this is an *XML processor*. These include validity checkers, web browsers, XML editors, and data and archiving systems, but the most fundamental XML processor is a parser. It turns a stream of characters into tokens, which are either interpreted as events to drive a program or are built into a tree representation in memory for a program to act on. XML parsers are notoriously strict.

1.2 Origins of XML

Early electronic formats were more concerned with presentation than with document structure and meaning. (This concern is carried on with TeX and the popular interface to it, LaTeX, itself a markup language.) It was hard to write programs to filter data, to cross-reference, or to re-purpose documents for different applications. *Generic coding* uses descriptive tags rather than formatting codes. An example of this is found in IBM's GML (Generalized Markup Language) project, which addressed the problem of encoding documents for use with multiple information subsystems. IBM has made extensive use of GML – esp. for publishing technical manuals.

The ANSI committee on Information Processing assembled a team to develop a standard text-description language based on GML. The industry standard called Standard Generalized Markup Language (SGML – 1983) was ratified by the ISO in 1986. SGML was designed to be a flexible and all-encompassing coding scheme. Like XML, it is basically a toolkit for describing specialized markup languages. SGML, however, is much bigger than XML, with a looser syntax and lots of esoteric parameters. In fact, it is so flexible that software to process it is complex and expensive.

The public revolution in generic coding happened in the early 1990s, when Berners-Lee and Berlund at CERN developed an SGML document type for hypertext documents that was compact and efficient; this was HTML. HTML, however, is in some ways a step backward. To achieve the simplicity needed to be useful, some principles of generic coding were sacrificed. For example, one document type is used for all purposes, forcing us to overload tags rather than define specific-purpose tags. Also, many of the tags are purely presentational.

To return to the ideals of generic coding, some tried to adapt SGML for the Web, but this was too hard since SGML is too big for a browser. A smaller language that retained the generality of SGML was required—XML.

1.3 The Goals of XML

The World Wide Web Consortium (W3C) began work in the mid-1990s on a markup language that combined the flexibility of SGML with the simplicity of HTML. The following tenets have guided the development XML.

Application-Specific Markup Languages. Using XML, you can make up your own markup language to express your information in the best way possible, and your syntax rules will be enforced.

Unambiguous Structure. XML requires strict adherence to its syntax, which vastly reduces errors and code complexity. Besides the basic syntax, you can create your own rules with a DTD or a schema, which allows almost endless possibilities for error-checking and structure control are incredible.

Presentation Stored Elsewhere. The following are some of the benefits of stylesheets, which store style information externally:

- The same style settings can be used for many documents.
- To change a style setting, fix it in one place, and all documents are affected.
- You can swap stylesheets for different purposes.
- The document's content and structure is intact no matter how you play with the presentation.

- The document's content is not cluttered with the vocabulary of style.
- You can choose names that precisely reflect the purpose of items. This simplifies editing and transformation.

Keep It Simple. Simplicity leads to wide acceptance and benefits application development, so there are more and cheaper programs available to the public. Since XML is behind many kinds of information expression, you can use the same tools to edit and process many technologies.

Maximum Error Checking. Some markup languages (such as HTML) are so lenient about syntax that errors go undiscovered. When errors build up in a file, it no longer behaves as we want.

1.4 The Standards Process

The best standards are *open* (as opposed to *proprietary*)—not controlled or owned by any one company. XML is an open standard managed by the W3C as a formal recommendation. But it isn't strictly binding: there is no certification process, no licensing agreement, and no punishment for those who fail to implement XML correctly except community disapproval. A loosely binding recommendation is useful in that standards enforcement takes time and resources no consortium member wants to spend, and it allows developers to create their own extensions or to make partially working implementations that do most of the job pretty well. The down side that there is no guarantee that anyone will do a good job. For example, the CSS standard has languished for years because browser manufacturers cannot be bothered to implement it fully.

The W3C, the unofficial smithy of the Web, was founded in 1994 by several organizations and companies around the world with vested interest in the Web. They help banish the chaos of competing, half-baked technologies by issuing technical documents and recommendations.

A recommendation begins as a project, or activity, when the W3C Director is sent a formal proposal called a briefing package. If approved, the activity gets its own working group, which issues progress reports, posted on a web page—these are working drafts. When a draft is submitted for public evaluation, it becomes a candidate recommendation, with a deadline for receiving comments; it benefits from expert insight and implementations of parts. The Director can bless the recommendation into a proposed recommendation, which is scrutinized by the Advisory Council. Until the final recommendation is released (which can take years), everything can change overnight.

Visit the W3C's web site (<http://www.w3.org>) off and on for information about evolving standards, links to tutorials, and pointers to tools.

1.5 Satellite Technologies

XML is technically a set of rules for creating your own markup language as well as for reading and writing documents in a markup language. Other specifications such as CSS (with its own W3C formal specification) complement it. In fact, there is an entire family of specifications allied with XML, many of which we shall cover in detail. Here we list them under the major categories.

Core syntax includes standards contributing to the basic XML functionality, such as

- the XML specification itself,
- namespaces (a way to combine document types), and

- XLinks (a language for linking documents).

XML applications include some useful XML-derived markup languages, such as

- XHTML (an XML-compatible version of HTML) and
- MathML (a mathematical equation language).

Document modeling includes the structure-enforcing languages DTDs and XML schema.

Data addressing and querying include specifications for locating documents and data within them, such as

- XPath (describing paths to data inside documents),
- XPointer (a way to describe locations of files on the Internet), and
- XQL (XML Query Language – a database access language).

Styles and transformations include languages to describe presentation and ways to mutate documents into new forms, including

- XSL (XML stylesheet language),
- XSLT (the XSL Transformation Language),
- XSL-FO (the Extensible Stylesheet Language for Formatting Objects), and
- CSS

Programming and infrastructure include interfaces for accessing and processing XML-encoded information, including

- DOM (the Document Object Model, a generic programming interface),
- the XML Information Set (a language for describing the contents of documents),
- the XML Fragment Interchange (describing how to split documents for transport across networks), and
- SAX (the Simple API for XML, a programming interface to process XML data).