|  |  |  |
| --- | --- | --- |
|  | **Contents** | **Page No:** |
| **1** | **Introduction** | **1** |
| **2** | **History** | **3** |
| **3** | **List of Constructs Which appear in XML** | **4** |
| **4** | **Difference between XML and HTML** | **6** |
| **5** | **Entities** | **7** |
| **6** | **Comments** | **7** |
| **7** | **XML Elements** | **7** |
| **8** | **XML Documents** | **9** |
| **9** | **XML Data Model** | **12** |
| **10** | **DTD** | **13** |
| **11** | **XML Parsing Methodologies** | **15** |
|  | * **SAX** | **15** |
|  | * **DOM** | **16** |
| **12** | **Conclusion** | **19** |
| **13** | **Reference** | **20** |

**eXtensible Markup Language** (**XML**)

**Introduction to XML**

**Extensible Markup Language** (**XML**) is a [markup language](http://en.wikipedia.org/wiki/Markup_language) that defines a set of rules for encoding documents in a [format](http://en.wikipedia.org/wiki/File_format) that is both [human-readable](http://en.wikipedia.org/wiki/Human-readable_medium) and [machine-readable](http://en.wikipedia.org/wiki/Machine-readable). The design goals of XML emphasize simplicity, generality, and usability over the [Internet](http://en.wikipedia.org/wiki/Internet). It is a textual data format with strong support via [Unicode](http://en.wikipedia.org/wiki/Unicode) for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary [data structures](http://en.wikipedia.org/wiki/Data_structures), for example in [web services](http://en.wikipedia.org/wiki/Web_service).

Many [application programming interfaces](http://en.wikipedia.org/wiki/Application_programming_interfaces) (APIs) have been developed for software developers to use to process XML data, and several [schema systems](http://en.wikipedia.org/wiki/XML_schema) exist to aid in the definition of XML-based languages.

As of 2009[[update]](http://en.wikipedia.org/w/index.php?title=XML&action=edit), hundreds of XML-based languages have been developed, including [RSS](http://en.wikipedia.org/wiki/RSS), [Atom](http://en.wikipedia.org/wiki/Atom_%28standard%29), [SOAP](http://en.wikipedia.org/wiki/SOAP), and [XHTML](http://en.wikipedia.org/wiki/XHTML). XML-based formats have become the default for many office-productivity tools, including [Microsoft Office](http://en.wikipedia.org/wiki/Microsoft_Office) ([Office Open XML](http://en.wikipedia.org/wiki/Office_Open_XML)), [OpenOffice.org](http://en.wikipedia.org/wiki/OpenOffice.org) (Open Document), and [Apple](http://en.wikipedia.org/wiki/Apple_Computer)'s [iWork](http://en.wikipedia.org/wiki/IWork). XML has also been employed as the base language for [communication protocols](http://en.wikipedia.org/wiki/Communication_protocol), such as [XMPP](http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol).

XML is a new type of language which has been developed for the web which is different to any other type of scripting or programming language available before. Instead of being concerned with the processing and display of data, XML's primary purpose is to tell the computer what data entered actually means.

XML and HTML have a similar syntax … both derived from SGML. The Standard Generalized Markup Language (SGML) is a language for defining markup languages. HTML is one such "application" of SGML.

An SGML application consists of several parts:

1. The SGML declaration. The SGML declaration specifies which characters and delimiters may appear in the application.
2. The document type definition (DTD). The DTD defines the syntax of markup constructs. The DTD may include additional definitions such as numeric and named character entities.
3. A specification that describes the semantics to be ascribed to the markup. This specification also imposes syntax restrictions that cannot be expressed within the DTD.
4. Document instances containing data (contents) and markup. Each instance contains a reference to the DTD to be used to interpret it.

**History**

XML is an application profile of [SGML](http://en.wikipedia.org/wiki/SGML) (ISO 8879). The versatility of [SGML](http://en.wikipedia.org/wiki/SGML) for dynamic information display was understood by early digital media publishers in the late 1980s prior to the rise of the Internet. By the mid-1990s some practitioners of SGML had gained experience with the then-new [World Wide Web](http://en.wikipedia.org/wiki/World_Wide_Web), and believed that SGML offered solutions to some of the problems the Web was likely to face as it grew. [Dan Connolly](http://en.wikipedia.org/wiki/Dan_Connolly) added SGML to the list of W3C's activities when he joined the staff in 1995; work began in mid-1996 when Sun Microsystems engineer [Jon Bosak](http://en.wikipedia.org/wiki/Jon_Bosak) developed a charter and recruited collaborators. Bosak was well connected in the small community of people who had experience both in SGML and the Web.

