**XML**

Table of Contents

[XML vs HTML 3](#_Toc100513613)

[SGML 3](#_Toc100513614)

[XML Components 4](#_Toc100513615)

[Attributes 5](#_Toc100513616)

[Tree Structure 6](#_Toc100513617)

[Processing Instructions 7](#_Toc100513618)

[Comments 7](#_Toc100513619)

[Legal XML 8](#_Toc100513620)

[Logical Structure 9](#_Toc100513621)

[XML Namespaces 10](#_Toc100513622)

The **Extensible Markup Language** (XML) is a markup language similar to HTML, albeit with some key differences. Unlike HTML, XML is designed to **carry data** not display it.

<Course id="CSE 4635">  
 <WebArchitecture>  
 <Lectures>  
 <Lecture1>  
 Introduction to Web Architecture  
 </Lecture1>  
 <Lecture2>  
 Introduction to XML  
 </Lecture2>  
 </Lectures>  
 </WebArchitecture>  
</Course>

XML

As can be seen in the example above, XML uses **tags**. These tags are not pre-defined, but must be defined by the developer. The tags give meaning to the elements, thus making it human readable. Thus, XML is just a shell which can be extended upon by developers.

XML is a W3C recommendation.

## XML vs HTML

As discussed before, XML is used to store data, unlike HTML which not only stores data, but also the logic to display it. The data stored in an XML file can be shared between diverse applications. XML documents can be displayed using other technologies like CSS and XMLT.

## SGML

The **Standard General Markup Language** (SGML) is an ISO standard which provides a formal notation for the definition of generalized markup languages. SGML is not a language itself, but rather a meta-language which is used to define other markup languages.

SGML documents must consist of three parts:

* **Content** – This is the actual body of the document.
* **DTD** – This is the grammar or the syntax that is to be used in the document.
* **Stylesheet** – This describes how to render the content on an output device.

In the case of HTML, the DTD and the Stylesheet are essentially hardcoded into the browser itself. In the case of XML, the DTD is optional, but the stylesheet is not.

## XML Components

<bookstore>  
 <book categoryID="01">  
 <title>Learning XML</title>  
 <author>Erik T. Ray</author>  
 <year>2003</year>  
 <price>39.95</price>  
 </book>  
</bookstore>

XML

* **Elements** – An element begins with a start tag and ends with an end tag. Inside the two tags, we can have other elements or text. In the example above, book is an element, which contains other elements, while author is also an element, which contains just text.
* **Tag** – This refers to either the starting or the ending tag used to mark the beginning and the end of an element respectively.
* **Content** – Everything inside the opening and closing tags of an element.
* **Node** – This is a generic term used to refer to any XML objects like elements, attributes, comments, processing instructions or plain text.

## Attributes

Consider the following HTML element:

<img src="webarchi.jpg">

HTML

This defines an image which comes from a specific source. The path to the image file is provided as an **attribute**.

In XML, elements can also have attributes which are defined in the exact same way.