XML was compiled by a [working group](http://en.wikipedia.org/wiki/Working_group) of eleven members, supported by an (approximately) 150-member Interest Group. Technical debate took place on the Interest Group mailing list and issues were resolved by consensus or, when that failed, majority vote of the Working Group. A record of design decisions and their rationales was compiled by [Michael Sperberg-McQueen](http://en.wikipedia.org/wiki/Michael_Sperberg-McQueen) on December 4, 1997. [James Clark](http://en.wikipedia.org/wiki/James_Clark_%28XML_expert%29) served as Technical Lead of the Working Group, notably contributing the empty-element "<empty />" syntax and the name "XML". Other names that had been put forward for consideration included "MAGMA" (Minimal Architecture for Generalized Markup Applications), "SLIM" (Structured Language for Internet Markup) and "MGML" (Minimal Generalized Markup Language). The co-editors of the specification were originally [Tim Bray](http://en.wikipedia.org/wiki/Tim_Bray) and [Michael Sperberg-McQueen](http://en.wikipedia.org/wiki/Michael_Sperberg-McQueen). Halfway through the project Bray accepted a consulting engagement with [Netscape](http://en.wikipedia.org/wiki/Netscape_Communications_Corporation), provoking vociferous protests from Microsoft. Bray was temporarily asked to resign the editorship. This led to intense dispute in the Working Group, eventually solved by the appointment of Microsoft's [Jean Paoli](http://en.wikipedia.org/wiki/Jean_Paoli) as a third co-editor.

The XML Working Group never met face-to-face; the design was accomplished using a combination of email and weekly teleconferences. The major design decisions were reached in a short burst of intense work between August and November 1996, when the first Working Draft of an XML specification was published. Further design work continued through 1997, and XML 1.0 became a [W3C](http://en.wikipedia.org/wiki/W3C) Recommendation on February 10, 1998

**List of Constructs Which Appear In XML**

The material in this section is based on the XML Specification. This is not an exhaustive list of all the constructs which appear in XML; it provides an introduction to the key constructs most often encountered in day-to-day use.

**(Unicode) Character**

By definition, an XML document is a string of characters. Almost every legal [Unicode](http://en.wikipedia.org/wiki/Unicode) character may appear in an XML document.

**Processor and Application**

The processor analyzes the markup and passes structured information to an application. The specification places requirements on what an XML processor must do and not do, but the application is outside its scope. The processor (as the specification calls it) is often referred to colloquially as an XML parser.

**Markup and Content**

The characters which make up an XML document are divided into markup and content. Markup and content may be distinguished by the application of simple syntactic rules. All strings which constitute markup either begin with the character < and end with a >, or begin with the character & and end with a ;. Strings of characters which are not markup are content.

**Tag**

A markup construct that begins with < and ends with >. Tags come in three flavors:

* start-tags; for example: <section>
* end-tags; for example: </section>
* empty-element tags; for example: <line-break />

**Element**

A logical document component either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example of an element is <Greeting>Hello, world. </Greeting> (see [hello world](http://en.wikipedia.org/wiki/Hello_world_program)). Another is <line-break />.

**Attribute**

A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example (below) the element img has two attributes, src and alt: <img src="madonna.jpg" alt='Foligno Madonna, by Raphael' />. Another example would be <step number="3">Connect A to B.</step> where the name of the attribute is "number" and the value is "3".

**XML Declaration**

XML documents may begin by declaring some information about themselves, as in the following example:

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

**Encoding detection**

The Unicode character set can be encoded into bytes for storage or transmission in a variety of different ways, called "encodings". Unicode itself defines encodings that cover the entire repertoire; well-known ones include [UTF-8](http://en.wikipedia.org/wiki/UTF-8) and [UTF-16](http://en.wikipedia.org/wiki/UTF-16). There are many other text encodings that pre-date Unicode, such as [ASCII](http://en.wikipedia.org/wiki/ASCII) and [ISO/IEC 8859](http://en.wikipedia.org/wiki/ISO/IEC_8859); their character repertoires in almost every case are subsets of the Unicode character set.