<book id="03">The Kite Runner</book>

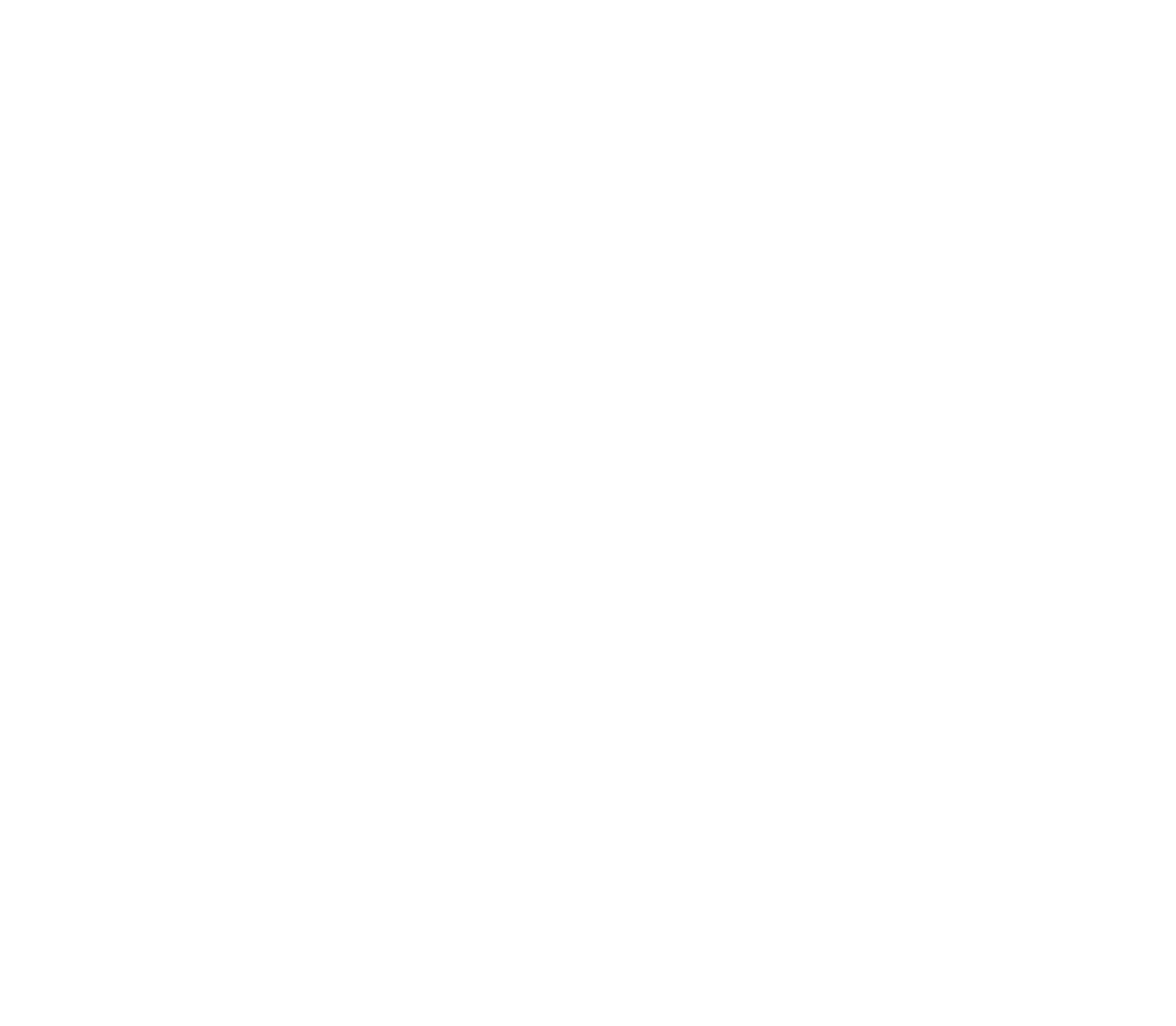
XML

The attributes can use either **double quotes** or **single quotes**.

Formally, an attribute is supposed to provide information that is not directly a part of the data. For example, an end-user will not be interested in the ID of a book in a library, just its name. However, a piece of software may require it to process the book. We need to judge on a case-by-case basis what information to put as attributes of an element and what information to put as sub-elements.

## Tree Structure

You may already have noticed that an XML document follows a **hierarchical structure** with a **single root element** and children under the element. Each element can have various siblings, children and a single parent. The root element is the only exception, since it cannot have any siblings.



## Processing Instructions

**Processing Instructions** provide information about how to process the XML document. They can be included anywhere in the document like this:

<?xml-stylesheet type="text/css" href="theme1.css"?>

XML

However, there is a specific type of processing instruction called an **XML declaration**, which must be provided at the very beginning of the XML document.

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

XML

## Comments

**Comments** can be included like this:

<!--  
Document: example.xml  
Author: someone  
-->

XML

## Legal XML

A **legal** XML document must have two attributes, well-formedness and validity.

A **well-formed** XML document follows the following rules:

1. There must be a **single root element** containing all other elements.
2. All non-empty elements must have a start and end tag.
3. **Empty elements** must be **self-closed** using either <tag/> or <tag></tag>.
4. Elements must be nested properly, i.e. a child element must be defined entirely, including both start and end tags, inside its parent element.
5. All attributes must be quoted and empty elements are allowed to have attributes.
6. Both the opening and closing tags must use the same case, since XML tags are **case-sensitive**.
7. Element names must not start with the letters ‘XML’ in any form, a number or a punctuation mark and cannot contain spaces.
8. White space is preserved.
9. **No markup characters** are allowed. If a markup character, such as <, must be used, then the **entity reference** for that character should be used instead, such as &lt; for <.

|  |  |  |
| --- | --- | --- |
| **Built-In Entity** | **Symbol** | **Meaning** |
| &lt; | < | Less than |
| &gt; | > | Greater than |
| &amp; | & | Ampersand |
| &apos; | ‘ | Apostrophe |
| &quot; | “ | Quotation Marks |

A **valid** XML document is one which, on top of being well-formed, also follows the rules set in the document’s DTD. However, we usually don’t care whether an XML document is valid or not, only that it is well-formed. If we are in a situation where the DTD becomes important, we will need special validating parsers, since most XML parsers are non-validating.

## Logical Structure

An XML document is structured in the following order:

1. Optional XML declaration
2. Optional DTD declaration
3. Optional comments and processing instructions
4. Root element’s start tag
5. All other elements, comments and processing instructions
6. Root element’s end tag

## XML Namespaces

Consider that we have the following two XML documents:

<!--Document: patients.xml-->  
<patient id="1">  
 <name>Someone</name>  
 <diagnosis>Fever</diagnosis>  
</patient>

XML

<!--Document: doctors.xml-->  
<doctor id="1">  
 <name>Someone Else</name>  
 <expertise>Medicine</expertise>  
</doctor>

XML

One element from each document has the **same attribute**, name. This sort of situation occurs often when retrieving XML documents from different sources and can lead to conflicts when trying to parse the documents. The parser will not be able to differentiate between the elements, even though humans will be able to tell that they refer to different things.

To deal with this, we can use **XML Namespaces**. This allows us to create a unique namespace based on a **Uniform Resources Identifier** (URI), give the namespace a prefix and apply that prefix to XML document elements.

<?xml version="1.0" encoding="UTF-8"?>  
<Course id="CSE 4635" xmlns:crs="course-information-system">  
 <crs:WebArchitecture>  
 <Lectures>  
 <Lecutre1>  
 Introduction to Web Architecture  
 </Lecutre1>  
 <Lecture2>  
 Introduction to XML  
 </Lecture2>  
 </Lectures>  
 </crs:WebArchitecture>  
</Course>

XML

Generally, an XML namespace is declared as an attribute at the root element of the XML document, as shown above. In the **DTD**, we simply add the keyword as though it were a part of the element’s name.

<!ELEMENT crs:WebArchitecture (Lectures)>  
<!ELEMENT Course (crs:WebArchitecture)>

DTD

We mentioned that we are using a URI. A URI can be of two types, a URL, which we have already seen, or a URN. A URN is a **Unique Resource Name**. In the above example, we created the URN course-information-system. Next, we set this URN to a **prefix**, crs. To inform any XML parser that we are creating a namespace, we must use the notation xmlns:prefix="unique-resource-name". We can then use the prefix with the other elements in the document.