XML allows the use of any of the Unicode-defined encodings, and any other encodings whose characters also appear in Unicode. XML also provides a mechanism whereby an XML processor can reliably, without any prior knowledge, determine which encoding is being used. Encodings other than UTF-8 and UTF-16 will not necessarily be recognized by every XML parser.

**The Main Differences between XML and HTML**

**XML is designed to carry data.**

XML describes and focuses on the **data** while HTML only displays and focuses on **how data looks**. HTML is all about **displaying** information but XML is all about **describing** information. In current scenario XML is the most common tool for data manipulation and data transmission.

XML is used to store data in files and for sharing data between diverse applications. Unlike HTML document where data and display logic are available in the same file, XML hold only data. Different presentation logics could be applied to display the xml data in the required format. XML is the best way to exchange information.

**XML is Free and Extensible**

XML tags are not predefined. User must "invent" his tags. The tags used to mark up HTML documents and the structure of HTML documents are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like <p>, <h1>, etc.). XML allows the user to define his own tags and document structure.

**XML Tags are Case Sensitive**

Unlike HTML, XML tags are case sensitive. In HTML the following will work:

<Message>This is incorrect</message>

In XML opening and closing tags must therefore be written with the same case:

<message>This is correct</message>

**XML Elements Must be Properly Nested**

Improper nesting of tags makes no sense to XML.In HTML some elements can be improperly nested within each other like this:

<b><i>This text is bold and italic</b></i>

In XML all elements must be properly nested within each other like this:

<b><i>This text is bold and italic</i></b>

**XML is a Complement to HTML**

XML is not a replacement for HTML***.*** It is important to understand that XML is not a replacement for HTML. In Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data.

**Entities**

An entity is a symbolic representation of information. What does that mean? There are five predefined entities:

* &lt; represents "<"
* &gt; represents ">"
* &amp; represents "&"
* &apos; represents '
* &quot; represents "

**Comments**

Comments may appear anywhere in a document outside other markup. Comments cannot appear before the XML declaration. The string "--" (double-hyphen) is not allowed inside comments. Comments start with "<!--" and end with "-->". The ampersand has no special significance within comments, so entity and character references are not recognized as such, and there is no way to represent characters outside the character set of the document encoding. An example of a valid comment: "<!-- no need to escape <code> & such in comments -->"

**XML Elements**

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

* other elements
* text
* attributes
* Or a mix of all of the above...

<bookstore>  
  <book category="CHILDREN">  
 <title>Harry Potter</title>  
    <author>J K. Rowling</author>  
    <year>2005</year>  
    <price>29.99</price>  
  </book>  
<book category="WEB">  
    <title>Learning XML</title>  
    <author>Erik T. Ray</author>  
    <year>2003</year>  
    <price>39.95</price>  
  </book>  
</bookstore>

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

XML Naming Rules

XML elements must follow these naming rules:

* Names can contain letters, numbers, and other characters
* Names cannot start with a number or punctuation character
* Names cannot start with the letters xml (or XML, or Xml, etc)
* Names cannot contain spaces

Any name can be used, no words are reserved.

**Best Naming Practices**

Make names descriptive. Names with an underscore separator are nice: <first\_name>, <last\_name>.

Names should be short and simple, like this: <book\_title> not like this: <the\_title\_of\_the\_book>.

Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.

Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."

Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).

XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

**XML Elements are Extensible**

XML elements can be extended to carry more information.

Look at the following XML example:

<note>  
<to>Tove</to>  
<from>Jani</from>

<body>Don't forgetme this weekend!</body>  
</note

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

**MESSAGE**

**To:** Tove  
**From:** Jani

Don't forget me this weekend!

**XML Attributes**

Attributes are used to specify additional information about the element. It may help to think of attributes as a means of specializing generic elements to fit your needs. An attribute for an element appears within the opening tag.

If there are multiple values an attribute may have, then the value of the attribute must be specified. For example, if a tag had a color attribute then the value would be red,blue,green etc. The syntax for including an attribute in an element is:

<element attributeName=”value”>

In this example we will be using a madeup XML element named “friend” that has an optional attribute age.

<friend age=”23”>Samantha</friend>

**XML Documents**

Wel-formedness is essential in XML. The W3C instructs us that "violations of well-formedness constraints are fatal errors." Documents that are not well-formed will not load in a browser or will not be processed by an XML parser, according to the XML Recommendation.

**Five rules for well-formed documents**

Five basic rules will help you construct well-formed XML documents. You should commit these rules to memory:

1. XML uses elements to markup content.

XML elements consist of a start tag and an end tag. Start tags begin with < and end with >. End tags begin with </ and end with >. Element names in XML are case sensitive. They may start with a letter, an underscore character, or a colon character. The next characters in an element name may be letters, digits, underscores, hyphens, periods, and colons but not white space. Spaces, carriage returns, line feeds, and tabs are all treated as white space in XML.

2. [Tags cannot be inferred; they must be explicit](http://www.ewebprogrammer.com/xml-programming/module3/xml-start-end-tags.jsp).

All start tags must have corresponding ending tags. All ending tags must have corresponding start tags.  
For the document to be well-formed, it must be written in the following way:

<NAME><FIRST>John</FIRST></NAME>

3. An empty element must be closed with />.

Empty elements may be used for elements that have no content.  
You may be familiar with the <IMG> and <BR> empty tags from HTML. In HTML, empty tags are not required to have closing tag in the form />. In XML, empty elements must be closed with />. For example

<PURCHASE-ORDER NUMBER="1234"/>

1. XML elements that have name-value pair attributes must enclose [attribute values](http://www.ewebprogrammer.com/xml-programming/module3/attribute-values-xml-documents.jsp) in single or double quotation marks.

For example:

<BOOK ISBN="345671">

<AUTHOR>James Gosling</AUTHOR>

</BOOK>

In this example, the <BOOK> element has an attribute named ISBN with the value 345671.  
Note that the attribute value is enclosed in double quotation marks for the document to be well-formed.

1. XML elements must nest and un-nest in reverse order.

For example, the following XML document is not well-formed because it violates the rule for correctly nesting elements:

<NAME><FIRST>John</NAME></FIRST>

The correct nesting of these elements would be:

<NAME><FIRST>John</FIRST></NAME>

**Data Islands**  
  
this is the way of formatting XML is to use Data Islands. Currently, only Internet Explorer 5 and upwards support this, and it is an unofficial standard. Again, I will use the same XML to demonstrate this. Using this method, you use the unofficial or you can embed a remote file.  
  
To embed data straight into the file you use the folloing format:  
  
<xml id="emails">  
XML code goes in here but without first declaration line  
</xml>  
  
To embed XML from a remote file use:  
  
<xml id="emails" src="emails.xml">  
</xml>   
  
As you will have noticed, you must give an ID to your XML.  
  
Now you have got the XML data into the file, you can format it by normal HTML, but using tags to insert particular fields. This is an example of formatting the e-mail file:  
  
<html>  
<body>  
  
<xml id="emails" src="emaildata.xml"></xml>  
<table bgcolor= "#EEEEEE" border="0" datasrc="#emails">  
<tr bgcolor="#CCCCCC"><td>To: <span datafld="to"></span></td></tr>  
<tr bgcolor="#CCCCCC"><td>From: <span datafld="from"></span></td></tr>  
<tr bgcolor="#CCCCCC"><td><b>Subject: <span datafld="subject"></span></b></td></tr>  
<tr><td><span datafld="body"></span></td></tr>  
</table>  
</body>  
  
Although I used the same XML data for this as for all the others, I removed the <header> item as the data objects only appear to work on the first level of the document.

**XML Data Model**

The data model for XML is very simple - or very abstract, depending on one's point of view. XML provides no more than a baseline on which more complex models can be built. All those more restricted applications will share some common invariants, however, and it is those that are given below.

Think of an XML document as a linearization of a tree structure. At every node in the tree there are several character strings. The tree structure and the character strings together form the information content of an XML document. Almost everything will follow naturally from that. Some of the characters in the document are only there to support the linearization, others are part of the information content..

## A tree and a graph overlaid

The main structure of an XML document is tree-like, and most of the lexical structure is devoted to defining that tree, but there is also a way to make connections between arbitrary nodes in a tree. For example, in the following document there is a root node with three children, but one of the children has a link to one of the other children:

<p>

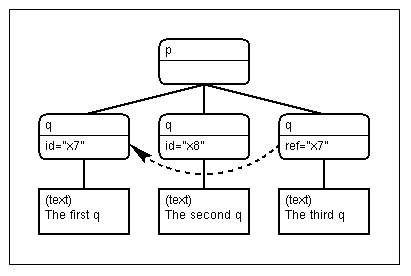
<q id="x7">The first q</q>

<q id="x8">The second q</q>

<q href="#x7">The third q</q>

</p>

The tree corresponding to this document can be visualized as follows:



The last q has an `href' attribute and it points to an element with an `id.' In this case the first q has an id with the same value as the href (minus the `#'), so the third q points to the first. (Note that this is a generalization of a similar mechanism in HTML

**Schemas and validation**

In addition to being well-formed, an XML document may be valid. This means that it contains a reference to a [Document Type Definition (DTD)](http://en.wikipedia.org/wiki/Document_Type_Definition), and that its elements and attributes are declared in that DTD and follow the grammatical rules for them that the DTD specifies.

XML processors are classified as validating or non-validating depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue normal processing.

A DTD is an example of a [schema](http://en.wikipedia.org/wiki/XML_schema) or grammar. Since the initial publication of XML 1.0, there has been substantial work in the area of schema languages for XML. Such schema languages typically constrain the set of elements that may be used in a document, which attributes may be applied to them, the order in which they may appear, and the allowable parent/child relationships.

**DTD**

Main article: [Document Type Definition](http://en.wikipedia.org/wiki/Document_Type_Definition)

The oldest schema language for XML is the [Document Type Definition](http://en.wikipedia.org/wiki/Document_Type_Definition) (DTD), inherited from [SGML](http://en.wikipedia.org/wiki/SGML).

DTDs have the following benefits:

* DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.
* DTDs are terse compared to element-based schema languages and consequently present more information in a single screen.
* DTDs allow the declaration of [standard public entity sets](http://en.wikipedia.org/wiki/SGML_entity) for publishing characters.
* DTDs define a *document type* rather than the types used by a namespace, thus grouping all constraints for a document in a single collection.

DTDs have the following limitations:

* They have no explicit support for newer [features](http://en.wikipedia.org/wiki/Feature_%28software_design%29) of XML, most importantly [namespaces](http://en.wikipedia.org/wiki/XML_Namespace).
* They lack expressiveness. XML DTDs are simpler than SGML DTDs and there are certain structures that cannot be expressed with regular grammars. DTDs only support rudimentary datatypes.
* They lack readability. DTD designers typically make heavy use of parameter entities (which behave essentially as textual [macros](http://en.wikipedia.org/wiki/Macro_%28computer_science%29)), which make it easier to define complex grammars, but at the expense of clarity.
* They use a syntax based on [regular expression](http://en.wikipedia.org/wiki/Regular_expression) syntax, inherited from [SGML](http://en.wikipedia.org/wiki/SGML), to describe the schema. Typical XML APIs such as [SAX](http://en.wikipedia.org/wiki/Simple_API_for_XML) do not attempt to offer applications a structured representation of the syntax, so it is less accessible to programmers than an element-based syntax may be.

Two peculiar features that distinguish DTDs from other schema types are the syntactic support for embedding a DTD within XML documents and for defining *entities*, which are arbitrary fragments of text and/or markup that the XML processor inserts in the DTD itself and in the XML document wherever they are referenced, like character escapes.

DTD technology is still used in many applications because of its ubiquity..

**Use on the Internet**

XML has come into common use for the interchange of data over the Internet. [RFC 3023](http://tools.ietf.org/html/rfc3023) gives rules for the construction of [Internet Media Types](http://en.wikipedia.org/wiki/Internet_media_type) for use when sending XML. It also defines the media types application/xml and text/xml, which say only that the data are in XML, and nothing about its [semantics](http://en.wikipedia.org/wiki/Semantics). The use of text/xml has been criticized as a potential source of encoding problems and it has been suggested that it should be deprecated.

[RFC 3023](http://tools.ietf.org/html/rfc3023) also recommends that XML-based languages be given media types beginning in application*/* and ending in *+*xml; for example application/svg+xml for [SVG](http://en.wikipedia.org/wiki/SVG).

Further guidelines for the use of XML in a networked context may be found in [RFC 3470](http://tools.ietf.org/html/rfc3470), also known as IETF BCP 70; this document is very wide-ranging and covers many aspects of designing and deploying an XML-based language

**XML as data type**

XML is beginning to appear as a first-class data type in other languages. The [ECMAScript for XML](http://en.wikipedia.org/wiki/E4X) (E4X) extension to the [ECMAScript](http://en.wikipedia.org/wiki/ECMAScript)/JavaScript language explicitly defines two specific objects (XML and XMLList) for JavaScript, which support XML document nodes and XML node lists as distinct objects and use a dot-notation specifying parent-child relationships. E4X is supported by the [Mozilla](http://en.wikipedia.org/wiki/Mozilla) 2.5+ browsers and Adobe [Actionscript](http://en.wikipedia.org/wiki/Actionscript), but has not been adopted more universally. Similar notations are used in Microsoft's [LINQ](http://en.wikipedia.org/wiki/LINQ) implementation for Microsoft .NET 3.5 and above, and in [Scala](http://en.wikipedia.org/wiki/Scala_%28programming_language%29) (which uses the Java VM). The open-source xmlsh application, which provides a Linux-like shell with special features for XML manipulation, similarly treats XML as a data type, using the <[ ]> notation. The [Resource Description Framework](http://en.wikipedia.org/wiki/Resource_Description_Framework) defines a data type rdf:XMLLiteral to hold wrapped, [canonical XML](http://en.wikipedia.org/wiki/Canonical_XML).

**XML Parsing Methodologies**

**SAX (Simple API for XML)**

**SAX** (**Simple API for XML**) is an event-based [sequential access](http://en.wikipedia.org/wiki/Sequential_access) [parser](http://en.wikipedia.org/wiki/Parser) [API](http://en.wikipedia.org/wiki/Application_programming_interface) developed by the XML-DEV mailing list for [XML](http://en.wikipedia.org/wiki/XML) documents.SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the [Document Object Model](http://en.wikipedia.org/wiki/Document_Object_Model) (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

## Definition

Unlike [DOM](http://en.wikipedia.org/wiki/Document_Object_Model), there is no formal specification for SAX. The [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) implementation of SAX is considered to be [normative](http://en.wikipedia.org/wiki/Normative#Standards_documents). It is used for state-independent processing of XML documents, in contrast to [StAX](http://en.wikipedia.org/wiki/StAX) that processes the documents state-dependently.

## Benefits

SAX parsers have certain benefits over DOM-style parsers. The quantity of [memory](http://en.wikipedia.org/wiki/Memory_(computers)) that a SAX parser must use in order to function is typically much smaller than that of a DOM parser. DOM parsers must have the entire tree in memory before any processing can begin, so the amount of memory used by a DOM parser depends entirely on the size of the input data. The memory footprint of a SAX parser, by contrast, is based only on the maximum depth of the XML file (the maximum depth of the XML tree) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the parsed tree itself. Because of the event-driven nature of SAX, processing documents can often be faster than DOM-style parsers. Memory allocation takes time, so the larger memory footprint of the DOM is also a performance issue.

Due to the nature of DOM, streamed reading from disk is impossible. Processing XML documents larger than main memory is also impossible with DOM parsers, but can be done with SAX parsers. However, DOM parsers may make use of [disk space as memory](http://en.wikipedia.org/wiki/Virtual_Memory) to sidestep this limitation.

## Drawbacks

The event-driven model of SAX is useful for XML parsing, but it does have certain drawbacks. Certain kinds of [XML validation](http://en.wikipedia.org/wiki/XML_validation) require access to the document in full. For example, a [DTD](http://en.wikipedia.org/wiki/Document_Type_Definition) IDREF attribute requires that there be an element in the document that uses the given string as a DTD ID attribute. To validate this in a SAX parser, one would need to keep track of every previously encountered ID attribute and every previously encountered IDREF attribute, to see if any matches are made. Furthermore, if an IDREF does not match an ID, the user only discovers this after the document has been parsed; if this linkage was important to building functioning output, then time has been wasted in processing the entire document only to throw it away.

Additionally, some kinds of XML processing simply require having access to the entire document. [XSLT](http://en.wikipedia.org/wiki/XSLT) and [XPath](http://en.wikipedia.org/wiki/XPath), for example, need to be able to access any node at any time in the parsed XML tree. While a SAX parser could be used to construct such a tree, the DOM already does so by design.

**Document Object Model(DOM)**

The [Document Object Model](http://en.wikipedia.org/wiki/Document_Object_Model) (DOM) is an [interface](http://en.wikipedia.org/wiki/User_interface)-oriented [application programming interface](http://en.wikipedia.org/wiki/Application_programming_interface) that allows for navigation of the entire document as if it were a tree of [node](http://en.wikipedia.org/wiki/Node_%28computer_science%29) [objects](http://en.wikipedia.org/wiki/Object_%28computer_science%29) representing the document's contents. A DOM document can be created by a parser, or can be generated manually by users (with limitations). Data types in DOM nodes are abstract; implementations provide their own [programming](http://en.wikipedia.org/wiki/Programming) language-specific [bindings](http://en.wikipedia.org/wiki/Language_binding). DOM implementations tend to be [memory](http://en.wikipedia.org/wiki/Memory) intensive, as they generally require the entire document to be loaded into memory and constructed as a tree of objects before access is allowed.

The Document Object Model (DOM) is a [cross-platform](http://en.wikipedia.org/wiki/Cross-platform) and [language](http://en.wikipedia.org/wiki/Programming_language)-independent convention for representing and interacting with [objects](http://en.wikipedia.org/wiki/Object_(computer_science)) in [HTML](http://en.wikipedia.org/wiki/HTML), [XHTML](http://en.wikipedia.org/wiki/XHTML) and [XML](http://en.wikipedia.org/wiki/XML) documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its [application programming interface](http://en.wikipedia.org/wiki/Application_programming_interface) (API).

The XML DOM defines a standard way for accessing and manipulating XML documents.

The DOM presents an XML document as a tree-structure.

Knowing the XML DOM is a must for anyone working with XML.

## XML DOM Tree Example



**Data binding**

Another form of XML processing API is [XML data binding](http://en.wikipedia.org/wiki/XML_data_binding), where XML data are made available as a hierarchy of custom, strongly typed classes, in contrast to the generic objects created by a [Document Object Model](http://en.wikipedia.org/wiki/Document_Object_Model) parser. This approach simplifies code development, and in many cases allows problems to be identified at compile time rather than run-time. Example data binding systems include the [Java Architecture for XML Binding](http://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding) (JAXB) and XML Serialization in .NET.

**Real World Usage**  
It doesn't improve the look of your web page and the lack of browser support means that you can't use it as an alternative to a server-side database. There are uses which have been developed, though, although it will take a lot more development to make XML a mainstream language.  
 XMLNews is a system which allows news stories to be stored as XML. By using tags like, and web pages and software systems can be developed which will take the XML data and will output it as a correctly formatted web page. In fact, the same story could be displayed on a WAP phone, news website, headlines news ticker, news e-mail, SMS message or in a piece of software, all from the same source file. As you can see, this creates a huge benefit, as a story can be written once by a journalist, but distributed around the world in many different formats. You can find more information at XMLNews.org.

**The Two Problems**  
There are two main reasons for the development of XML:

* Computers do not understand the information placed in them. For example there is no way for a search engine, or any other computer, to know that this is page contains the introduction part of an XML tutorial. All it is a collection of letters and numbers, with HTML formatting around it. The computer cannot even tell what on this page a heading is, what text is and what is an advert. This is the main problem which XML was designed to overcome. If a page or document is written in XML, a computer can understand exactly what it is about. As will probably be obvious, this has very major implications for search engine technology. If a search engine knew exactly what was on a page, it would be able to instantly provide the exact results a person was looking for, with no inaccurate matches and no half-relevant pages. This is just the revolution the over-bloated web needs.
* Web pages are not compatible across different devices. One of the major difficulties that web designers have today is that people are now accessing the pages from a variety of different devices. PCs, Macs, mobile phones, palmtop computers and even televisions. Because of this, web designers must now either produce their pages in several different formats to cope with this, or they must cut back on the design in order to have the page compatible across the different formats. Because XML is used to define what data means and not how it is displayed, it makes it very easy to use the same data on several different platforms.

**CONCLUSION**

Although XML still has a long way to go to become a mainstream programming language, it has great potential. XML is a new type of language which has been developed for the web which is different to any other type of scripting or programming language available before. Instead of being concerned with the processing and display of data, XML's primary purpose is to tell the computer what data entered actually means.

**REFERENCE**

* [www.w3.org/XML](http://www.w3.org/XML)
* <http://wikipedia.org/wiki/XML>